
OpenFAST Documentation

Release v3.2.0

National Renewable Energy Laboratory

Aug 12, 2022

CONTENTS

1	This documentation	3
2	Installing OpenFAST	5
2.1	Download binaries	5
2.2	Compile from source	8
2.3	C++ API	16
2.4	FAST.Farm	16
2.5	Appendix	17
3	Testing OpenFAST	21
3.1	Unit tests	21
3.2	Regression tests	24
3.3	Obtaining and configuring the test suite	33
4	User Documentation	35
4.1	General	35
4.2	Module Documentation	59
4.3	Modularization Framework	403
4.4	Glue Code and Mesh Mapping	403
4.5	NWTC Subroutine Library	404
5	Developer Documentation	405
5.1	Development Philosophy and Guidelines	405
5.2	API Reference	424
5.3	Other Documentation	424
6	Licensing	425
7	Getting Help	427
8	Acknowledgements	429
	Bibliography	431
	Index	439

OpenFAST is a multi-physics, multi-fidelity tool for simulating the coupled dynamic response of wind turbines. Practically speaking, OpenFAST is the framework (or “glue code”) that couples computational modules for aerodynamics, hydrodynamics for offshore structures, control and electrical system (servo) dynamics, and structural dynamics to enable coupled nonlinear aero-hydro-servo-elastic simulation in the time domain. OpenFAST enables the analysis of a range of wind turbine configurations, including two- or three-blade horizontal-axis rotor, pitch or stall regulation, rigid or teetering hub, upwind or downwind rotor, and lattice or tubular tower. The wind turbine can be modeled on land or offshore on fixed-bottom or floating substructures.

Established in 2017, OpenFAST is an open-source software package that builds on FAST v8 (see [FAST v8 and the transition to OpenFAST](#)). The glue code and underlying modules are mostly written in Fortran (adhering to the 2003 standard), and modules can also be written in C or C++. It was created with the goal of being a community model developed and used by research laboratories, academia, and industry. It is managed by a dedicated team at the National Renewable Energy Lab. Our objective is to ensure that OpenFAST is well tested, well documented, and self-sustaining software. To that end, we are continually improving the documentation and test coverage for existing code, and we expect that new capabilities will include adequate testing and documentation. If you’d like to contribute, see the [Developer Documentation](#) and any open GitHub issues with the [Help Wanted](#) tag.

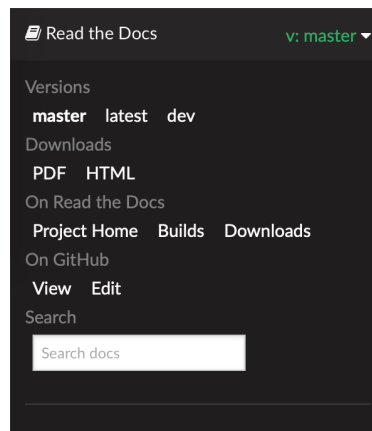
The following links provide more insight into OpenFAST as a software package:

- [OpenFAST Github Organization](#)
- [Github Repository](#)

Documentation Directory

THIS DOCUMENTATION

OpenFAST documentation is hosted on [readthedocs](#), and is automatically generated from both the [main](#) and [dev](#) branches whenever new commits are added. Clicking on the bar on the lower left corner of the page reveals a panel (see image below) containing options to select the branch of the repository, download the documentation other formats (PDF, HTML, EPub), and link to other relevant websites.



While OpenFAST developer documentation is being enhanced here, developers are encouraged to consult the legacy FAST v8 NWTC Programmer's Handbook. Instructions on obtaining and installing OpenFAST are available in *Installing OpenFAST*, and documentation for verifying an installation with the automated tests is at *Testing OpenFAST*.

The majority of this documentation is divided into two parts:

User Documentation

Directed towards end-users, this part provides detailed documentation regarding usage of the OpenFAST and its underlying modules, as well as theory and verification documentation.

Developer Documentation

The developer guide is targeted towards users wishing to extend the functionality provided within OpenFAST. Here you will find details regarding the code structure, API supported by various classes, and links to source code documentation extracted using Doxygen.

INSTALLING OPENFAST

Guidelines and procedures for obtaining precompiled binaries or compiling OpenFAST from source code are described here. While there are multiple ways to achieve the same outcome, the OpenFAST team has developed a comprehensive and well-thought out system for installation, so the methods described here are the only officially supported and maintained paths for obtaining an OpenFAST executable.

Most users of OpenFAST will not require modifying or compiling the source code. **For the simplest installation of OpenFAST without changing the source code**, refer to the table in the [Download binaries](#) section and read the corresponding documentation for specific instructions. For instructions on compiling, see [Compile from source](#).

2.1 Download binaries

For users who intend to run OpenFAST simulations without changing the source code, installation with precompiled binaries is recommended. The installation procedures are specific for each supported operating system, and the table below maps operating systems to the method for obtaining binaries. “Release” versions are well tested and stable versions of OpenFAST. A new release corresponds to a merge from the dev branch of the repository to the main branch along with a version tag. “Prerelease” versions contain the latest commits to the dev branch and may contain unstable code but will always have the latest features.

Operating System	Method	OpenFAST Version	Docs Section
Linux	Conda	Release, Prerelease	Conda Installation
macOS	Conda	Release, Prerelease	Conda Installation
macOS	Homebrew	Release	Homebrew Installation
Windows	GitHub Releases	Release	GitHub Releases

2.1.1 Conda Installation

OpenFAST releases are distributed through the [Anaconda](#) package manager via the [OpenFAST Conda Forge](#) channel for macOS and Linux. The installation includes

- OpenFAST glue-code executable
- Available module drivers
- C++ header files

The following commands describe how to create a new environment, install OpenFAST, and test the installation.

```
# Create a new conda environment
conda create -n openfast_env
```

(continues on next page)

(continued from previous page)

```
# Install OpenFAST through the Conda Forge channel
conda install -c conda-forge openfast

# Test OpenFAST
which openfast
openfast -v

# Test the HydroDyn driver
which hydrodyn_driver
hydrodyn_driver -v
```

Prereleases can be installed via conda by specifying the dev label, as shown below.

```
conda install -c conda-forge/label/dev openfast
```

These are always the latest commits to the dev branch of the repository and contain the latest changes to OpenFAST, but these builds are not as well tested as the full release versions.

2.1.2 Homebrew Installation

For macOS systems, OpenFAST releases are distributed through the [Homebrew](#) package manager. The installation includes only the OpenFAST glue-code executable.

To install with Homebrew and test the installation, use the following commands.

```
# Update Homebrew
brew update

# Install OpenFAST
brew search openfast
brew install openfast

# Test OpenFAST
which openfast
openfast -v
```

2.1.3 GitHub Releases

For Windows systems only, precompiled binaries are made available for each release on the OpenFAST GitHub [Releases](#) page. The binaries are compiled with the Intel Fortran compiler version 2020.

Important: The precompiled binaries require either the Intel Fortran compiler or the Intel MKL redistributable libraries, which are not by default included with the binaries. To configure the libraries, download the installers from the bottom of [this page](#). If you have a Command Prompt open, you will need to close it after installing the libraries in order for the changes to take effect. Admin privileges are required to install the Intel libraries.

The OpenFAST executables can be downloaded from the “Assets” dropdown in each Release. The two assets named “Source code” are not needed.

v2.0.0
0769598

Compare

Edit

OpenFAST v2.0.0

rafmudaf released this on Dec 5, 2018

OpenFAST v2.0.0
Breaking changes to the BeamDyn API have been introduced. Please refer to the BeamDyn input file documentation for updates.

BeamDyn

- Linearization
- Quasi steady start up
- Finite difference capability
- Computation cleanup for improved performance
- Expanded regression tests
- Bug fixes

AeroDyn

- DBEMT
- Bug fixes

Build system updates

- Parallel build enabled through CMake
- Improved debug flags
- Support for CMake-generated Visual Studio solution
- Single precision bug fixes

TurbSim bug fixes

- Documentation updates and improvements
- Increased robustness for interpolation of tower properties in the OpenFOAM module
- Added a nacelle model to the c++ glue code
- Support Bladed style brake torque input by array
- FAST renamed to OpenFAST in source

Assets 3

windows_openfast_v2.0.0.zip	41.1 MB
Source code (zip)	
Source code (tar.gz)	

The zipped file contains the following items:

File Name	Description
openfast_Win32.exe	32-bit single precision
openfast_x64.exe	64-bit single precision
openfast_x64_double.exe	64-bit double precision
Map_Win32.dll	32-bit MAP++ library
Map_x64.dll	64-bit MAP++ library
DISCON_DLLS/<64bit or Win32>/DISCON.dll	Controller library for NREL 5MW
DISCON_DLLS/<64bit or Win32>/DISCON_ITIBarge.dll	Controller library for NREL 5MW - ITI Barge
DISCON_DLLS/<64bit or Win32>/DISCON_OC3Hywind.dll	Controller library for NREL 5MW - OC3 Hywind

After extracting the contents, the OpenFAST executables can be tested by opening a command prompt, moving into the directory containing the executables, and running a simple test command:

```
cd C:\your\path\Desktop\openfast_binaries\
openfast_x64.exe /h
```

2.2 Compile from source

To compile from source code, the NREL OpenFAST team has developed an approach that uses CMake to generate build files for all platforms. Currently, CMake support for Visual Studio while doing active development is not well supported, so OpenFAST maintains a Visual Studio Solution giving Windows developers another option for writing code, compiling and debugging in a streamlined manner. See [Visual Studio Solution for Windows](#) for more information. If Visual Studio is not a requirement in Windows development, CMake is adequate. Background on CMake is given in [Understanding CMake](#), and procedures for configuring and compiling are given in [CMake with Make for Linux/macOS](#) and [CMake with Visual Studio for Windows](#).

Generally, the steps required to compile are:

1. Install Dependencies (Section [Section 2.2.1](#))
2. Configure the build system (Visual Studio: [Section 2.2.3](#), CMake: [Section 2.2.4](#))
3. Compile and test binaries (Visual Studio: [Section 2.2.3](#), CMake: [Section 2.2.5](#) and [Section 2.2.6](#))

2.2.1 Dependencies

Compiling OpenFAST from source requires additional libraries and tools that are not distributed with the OpenFAST repository. Each of the following components are required for the minimum OpenFAST compilation.

- C++, C, and Fortran compiler
- BLAS and LAPACK math library
- Build system

In many cases, these tools can be installed with a system's package manager (e.g. `homebrew` for macOS, `yum` for CentOS/Red Hat, or `apt` for Debian-based systems like Ubuntu). For Ubuntu and macOS, the following commands install all required dependencies.

System	Dependency Installation Command
Ubuntu 20.04	<code>apt install git cmake libblas-dev liblapack-dev gfortran-10 g++</code>
macOS 10.15	<code>brew install git cmake make openblas gcc</code>

If dependencies are downloaded from vendors directly, they must be installed in a standard location for your system so that the OpenFAST build systems can find them.

Compilers

Compiling OpenFAST requires a C, C++, and Fortran compiler. Though many options exist, the most common and best supported compilers are listed below.

Vendor / Compiler	Applicable systems	Minimum version	Link
GNU Compiler Collection (gfortran, gcc, g++)	macOS, Linux	4.6.0	https://gcc.gnu.org
Intel Compilers (ifort, icc)	All	2013	https://software.intel.com/content/www/us/en/develop/tools/oneapi/hpc-toolkit.html

Other compiler packages may work and can be well suited to a particular hardware, but their mileage may vary with OpenFAST. For instance, MinGW, CygWin, and LLVM are options for obtaining compilers on various systems. It is highly recommended to use the latest version of one of the above.

Math libraries

Math libraries with the BLAS and LAPACK interfaces are also required. All major options can be obtained as free downloads. The most common options are listed in the table below.

Library	Distributor	Open Source?	Link
BLAS/LAPACK	NetLib	Yes	http://www.netlib.org/blas/ , http://www.netlib.org/lapack/
BLAS/LAPACK	Open-BLAS	Yes	https://www.openblas.net
MKL	Intel	No	https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onemkl.html

Build tools

An environment-specific build system is required and may consist of a combination of the packages listed in the table below.

Package	Applicable systems	Minimum version	Link
CMake	All	3.0	https://cmake.org
GNU Make	macOS, Linux	1.8	https://www.gnu.org/software/make/
Visual Studio	Windows	2015	">https://visualstudio.microsoft.com>

For Windows, CMake may be used to generate a Visual Studio Solution that can then be used to compile OpenFAST. OpenFAST also contains a standalone Visual Studio project, see *Visual Studio Solution for Windows*.

For macOS and Linux, the recommended tools are CMake and GNU Make. CMake is used to generate Makefiles that are inputs to the GNU Make program. Other build tools exist for both Linux and macOS (Xcode, Ninja), but these are not well supported by the OpenFAST system.

2.2.2 Get the code

OpenFAST can be cloned (i.e., downloaded) from its [Github repository](https://github.com/OpenFAST/OpenFAST) via the command line:

```
git clone https://github.com/OpenFAST/OpenFAST.git
```

An archive of the source code can also be downloaded directly from these links:

- “main” branch - Stable release
- “dev” branch - Latest updates

2.2.3 Visual Studio Solution for Windows

A complete Visual Studio solution is maintained for working with the OpenFAST on Windows systems. The procedure for configuring the system and proceeding with the build process are documented in the following section:

Building OpenFAST on Windows with Visual Studio

These instructions are specifically for the standalone Visual Studio project at *openfast/vs-build*. Separate CMake documentation is provided for Windows users at [Section 2.2.6](#).

Prerequisites

1. A version of Visual Studio (VS).
 - Currently VS 2013 Professional and VS 2015 Community Edition have been tested with OpenFAST.
 - A list of Intel Fortran compatible VS versions and specific installation notes are found [here](#).
 - The included C/C++ project files for MAP++ and the Registry are compatible with VS 2013, but will upgrade seamlessly to a newer version of VS.
 - If you download and install [Visual Studio 2015 Community Edition](#), you will need to be sure and select the C/C++ component using the Customize option.
2. Intel Fortran Compiler
 - Currently only version 2017.1 has been tested with OpenFAST, but any newer version should be compatible.
 - You can download an Intel Fortran compiler [here](#).
 - Only install Intel Fortran after you have completed your Visual Studio installation.
3. Git for Windows
 - Download and install [git](#) for Windows.
4. Python 3.x for Windows (for regression/unit testing)
 - The testing framework of OpenFAST requires the use of Python.
 - Please see [Section 3](#) on testing OpenFAST for further information on this topic.
 - We have been working with Continuum's [Anaconda](#) installation of Python 3.6 for Windows.

Compiling OpenFAST

1. Open A command prompt, or git bash shell from the Start menu
2. Create a directory where you will clone OpenFAST repository (change code to your preferred name)

```
mkdir code  
cd code
```

3. Clone the OpenFAST repository

```
git clone https://github.com/openfast/openfast.git
```

This will create a directory called `openfast` within the `code` directory.

4. Using Windows Explorer, navigate to the directory `openfast\vs-build\Fast` and double-click on the `FAST.sln` Visual Studio solution file. This will open Visual Studio with the FAST solution and its associated projects.

NOTE: If you are using Visual Studio 2015 or newer, you will be asked to upgrade both the `Fast_Registry.vcxproj` and the `MAP_dll.vcxproj` files to a newer format. Go ahead and accept the upgrade on those files.

5. Select the desired Solution Configuration, such as Release, and the desired Solution Platform, such as x64 by using the drop down boxes located below the menubar.
6. Build the solution using the Build->Build Solution menu option.
7. If the solution built without errors, the executable will be located under the openfast\build\bin folder.

2.2.4 Understanding CMake

To more fully understand CMake and its methodology, visit this guide on [running CMake](#).

CMake is a build configuration system that creates files as input to a build tool like GNU Make, Visual Studio, or Ninja. CMake does not compile code or run compilers directly, but rather creates the environment needed for another tool to run compilers and create binaries. A CMake project is described by a series of files called `CMakeLists.txt` located in directories throughout the project. The main CMake file for OpenFAST is located at `openfast/CMakeLists.txt` and each module and glue-code has its own `CMakeLists.txt`; for example, AeroDyn and BeamDyn have one at `openfast/modules/aerodyn/CMakeLists.txt` and `openfast/modules/beamdyn/CMakeLists.txt`, respectively.

Running CMake

Running CMake and a build tool will create many files (text files and binaries) used in the various stages of the build. For this reason, a build folder should be created to contain all of the generated files associated with the build process. Here, an important file called `CMakeCache.txt` contains the user-defined settings for the CMake configuration. This file functions like memory storage for the build. It is initially created the first time the CMake command is run and populated with the initial settings. Then, any subsequent changes to the settings will be updated and stored there.

CMake can be executed in a few ways:

- Command line interface: `cmake`
- Command line curses interface: `ccmake`
- Official CMake GUI

The CMake GUI is only distributed for Windows, but it can be built from source for other platforms. OpenFAST's build process focuses on the command line execution of CMake for both the Linux/macOS and Windows terminals. The command line syntax to run CMake for OpenFAST is generally:

```
cmake <path-to-primary-CMakeLists.txt> [options]
```

Options

```
-D <var>[:<type>]=<value>    = Create or update a cmake cache entry.
```

For example, a common CMake command issued from the `openfast/build` directory is:

```
# cmake <path-to-primary-CMakeLists.txt> [options]
# where
#   <path-to-primary-CMakeLists.txt> is ".."
#   [options] can be
#       -DBUILD_SHARED_LIBS:BOOL=ON or
#       -DBUILD_SHARED_LIBS=ON

cmake .. -DBUILD_SHARED_LIBS=ON
```

The command line curses interface can be invoked similarly:

```
ccmake ..
```

The interface will be rendered in the terminal window and all navigation happens through keyboard inputs.

OpenFAST CMake options

CMake has a large number of general configuration variables available. A good resource for useful CMake variables is at this link: [GitLab CMake variables](#). The [CMake API documentation](#) is also helpful for searching through variables and determining the resulting action. Note that the CMake process should be well understood before customizing the general options.

The CMake options specific to OpenFAST and their default settings are:

BUILD_DOCUMENTATION	- Build documentation (Default: OFF)
BUILD_FASTFARM	- Enable FAST.Farm capabilities (Default: OFF)
BUILD_OPENFAST_CPP_API	- Enable building OpenFAST - C++ API (Default: OFF)
BUILD_OPENFAST_SIMULINK_API	- Enable building OpenFAST for use with Simulink.
↪ (Default: OFF)	
BUILD_SHARED_LIBS	- Enable building shared libraries (Default: OFF)
BUILD_TESTING	- Build the testing tree (Default: OFF)
CMAKE_BUILD_TYPE	- Choose the build type : Debug Release (Default: Release)
CMAKE_Fortran_MODULE_DIRECTORY	- Set the Fortran Modules directory
CMAKE_INSTALL_PREFIX	- Install path prefix, prepended onto install directories.
CODECOV	- Enable infrastructure for measuring code coverage.
↪ (Default: OFF)	
DOUBLE_PRECISION	- Treat REAL as double precision (Default: ON)
FPE_TRAP_ENABLED	- Enable Floating Point Exception (FPE) trap in compiler.
↪ options (Default: OFF)	
GENERATE_TYPES	- Use the openfast-registry to autogenerate types modules.
↪ (Default: OFF)	
OPENMP	- Enable OpenMP support (Default: OFF)
ORCA_DLL_LOAD	- Enable OrcaFlex library load (Default: OFF)
USE_DLL_INTERFACE	- Enable runtime loading of dynamic libraries (Default: ON)
↪ ON)	

Additional system-specific options may exist for a given system, but those should not impact the OpenFAST configuration. As mentioned above, the configuration variables are set initially but can be changed at any time. For example, the defaults may be accepted to initially configure the project, but then the settings may be configured individually:

```
# Initial configuration with the default settings
cmake ..

# Change the build to Debug mode rather than Release
cmake .. -DCMAKE_BUILD_TYPE=Debug

# Use dynamic linking rather than static linking
cmake .. -DBUILD_SHARED_LIBS=ON
```

The commands above are equivalent to having run this command the first time:

```
# Initial configuration in Debug mode with dynamic linking
cmake .. -DCMAKE_BUILD_TYPE=Debug -DBUILD_SHARED_LIBS=ON
```

CMAKE_BUILD_TYPE

This option allows to set the compiler optimization level and debug information. The value and its effect are listed in the table below.

CMAKE_BUILD_TYPE	Effect
Release	-O3 optimization level
RelWithDebInfo	-O2 optimization level with -g flag for debug info
MinSizeRel	-O1 optimization level
Debug	No optimization and -g flag for debug info; additional debugging flags: -fcheck=all -pedantic -fbacktrace

Use **Debug** during active development to add debug symbols for use with a debugger. This build type also adds flags for generating runtime checks that would otherwise result in undefined behavior. **MinSizeRel** adds basic optimizations and targets a minimal size for the generated executable. The next level, **RelWithDebInfo**, enables vectorization and other more aggressive optimizations. It also adds debugging symbols and results in a larger executable size. Finally, use **Release** for best performance at the cost of increased compile time.

This flag can be set with the following command:

```
cmake .. -DCMAKE_BUILD_TYPE=RelWithDebInfo
```

CMAKE_INSTALL_PREFIX

This flag sets the location of the compiled binaries when the build tool runs the `install` command. It should be a full path in a carefully chosen location. The binaries will be copied into `include`, `lib`, and `bin` subfolders under the value of this flag. The default is to install binaries within the repository in a folder called `install`.

This flag can be set with the following command:

```
cmake .. -DCMAKE_INSTALL_PREFIX="/usr/local/"
```

Setting the build tool

CMake can target a variety of build tools or *generators*. To obtain a list of available generators on the current system, run with the empty generator flag, select the target from the list, and rerun with the generator flag populated:

```
# Run with the empty -G flag to get a list of available generators
cmake .. -G

# CMake Error: No generator specified for -G
#
# Generators
# * Unix Makefiles           = Generates standard UNIX makefiles.
# Ninja                     = Generates build.ninja files.
# Xcode                     = Generate Xcode project files.
# CodeBlocks - Ninja        = Generates CodeBlocks project files.
# CodeBlocks - Unix Makefiles = Generates CodeBlocks project files.
# CodeLite - Ninja          = Generates CodeLite project files.
# CodeLite - Unix Makefiles = Generates CodeLite project files.
# Sublime Text 2 - Ninja    = Generates Sublime Text 2 project files.
```

(continues on next page)

(continued from previous page)

```
# Sublime Text 2 - Unix Makefiles
#                                     = Generates Sublime Text 2 project files.
# Kate - Ninja                       = Generates Kate project files.
# Kate - Unix Makefiles              = Generates Kate project files.
# Eclipse CDT4 - Ninja               = Generates Eclipse CDT 4.0 project files.
# Eclipse CDT4 - Unix Makefiles= Generates Eclipse CDT 4.0 project files.

# Choose one from the list above and pass it as an argument after -G
# NOTE: wrap this is in quotes!
cmake .. -G"Sublime Text 2 - Ninja"
```

Note: If the chosen generator name contains spaces, be sure to wrap it in quotes.

Math libraries

The CMake project is configured to search for the required math libraries in default locations. However, if math libraries are not found, they can be specified directly to CMake. The two required libraries are BLAS and LAPACK, and their location can be passed to CMake with this command syntax:

```
cmake .. -DBLAS_LIBRARIES="/path/to/blas" -DLAPACK_LIBRARIES="/path/to/lapack"
```

The paths given should be to the directory which contains the libraries, not to the libraries themselves.

2.2.5 CMake with Make for Linux/macOS

After installing all dependencies and reading *Understanding CMake*, proceed with configuring OpenFAST. The CMake project is well developed for Linux and macOS systems, so the default settings should work as given. These settings should only be changed when a custom build is required.

The full procedure for installing dependencies, configuring CMake and compiling with GNU Make on Linux and macOS systems is given below.

```
# For Ubuntu Linux, this installs all dependencies
apt install git cmake libblas-dev liblapack-dev gfortran-10 g++

# For macOS using Homebrew, this installs all dependencies
brew install git cmake make openblas gcc

# Clone the repository from GitHub using git
git clone https://github.com/OpenFAST/OpenFAST.git

# Move into the OpenFAST directory
cd OpenFAST

# Create the build directory and move into it
mkdir build
cd build

# Execute CMake with the default options;
```

(continues on next page)

(continued from previous page)

```
# this step creates the Makefiles
cmake ..

# Execute the Make-help command to list all available targets
make help

# Choose a particular target or give no target to compile everything
make hydrodyn_driver
# or
make openfast
# or
make

# Test the compiled binary, for example
./glue-codes/openfast/openfast -v
./modules/hydrodyn/hydrodyn_driver -v

# Move the binaries and other run-time files to the install location
# The default is `openfast/install`
make install
```

Tip: Compile in parallel by adding “-jN” to the make command where N is the number of parallel processes to use; i.e. `make -j4 openfast`.

This will build the OpenFAST project in the build directory. Binaries are located in `openfast/build/glue-codes/` and `openfast/build/modules/`. Since all build-related files are located in the build directory, a new fresh build process can be accomplished by simply deleting the build directory and starting again.

2.2.6 CMake with Visual Studio for Windows

After installing all dependencies and reading *Understanding CMake*, proceed with configuring OpenFAST. The result of this configuration process will be a Visual Studio solution which will be fully functional for compiling any of the targets within OpenFAST. However, this method lacks support for continued active development. Specifically, any settings that are configured in the Visual Studio solution directly will be lost any time CMake is executed. Therefore, this method should only be used to compile binaries, and the procedure described in *Visual Studio Solution for Windows* should be used for active OpenFAST development on Windows.

The procedure for configuring CMake and compiling with Visual Studio on Windows systems is given below.

```
# Clone the repository from GitHub using git
git clone https://github.com/OpenFAST/OpenFAST.git

# Move into the OpenFAST directory
cd OpenFAST

# Create the build directory and move into it
mkdir build
cd build

# Execute CMake with the default options and a specific Visual Studio version
```

(continues on next page)

(continued from previous page)

```
# and build architecture. For a list of available CMake generators, run
# ``cmake .. -G``.
# This step creates the Visual Studio solution.
cmake .. -G "Visual Studio 14 2015 Win64"

# Open the generated Visual Studio solution
start OpenFAST.sln
```

Visual Studio will open a solution containing all of the OpenFAST projects, and any module library, module driver, or glue-code can be compiled from there. The compiled binaries are located within a directory determined by the Visual Studio build type (Release, Debug, or RelWithDebInfo) in `openfast/build/glue-codes/` and `openfast/build/modules/`. For example, the OpenFAST executable will be located at `openfast/build/glue-codes/Release/openfast.exe` when compiling in *Release* mode.

The CMake-generated Visual Studio build is not currently fully functional. Any configurations made to the Solution in the Visual Studio UI will be lost when CMake is executed, and this can happen whenever a change is made to the structure of the file system or if the CMake configuration is changed. It is recommended that this method **not** be used for debugging or active development on Windows. Instead, see *Visual Studio Solution for Windows*.

2.3 C++ API

When compiling the C++ API, the following additional dependencies are required:

- HDF5
- yaml-cpp
- libxml++

The C++ API is compiled only with CMake and it is possible to hint to CMake where to find some dependencies. The following commands configure CMake and compile the C++ interface.

```
# Enable compiling the C++ API
cmake .. -DBUILD_OPENFAST_CPP_API:BOOL=ON -DBUILD_SHARED_LIBS:BOOL=ON

# If CMake doesn't find HDF5, provide a hint
cmake .. -DHDF5_ROOT:STRING=/usr/lib/

# Compile the C++ API
make openfastcpplib
```

2.4 FAST.Farm

The FAST.Farm glue-code is included in the CMake project similar to the OpenFAST glue-code. See *Compile from source* for a full description on installing dependencies, configuring the project, and compiling. FAST.Farm is enabled in the CMake project with an additional flag:

```
# Enable compiling FAST.Farm
cmake .. -DBUILD_FASTFARM:BOOL=ON
```

(continues on next page)

(continued from previous page)

```
# Compile FAST.Farm
make FAST.Farm
```

OpenMP-Fortran is an additional dependency for FAST.Farm. These libraries can be installed with any package manager for macOS and Linux or through the Intel oneAPI distributions.

2.5 Appendix

The following are additional methods for installation which may not be fully test or may be deprecated in the future.

2.5.1 Building OpenFAST with Spack

The process to build and install OpenFAST with [Spack](#) on Linux or macOS is described here.

Dependencies

OpenFAST has the following dependencies:

- LAPACK libraries. Users should set `BLAS_LIBRARIES` and `LAPACK_LIBRARIES` appropriately for CMake if the library isn't found in standard paths. Use *BLASLIB* as an example when using Intel MKL.
- For the optional C++ API, [HDF5](#) (provided by `HDF5_ROOT`) and [yaml-cpp](#) (provided by `YAML_ROOT`)
- For the optional testing framework, Python 3+ and Numpy

Building OpenFAST Semi-Automatically Using Spack on macOS or Linux

The following describes how to build OpenFAST and its dependencies mostly automatically on macOS using [Spack](#). This can also be used as a template to build OpenFAST on any Linux system with Spack.

These instructions were developed on macOS 10.11 with the following tools installed via Homebrew:

- GCC 6.3.0
- CMake 3.6.1
- pkg-config 0.29.2

Step 1

Checkout the official Spack repo from github (we will checkout into `${HOME}`):

```
cd ${HOME} && git clone https://github.com/LLNL/spack.git
```

Step 2

Add Spack shell support to your `.profile` by adding the lines:

```
export SPACK_ROOT=${HOME}/spack
. $SPACK_ROOT/share/spack/setup-env.sh
```

Step 3

Copy the <https://raw.githubusercontent.com/OpenFAST/openfast/dev/share/spack/package.py> file to your installation of Spack:

```
mkdir ${SPACK_ROOT}/var/spack/repos/builtin/packages/openfast
cd ${SPACK_ROOT}/var/spack/repos/builtin/packages/openfast
wget --no-check-certificate https://raw.githubusercontent.com/OpenFAST/openfast/dev/
↪share/spack/package.py
```

Step 4

Try `spack info openfast` to see if Spack works. If it does, check the compilers you have available by:

```
machine:~ user$ spack compilers
==> Available compilers
-- gcc -----
gcc@6.3.0  gcc@4.2.1

-- clang -----
clang@8.0.0-apple  clang@7.3.0-apple
```

Step 5

Install OpenFAST with your chosen version of GCC:

```
spack install openfast %gcc@6.3.0
```

To install OpenFAST with the C++ API, do:

```
spack install openfast+cxx %gcc@6.3.0
```

That should be it! Spack will automatically use the most up-to-date dependencies unless otherwise specified. For example to constrain OpenFAST to use some specific versions of dependencies you could issue the Spack install command:

```
spack install openfast %gcc@6.3.0 ^hdf5@1.8.16
```

The executables and libraries will be located at

```
spack location -i openfast
```

Add the appropriate paths to your `PATH` and `LD_LIBRARY_PATH` to run OpenFAST.

2.5.2 Building OpenFAST on Windows with CMake and Cygwin 64-bit

WARNING: This build process takes a significantly long amount of time. If GNU tools are not required, it is recommended that Windows users see one of the following sections:

- *Download binaries*
- *CMake with Visual Studio for Windows*
- *Building OpenFAST on Windows with Visual Studio.*

Installing prerequisites

1. Download and install [Cygwin 64-bit](#). You will need to Run as Administrator to complete the installation process.
 - Choose `Install` from `internet`
 - Choose the default install location
 - Choose the default package download location
 - Choose `Direct connection`
 - Choose a download site
 - See next step for `select packages`. Alternately, you can skip this step and run `setup-x86_64.exe` anytime later to select and install required software.
2. Select packages necessary for compiling OpenFAST. Choose `binary` packages and not the `source` option.
 - Choose `Category` view, we will be installing packages from `Devel` and `Math`
 - From `Devel` mark the following packages for installation
 - `cmake`
 - `cmake-doc`
 - `cmake-gui`
 - `cygwin-devel`
 - `gcc-core`
 - `gcc-fortran`
 - `gcc-g++`
 - `git`
 - `make`
 - `makedepend`
 - From `Math` mark the following packages for installation
 - `liblapack-devel`
 - `libopenblas`
 - To run the test suite, install these optional packages from `Python`:
 - `python3`
 - `Python3-numpy`

- Click **Next** and accept all additional packages that the setup process requests to install to satisfy dependencies
3. It is *recommended* that you reboot the machine after installing Cygwin and all the necessary packages.

Compiling OpenFAST

From here, pick up from the Linux with CMake instructions at [CMake with Make for Linux/macOS](#).

Other tips

- If you would like to run `openfast.exe` from the `cmd` terminal, then you must add the `C:\cygwin64\lib\lapack` and `C:\cygwin64\home\<USERNAME>\software\bin` to your `%PATH%` variable in environment setting. Replace `<USERNAME>` with your account name on Windows system.
- It is suggested to compile with optimization level 2 for Cygwin. Do this by changing the build mode in the `cmake` command

```
cmake .. -DCMAKE_BUILD_TYPE=RelWithDebInfo
```

TESTING OPENFAST

OpenFAST is a complex software with many moving parts. In order to maintain stability in new and existing code, a test suite is included directly in the source code. Two primary levels of tests exist: regression tests at the highest level and unit tests at the lowest level. The regression tests compare locally generated results with stored “baseline” results. These tests give an indication of whether the full-system or sub-system response has changed. The unit tests focus on a single subroutine or code block. These tests need not be physically realistic and focus on the mathematics and exercising of an algorithm. The objective of the included tests is to quickly catch bugs or unexpected changes in results. Additionally, the tests can help programmers design their module and subroutine interfaces in a sustainable and maintainable manner.

All of the necessary files corresponding to the regression tests are contained in the `reg_tests` directory. The unit test framework is housed in `unit_tests` while the actual tests are contained in the directory corresponding to the tested module.

The OpenFAST GitHub repository uses [GitHub Actions](#) to automatically execute the test suite for new commits and pull requests. This cloud computing resource is available to all GitHub users and is highly recommended as part of the development workflow. After enabling GitHub Actions in an OpenFAST repository, simply pushing new commits will trigger the tests.

3.1 Unit tests

In a software package as dynamic and collaborative as OpenFAST, confidence in multiple layers of code is best accomplished with a strong system of unit tests. Through robust testing practices, the entire OpenFAST community can understand the intention behind code blocks and debug or expand functionality quicker and with more confidence and stability.

Unit testing in OpenFAST modules is accomplished through [pFUnit](#). This framework provides a Fortran abstraction to the popular [xUnit](#) structure. pFUnit is compiled along with OpenFAST through CMake when the CMake variable `BUILD_TESTING` is turned on.

The BeamDyn and NWTC Library modules contain some sample unit tests and should serve as a reference for future development and testing.

3.1.1 Dependencies

The following packages are required for unit testing:

- Python 3.7+
- CMake
- pFUnit - Included in OpenFAST repo through a git-submodule

3.1.2 Compiling

Compiling the unit tests is handled with CMake similar to compiling OpenFAST in general. After configuring CMake with BUILD_TESTING turned on, new build targets are created for each module included in the unit test framework named [module]_utest. Then, make the target to test:

```
cmake .. -DBUILD_TESTING=ON
make beamdyn_utest
```

This creates a unit test executable at openfast/build/unit_tests/beamdyn/beamdyn_utest.

3.1.3 Executing

To execute a module's unit test, simply run the unit test binary. For example:

```
>>>$ ./openfast/build/unit_tests/beamdyn/beamdyn_utest
.....
Time:          0.018 seconds

OK
(14 tests)
```

pFUnit will display a . for each unit test successfully completed and a F for each failing test. If any tests do fail, the failure criteria will be displayed listing which particular value caused the failure. Failure cases display the following output:

```
>>>$ ./unit_tests/beamdyn/beamdyn_utest
.....F.....
Time:          0.008 seconds

Failure
in:
test_BD_CrvMatrixH_suite.test_BD_CrvMatrixH
Location:
[test_BD_CrvMatrixH.F90:48]
simple rotation with known parameters: Pi on xaxis expected +0.5000000 but found: +0.
↳4554637; difference: |+0.4453627E-01| > tolerance:+0.1000000E-13; first difference_
↳at element [1, 1].

FAILURES!!!
Tests run: 13, Failures: 1, Errors: 0
Note: The following floating-point exceptions are signalling: IEEE_INVALID_FLAG IEEE_
↳DIVIDE_BY_ZERO
```

(continues on next page)

(continued from previous page)

```
ERROR STOP *** Encountered 1 or more failures/errors during testing. ***
```

```
Error termination. Backtrace:
```

```
#0 0x1073b958c
#1 0x1073ba295
#2 0x1073bb1b6
#3 0x106ecdd4f
#4 0x1063fabee
#5 0x10706691e
```

3.1.4 Adding unit tests

Unit tests should be included for each new, *testable* code block (subroutine or function). What is testable is the discretion of the developer, but an element of the pull request review process will be evaluating test coverage.

New unit tests can be added to a `tests` directory alongside the `src` directory included in each module. For example, a module directory may be structured as

```
openfast/
├── modules/
│   └── sampledyn/
│       ├── src/
│       │   ├── SampleDyn.f90
│       │   └── SampleDyn_Subs.f90
│       └── tests/
│           ├── test_SampleDyn_Subroutine1.F90
│           ├── test_SampleDyn_Subroutine2.F90
│           └── test_SampleDyn_Subroutine3.F90
```

Each unit test must be contained in a unique file called `test_[SUBROUTINE].F90` where `[SUBROUTINE]` is the code block being tested. The new files should contain a Fortran *module* which itself contains a Fortran *subroutine* for each specific test case. Generally, multiple tests will be required to fully test one subroutine.

Finally, update the CMake configuration for building a module's unit test executable by copying an existing unit test CMake configuration into a new module directory:

```
cp -r openfast/unit_tests/beamdyn openfast/unit_tests/[module]
```

Then, modify the new `CMakeLists.txt` with the appropriate list of test subroutines and module name variables.

For reference, a template unit test file is included at `openfast/unit_tests/test_SUBROUTINE.F90`. Each unit test should fully test the target code block. If full test coverage is not easily achievable, it may be an indication that refactoring would be beneficial.

Some useful topics to consider when developing and testing for OpenFAST are:

- Test driven development
- Separation of concerns
- pFUnit usage

3.2 Regression tests

The regression test executes a series of test cases which intend to fully describe OpenFAST and its module's capabilities. Jump to one of the following sections for instructions on running the regression tests:

- *Executing with Python driver*
- *Executing with CTest*
- *Regression test examples*
- *Windows with Visual Studio regression test*

Each locally computed result is compared to a static set of baseline results. To account for system, hardware, and compiler differences, the regression test attempts to match the current machine and compiler type to the appropriate solution set from these combinations:

Operating System	Compiler	Hardware
macOS 10.15	GNU 10.2	2020 MacbookPro
Ubuntu 20.04	Intel oneAPI	Docker
Ubuntu 20.04	GNU 10.2	Docker
Windows 10	Intel oneAPI	Dell Precision 3530

The compiler versions, specific math libraries, and more info on hardware used to generate the baseline solutions are documented in the [r-test repository documentation](#). Currently, the regression test supports only double precision builds.

The regression test system can be executed with CMake (through its included test driver, CTest) or manually with a custom Python driver. Both systems provide similar functionality with respect to testing, but CTest integration provides access to multithreading, automation, and test reporting via CDash. Both modes of execution require some configuration as described in the following sections.

In both modes of execution a directory is created in the build directory called `reg_tests` where all of the input files for the test cases are copied and all of the locally generated outputs are stored. Ultimately, both CTest and the manual execution program call a series of Python scripts and libraries in `reg_tests` and `reg_tests/lib`. One such script is `lib/pass_fail.py` which reads the output files and computes a norm on each channel reported. If the maximum norm is greater than the given tolerance, that particular test is reported as failed. The failure criteria is outlined below.

```

difference = abs(testData - baselineData)
for i in nChannels:
    if channelRange < 1:
        norm[i] = MaxNorm( difference[:,i] )
    else:
        norm[i] = MaxNorm( difference[:,i] ) / channelRange

if max(norm) < tolerance:
    pass = True
else:
    pass = False

```


3.2.1 Dependencies

The following packages are required for regression testing:

- Python 3.7+
- Numpy
- CMake and CTest (Optional)
- Bokeh 1.4 (Optional)

3.2.2 Executing with Python driver

The regression test can be executed manually with the included driver at `openfast/reg_tests/manualRegressionTest.py`. This program reads a case list file at `openfast/reg_tests/r-test/glue-codes/openfast/CaseList.md`. Cases can be removed or ignored by starting that line with a `#`. The driver program includes multiple optional flags which can be obtained by executing with the help option:

```
>>>$ python manualRegressionTest.py -h
usage: manualRegressionTest.py [-h] [-p [Plotting-Flag]] [-n [No-Execution]]
                               [-v [Verbose-Flag]] [-case [Case-Name]]
                               OpenFAST System-Name Compiler-Id Test-Tolerance

Executes OpenFAST and a regression test for a single test case.

positional arguments:
OpenFAST                path to the OpenFAST executable
System-Name             current system's name: [Darwin, Linux, Windows]
Compiler-Id            compiler's id: [Intel, GNU]
Test-Tolerance          tolerance defining pass or failure in the regression
                        test

optional arguments:
-h, --help              show this help message and exit
-p [Plotting-Flag], -plot [Plotting-Flag]
                        bool to include plots in failed cases
-n [No-Execution], -no-exec [No-Execution]
                        bool to prevent execution of the test cases
-v [Verbose-Flag], -verbose [Verbose-Flag]
                        bool to include verbose system output
-case [Case-Name]       single case name to execute
```

Note: For the NREL 5MW turbine test cases, an external ServoDyn controller must be compiled and included in the appropriate directory or all NREL 5MW cases will fail without starting. More information is available in the documentation for the [r-test repository](#), but be aware that these three DISCON controllers must exist

```
openfast/build/reg_tests/glue-codes/openfast/5MW_Baseline/ServoData/DISCON.dll
openfast/build/reg_tests/glue-codes/openfast/5MW_Baseline/ServoData/DISCON_ITIBarge.dll
openfast/build/reg_tests/glue-codes/openfast/5MW_Baseline/ServoData/DISCON_OC3Hywind.dll
```

3.2.3 Executing with CTest

CTest is included with CMake and is primarily a set of preconfigured targets and commands. To use the CTest driver for the regression test, execute CMake as described in *Installing OpenFAST*, but with this additional flag: `-DBUILD_TESTING=ON`.

The regression test specific CMake variables are

```
BUILD_TESTING
CTEST_OPENFAST_EXECUTABLE
CTEST_[MODULE]_EXECUTABLE where [MODULE] is the module name
CTEST_PLOT_ERRORS
CTEST_REGRESSION_TOL
```

Some additional resources that are required for the full regression test suite are included in the CMake project. Specifically, external ServoDyn controllers must be compiled for a given system and placed in a particular location. Thus, be sure to execute the build command with the `install` target:

```
# Configure CMake with testing enabled and accept the default
# values for all other test-specific CMake variables
cmake .. -DBUILD_TESTING=ON

# Build and install
make install
```

Note: REMINDER: For the NREL 5MW turbine test cases, an external ServoDyn controller must be compiled and included in the appropriate directory or all NREL 5MW cases will fail without starting. More information is available in the documentation for the *r-test repository*, but be aware that these three DISCON controllers must exist

```
openfast/build/reg_tests/glue-codes/openfast/5MW_Baseline/ServoData/DISCON.dll
openfast/build/reg_tests/glue-codes/openfast/5MW_Baseline/ServoData/DISCON_ITIBarge.dll
openfast/build/reg_tests/glue-codes/openfast/5MW_Baseline/ServoData/DISCON_OC3Hywind.dll
```

After CMake configuration and compiling, the automated regression test can be executed by running either of the commands `make test` or `ctest` from the build directory. If the entire OpenFAST package is to be built, CMake will configure CTest to find the new binary at `openfast/build/glue-codes/openfast/openfast`. However, if the intention is to build only the test suite, the OpenFAST binary should be specified in the CMake configuration under the `CTEST_OPENFAST_EXECUTABLE` flag. There is also a corresponding `CTEST_[MODULE]_NAME` flag for each module included in the regression test.

When driven by CTest, the regression test can be executed by running various forms of the command `ctest` from the build directory. The basic commands are:

```
# Run the entire regression test
ctest

# Disable actual execution of tests;
# this is helpful in formulating a particular ctest command
ctest -N

# Run the entire regression test with verbose output
ctest -V
```

(continues on next page)

(continued from previous page)

```
# Run tests by name where TestName is a regular expression (regex)
ctest -R [TestName]

# Run all tests with N tests executing in parallel
ctest -j [N]
```

Each regression test case contains a series of labels associating all of the modules used. The labeling can be seen in the test instantiation in `reg_tests/CTestList.cmake` or with the command:

```
# Print all available test labels
ctest --print-labels
```

The test cases corresponding to a particular label can be executed with this command:

```
# Filter the test cases corresponding to a particular label
ctest -L [Label]
```

Flags can be compounded making useful variations such as

```
# Run all cases that use AeroDyn14 with verbose output
ctest -V -L aerodyn14

# Run all cases that use AeroDyn14 in 16 concurrent processes
ctest -j 16 -L aerodyn14

# Run the case with name "5MW_DLL_Potential_WTurb" with verbose output
ctest -V -R 5MW_DLL_Potential_WTurb

# List all tests with the "beamdyn" label
ctest -N -L beamdyn

# List the labels included in all tests matching the regex "bd"
ctest -N -R bd --print-labels
```

The automated regression test writes new files only into the build directory. Specifically, all locally generated solutions are located in the corresponding glue-code or module within `openfast/build/reg_tests`. The baseline solutions contained in `openfast/reg_tests/r-test` are strictly read and are not modified by the automated process.

3.2.4 Regression test examples

The following examples illustrate methods of running the regression tests on Unix-based systems. However, similar procedures can be used on Windows with CMake and CTest. An alternate method of running the regression tests on Windows is given in *Detailed example of running on Windows*.

Compile OpenFAST and execute with CTest

The following example assumes the user is starting completely from scratch. The commands below download the source code, configure the OpenFAST project with CMake, compile all executables, and execute the full regression test suite.

```
# Download the source code from GitHub
#   Note: The default branch is 'main'
git clone --recursive https://github.com/openfast/openfast.git
cd openfast

# If necessary, switch to another target branch and update r-test
git checkout dev
git submodule update

# Create the build and install directories and move into build
mkdir build install && cd build

# Configure CMake for testing
# - BUILD_TESTING - turn ON
# - CTEST_OPENFAST_EXECUTABLE - accept the default
# - CTEST_[MODULE]_EXECUTABLE - accept the default
cmake .. -DBUILD_TESTING=ON

# Compile and install
make install

# Execute the full test suite with 4 concurrent processes
ctest -j4
```

Configure with CMake and a given executable

This example assumes the user has a fully functional OpenFAST executable available along with any necessary libraries, but does not have the source code repository downloaded. This might be the case when executables are distributed within an organization or downloaded from an [OpenFAST Release](#). Here, nothing will be compiled, but the test suite will be configured with CMake for use with the CTest command.

```
# Download the source code from GitHub
#   Note: The default branch is 'main'
git clone --recursive https://github.com/openfast/openfast.git
cd openfast

# If necessary, switch to another target branch and update r-test
git checkout dev
git submodule update
```

(continues on next page)

(continued from previous page)

```
# Create the build directory and move into it
mkdir build && cd build

# Configure CMake with openfast/reg_tests/CMakeLists.txt for testing
# - BUILD_TESTING - turn ON
# - CTEST_OPENFAST_EXECUTABLE - provide a path
# - CTEST_[MODULE]_EXECUTABLE - provide a path
cmake ../reg_tests \
  -DBUILD_TESTING=ON \
  -DCTEST_OPENFAST_EXECUTABLE=/home/user/Desktop/openfast_executable \
  -DCTEST_BEAMDYN_EXECUTABLE=/home/user/Desktop/beamdyn_driver

# Install required files
make install

# Execute the full test suite with 4 concurrent processes
ctest -j4
```

Python driver with a given executable

This example assumes the user has a fully functional OpenFAST executable available along with any necessary libraries, but does not have the source code repository downloaded. This might be the case when executables are distributed within an organization or downloaded from an [OpenFAST Release](#). Nothing will be compiled, but the test suite will be executed with the included Python driver.

```
# Download the source code from GitHub
# Note: The default branch is 'main'
git clone --recursive https://github.com/openfast/openfast.git
cd openfast

# If necessary, switch to another target branch and update r-test
git checkout dev
git submodule update

# Execute the Python driver
cd reg_tests
python manualRegressionTest.py -h
# usage: manualRegressionTest.py [-h] [-p [Plotting-Flag]] [-n [No-Execution]]
#                                     [-v [Verbose-Flag]] [-case [Case-Name]]
#                                     OpenFAST System-Name Compiler-Id Test-Tolerance
#
# Executes OpenFAST and a regression test for a single test case.
#
# positional arguments:
#   OpenFAST           path to the OpenFAST executable
#   System-Name        current system's name: [Darwin,Linux,Windows]
#   Compiler-Id        compiler's id: [Intel,GNU]
#   Test-Tolerance     tolerance defining pass or failure in the regression
#                       test
#
# optional arguments:
```

(continues on next page)

(continued from previous page)

```
# -h, --help          show this help message and exit
# -p [Plotting-Flag], -plot [Plotting-Flag]
#                      bool to include plots in failed cases
# -n [No-Execution], -no-exec [No-Execution]
#                      bool to prevent execution of the test cases
# -v [Verbose-Flag], -verbose [Verbose-Flag]
#                      bool to include verbose system output
# -case [Case-Name]   single case name to execute

python manualRegressionTest.py \
  ..\build\bin\openfast_x64_Double.exe \
  Windows \
  Intel \
  1e-5
```

Detailed example of running on Windows

The *Python driver with a given executable* example can be used for running the regression tests on a Windows computer. However, a more detailed, step-by-step description is given in *Windows with Visual Studio regression test*.

Windows with Visual Studio regression test

- 1) Clone the openfast repo and initialize the testing database
 - a) Open a git command shell window (like git bash)
 - b) Change your working directory to the location above where you want your local repo to be located (the repo will be placed into a folder called openfast at this location)
 - c. Type: `git clone https://github.com/openfast/openfast.git` (this creates a local version of the openfast repo on your computer). You should see something like this:

```
Cloning into 'openfast'...
remote: Counting objects: 23801, done.
remote: Compressing objects: 100% (80/80), done.
remote: Total 23801 (delta 73), reused 102 (delta 50), pack-reused 23670
Receiving objects: 100% (23801/23801), 92.10 MiB 18.99 MiB/s, done.
Resolving deltas: 100% (13328/13328), done.
Checking connectivity... done.
```

- d) Type: `cd openfast` (change your working directory to the openfast folder)
- e) Type: `git checkout dev` (this places your local repo on the correct branch of the openfast repo)
- f) Type: `git submodule update --init --recursive` (this downloads the testing database to your computer) You should see something like this:

```
Submodule 'reg_tests/r-test' (https://github.com/openfast/r-test.git)
  registered for path 'reg_tests/r-test'
Cloning into 'reg_tests/r-test'...
remote: Counting objects: 3608, done.
remote: Compressing objects: 100% (121/121), done.
```

(continues on next page)

(continued from previous page)

```
remote: Total 3608 (delta 22), reused 161 (delta 21), pack-reused 3442
Receiving objects: 100% (3608/3608), 154.52 MiB 26.29 MiB/s, done.
Resolving deltas: 100% (2578/2578), done.
Checking connectivity... done.
Submodule path 'reg_tests/r-test': checked out
↪ 'b808f1f3c1331fe5d03c5aaa4167532c2492d378'
```

2) Build The Regression Testing DISCON DLLs

- a) Open the Visual Studio Solution (Discon.sln) located in openfast\vs-build\Discon folder
- b) Choose Release and x64 for the Solutions Configuration and Solutions Platform, respectively
- c) From the menu bar select Build->Build Solution
- d) You should now see the files Discon.dll, Discon_ITIBarge.dll, and Discon_OC3Hywind.dll in your openfast\reg_tests\r-test\glue-codes\fast\5MW_Baseline\ServoData folder.

3) Build OpenFAST using Visual Studio

- a) Open the Visual Studio Solution (FAST.sln) located in openfast\vs-build\FAST folder
- b) Choose Release_Double and x64 for the Solutions Configuration and Solutions Platform, respectively
- c) From the menu bar select Build->Build Solution
 - i) If this is the first time you have tried to build openfast, you will get build errors!!! [continue to steps (ii) and (iii), otherwise if FAST builds successfully, continue to step (3d)]
 - ii) **Cancel build using the menubar Build->Cancel**
[VS is confused about the build-order/dependency of the project files in FASTlib., but canceling and restarting VS, it somehow as enough info from the partial build to get this right, now]
 - iii) Close your Visual Studio and then Repeat Steps (a) through (c)
- d) You should now see the file openfast_x64_Double.exe in your openfast\build\bin folder.

4) Prepare regression tests

- a) Create a subdirectory called reg_tests in your openfast\build folder.
- b) Copy the contents of openfast\reg_tests\r-test to openfast\build\reg_tests.

5) Execute the OpenFAST regression Tests

- a) Open a command prompt which is configured for Python [like Anaconda3]
- b) Change your working directory to openfast\reg_tests
- c) **Type: python manualRegressionTest.py ..\build\bin\openfast_x64_Double.exe Windows Intel 1e-5**
You should see this: executing AWT_YFix_WSt
- d) The tests will continue to execute one-by-one until you finally see something like this:

```
executing AWT_YFix_WSt                PASS
executing AWT_WSt_StartUp_HighSpShutDown  PASS
executing AWT_YFree_WSt                PASS
executing AWT_YFree_WTurb               PASS
executing AWT_WSt_StartUpShutDown        PASS
executing AOC_WSt                       PASS
executing AOC_YFree_WTurb               PASS
```

(continues on next page)

(continued from previous page)

executing	AOC_YFix_WSt	PASS
executing	UAE_Dnwind_YRamp_WSt	PASS
executing	UAE_Upwind_Rigid_WRamp_PwrCurve	PASS
executing	WP_VSP_WTurb_PitchFail	PASS
executing	WP_VSP_ECD	PASS
executing	WP_VSP_WTurb	PASS
executing	WP_Stationary_Linear	PASS
executing	SWRT_YFree_VS_EDG01	PASS
executing	SWRT_YFree_VS_EDC01	PASS
executing	SWRT_YFree_VS_WTurb	PASS
executing	5MW_Land_DLL_WTurb	PASS
executing	5MW_OC3Mnpl_DLL_WTurb_WavesIrr	PASS
executing	5MW_OC3Trpd_DLL_WSt_WavesReg	PASS
executing	5MW_OC4Jckt_DLL_WTurb_WavesIrr_MGrowth	PASS
executing	5MW_ITIBarge_DLL_WTurb_WavesIrr	PASS
executing	5MW_TLP_DLL_WTurb_WavesIrr_WavesMulti	PASS
executing	5MW_OC3Spar_DLL_WTurb_WavesIrr	PASS
executing	5MW_OC4Semi_WSt_WavesWN	PASS
executing	5MW_Land_BD_DLL_WTurb	PASS

e) If an individual test succeeds you will see PASS otherwise you will see FAIL after that test's name

3.2.5 Adding test cases

In all modes of execution, the regression tests are ultimately driven by a series of Python scripts located in the `openfast/reg_tests` directory with the naming scheme `execute<Module>RegressionTest.py`. The first step to adding a new regression test case is to verify that a script exists for the target module. If it does not, an issue should be opened in [OpenFAST Issues](#) to coordinate with the NREL team on creating this script.

The next step is to add the test case in the appropriate location in the *r-test* submodule. The directory structure in *r-test* mirrors the directory structure in OpenFAST, so module-level tests should be placed in their respective module directories and glue-code tests go in `r-test/glue-codes/openfast`. Note the naming scheme of files for existing tests and adapt the new test case files accordingly. Specifically, the main input file and output file names may be expected in a particular convention by the Python scripts. Also, consider that any relative paths within the input deck for the new test case must work within the *r-test* directory structure.

Once the test directory exists, the test case must be registered with the appropriate drivers. For OpenFAST glue-code tests, this happens both in CMake and a standalone list of test cases. For CMake, edit the file `openfast/reg_tests/CTestList.cmake`. The additional test should be added in the section corresponding to the module or driver at the bottom of that file. For the Python driver, the new test case must be added to `openfast/reg_tests/r-test/glue-codes/openfast/CaseList.md`. At this point, the registration with CTest can be verified:

```
# Move into the build directory
cd openfast/build

# Run CMake to take the new changes to the test list
cmake .. -DBUILD_TESTING=ON # If the BUILD_TESTING flag was previously enabled, this
                             ↪ can be left off

# List the registered tests, but don't run them
ctest -N
```


For module regression tests, the only option for execution is with the CMake driver, so follow the instructions above to edit `CTestList.cmake`.

Finally, the new test cases in the `r-test` submodule must be added to the `r-test` repository. To do this, open a new issue in [r-test Issues](#) requesting for support from the NREL team to commit your test.

3.3 Obtaining and configuring the test suite

Portions of the test suite are linked to the OpenFAST repository through a *git submodule*. Specifically, the following two repositories are included:

- `r-test`
- `pFUnit`

Tip: Be sure to clone the repo with the `--recursive` flag or execute `git submodule update --init --recursive` after cloning.

The test suite is configured with CMake similar to the general OpenFAST build process with an additional CMake flag:

```
# BUILD_TESTING      - Build the testing tree (Default: OFF)
cmake .. -DBUILD_TESTING:BOOL=ON
```

Aside from this flag, the default CMake configuration is suitable for most systems. See the [Understanding CMake](#) section for more details on configuring the CMake targets. While the unit tests must be built with CMake due to its external dependencies, the regression test may be executed without CMake, as described in [Executing with Python driver](#).

USER DOCUMENTATION

We are in the process of transitioning legacy FAST v8 documentation, which can be found at <https://www.nrel.gov/wind/nwtc.html>.

Note: Much of the documentation here is legacy documentation from FAST v8. While most of it is still directly applicable to OpenFAST, portions may be out of date.

4.1 General

4.1.1 FAST v8 and the transition to OpenFAST

This page describes the transition from FAST v8, a computer-aided engineering tool for simulating the coupled dynamic response of wind turbines, to OpenFAST. OpenFAST was established by researchers at the National Renewable Energy Laboratory (NREL) in 2017, who were supported by the U.S. Department of Energy Wind Energy Technology Office (DOE-WETO).

FAST v8

FAST v8 is a computer-aided engineering tool for simulating the coupled dynamic response of wind turbines. FAST joins aerodynamics models, hydrodynamics models for offshore structures, control and electrical system (servo) dynamics models, and structural (elastic) dynamics models to enable coupled nonlinear aero-hydro-servo-elastic simulation in the time domain. The FAST tool enables the analysis of a range of wind turbine configurations, including two- or three-blade horizontal-axis rotor, pitch or stall regulation, rigid or teetering hub, upwind or downwind rotor, and lattice or tubular tower. The wind turbine can be modeled on land or offshore on fixed-bottom or floating substructures. FAST is based on advanced engineering models derived from fundamental laws, but with appropriate simplifications and assumptions, and supplemented where applicable with computational solutions and test data.

The aerodynamic models use wind-inflow data and solve for the rotor-wake effects and blade-element aerodynamic loads, including dynamic stall. The hydrodynamics models simulate the regular or irregular incident waves and currents and solve for the hydrostatic, radiation, diffraction, and viscous loads on the offshore substructure. The control and electrical system models simulate the controller logic, sensors, and actuators of the blade-pitch, generator-torque, nacelle-yaw, and other control devices, as well as the generator and power-converter components of the electrical drive. The structural-dynamics models apply the control and electrical system reactions, apply the aerodynamic and hydrodynamic loads, adds gravitational loads, and simulate the elasticity of the rotor, drivetrain, and support structure. Coupling between all models is achieved through a modular interface and coupler.

Transition to OpenFAST

The release of OpenFAST represents a transition to better support an open-source developer community across research laboratories, industry, and academia around FAST-based aero-hydro-servo-elastic engineering models of wind-turbines and wind-plants. OpenFAST aims to provide a solid software-engineering framework for FAST development including well documented source code, extensive automated regression and unit testing, and a robust multi-platform and compiler build system.

OpenFAST includes the following organizational changes relative to FAST v8.16:

- A new GitHub organization has been established at <https://github.com/openfast>
- The OpenFAST glue codes, modules, module drivers, and compiling tools are contained within a single repository: <https://github.com/openfast/openfast>
- The FAST program has been renamed OpenFAST (starting from OpenFAST v1.0.0)
- Version numbering has been updated for OpenFAST (starting from OpenFAST v1.0.0), e.g., OpenFAST-v1.0.0-123-gabcd1234-dirty, where:
 - v1.0.0 is the major-minor-bugfix numbering system and corresponds to a tagged commit made by NREL on GitHub
 - 123-g is the number of additional commits after the most recent tag for a build [the ‘-g’ is for ‘git’]
 - abcd1234 is the first 8 characters of the current commit hash
 - dirty denotes that local changes have been made but not committed
- Because all modules are contained in the same repository, the version numbers of each module have been eliminated and now use the OpenFAST version number (starting from OpenFAST v1.0.0) though old documentation may still refer to old version numbers
- The OpenFAST regression test baseline solutions (formerly the Certification Tests or CertTest) reside in a standalone repository: <https://github.com/openfast/r-test> (starting from OpenFAST v1.0.0)
- Unit testing has been introduced at the subroutine level (starting with BeamDyn from OpenFAST v1.0.0).
- An online documentation system has been established to replace existing documentation of FAST v8: <http://openfast.readthedocs.io/>; during the transition to OpenFAST, most user-related documentation is still provided through the NWTC Information Portal, <https://nwtc.nrel.gov>
- Cross platform compiling is accomplished with CMake on macOS, Linux, and Cygwin (Windows) systems
- Visual Studio Projects (VS-Build) are provided for compiling OpenFAST on Windows (starting from OpenFAST v1.0.0), but the development team is working to automate the generation of Visual Studio build files via CMake in a future release
- **GitHub Issues** has been made the primary platform for developers to report and track bugs, request feature enhancements, and to ask questions related to the source code, compiling, and regression/unit testing; general user-related questions on OpenFAST theory and usage should still be handled through the forum at <https://wind.nrel.gov/forum/wind>
- A new API has been added that provides a high level interface to run OpenFAST through a C++ driver code helping to interface OpenFAST with external programs like CFD solvers written in C++ (starting in OpenFAST v1.0.0)

Release Notes for OpenFAST

This section outlines significant modifications to OpenFAST made with each tagged release.

v0.1.0 (April 2017)

Algorithmically, OpenFAST v0.1.0 is the release most closely related to FAST v8.16.

- Organizational changes:
 - A new GitHub organization has been established at <https://github.com/openfast>
 - The OpenFAST glue codes, modules, module drivers, and compiling tools are contained within a single repository: <https://github.com/openfast/openfast>
 - Cross platform compiling is accomplished with CMake on macOS, Linux, and Cygwin (Windows) systems
 - An online documentation system has been established to replace existing documentation of FAST v8: <http://openfast.readthedocs.io/>
 - **GitHub Issues** has been made the primary platform for developers to report and track bugs, request feature enhancements, and to ask questions related to the source code, compiling, and regression/unit testing; general user-related questions on OpenFAST theory and usage should still be handled through the forum at <https://wind.nrel.gov/forum/wind>
- The AeroDyn v15 aerodynamics module has been significantly updated. The blade-element/momentum theory (BEMT) solution algorithm has been improved as follows:
 - BEMT now functions for the case where the undisturbed velocity along the x-direction of the local blade coordinate system (V_x) is less than zero
 - BEMT no longer aborts when a valid value of the inflow angle (ϕ) cannot be found; in this case, the inflow angle is computed geometrically (without induction)
 - The inflow angle (ϕ) is now initialized on the first call instead of defaulting to using $\phi = 0$, giving better results during simulation start up
 - When hub- and/or tip-loss are enabled (HubLoss = True and/or TipLoss = True), tangential induction (a') is set to 0 instead of -1 at the root and/or tip, respectively (axial induction (a) is still set to 1 at the root and/or tip)
 - The BEMT solution has been made more efficient
 - In addition, several bugs in AeroDyn v15 have been fixed, including:
 - Fixed a bug whereby when hub- and/or tip-loss are enabled (HubLoss = True and/or TipLoss = True) along with the Pitt/Peters skewed-wake correction (SkewMod = 2), BEMT no longer modifies the induction factors at the hub and/or tip, respectively
 - Fixed a bug whereby the time series was affected after the linearization analysis with AeroDyn coupled to OpenFAST when frozen wake is enabled (FrozenWake = True)
- The BeamDyn finite-element blade structural-dynamics model has undergone an extensive cleanup of the source code. A bug in an off-diagonal term in the structural damping-induced stiffness (i.e., representing a change in the damping force with beam displacement) has been corrected.
- A new module for user-specified platform loading (ExtPtfm) has been introduced. ExtPtfm allows the user to specify 6x6 added mass, damping, and stiffness matrices, as well as a 6x1 load vector to define loads to be applied to ElastoDyn's tower base/platform, e.g., to support the modeling of substructures or foundations through a super-element representation (with super-element derived from external software). ExtPtfm also provides the user with a module to customize with more advanced platform applied loads. Module ExtPtfm can be enabled by setting

CompSub to 2 in the FAST primary input file (a new option) and setting SubFile to the name of the file containing the platform matrices and load time history, but setting CompSub to 2 requires one to disable hydrodynamics (by setting CompHydro to 0). Please note that the introduction of option 2 for CompSub represents a minor input file change (the only input file change in OpenFAST v0.1.0), but the MATLAB conversion scripts have not yet been updated.

- In the ServoDyn control and electrical-system module, the units and sign of output parameter YawMom have been corrected
- In the InflowWind wind-inflow module, the ability to use TurbSim-generated tower wind data files in Bladed-style format was corrected
- Minor fixes were made to the error checking in ElastoDyn

v1.0.0 (September 2017)

- Organizational changes:
 - The FAST program has been renamed OpenFAST
 - Version numbering has been updated for OpenFAST (see Section 4.3.2 for details)
 - The OpenFAST regression test baseline solutions (formerly the Certification Tests or CertTest) reside in a standalone repository: <https://github.com/openfast/r-test>
 - Unit testing has been introduced at the subroutine level (starting with BeamDyn)
 - The online documentation (<http://openfast.readthedocs.io/en/latest/index.html>) has been extensively updated with additions for installation, testing, user (AeroDyn BeamDyn, transition from FAST v8, release notes), and developer guides, etc
 - The scripts for compiling OpenFAST using CMake on macOS, Linux, and Cygwin (Windows) systems have been updated, including the ability to compile in single precision and building with Spack
 - Visual Studio Projects (VS-Build) are provided for compiling OpenFAST on Windows
 - TurbSim has been included in the OpenFAST repository
- The AeroDyn aerodynamics module has been updated:
- Added a cavitation check for marine hydrokinetic (MHK) turbines. This includes the additions of new input parameters CavitCheck, Patm, Pvp, and FluidDepth in the AeroDyn primary input file, the addition of the Cpmin to the airfoil data files (required when CavitCheck = True), and new output channels for the minimum pressure coefficient, critical cavitation, and local cavitation numbers at the blade nodes. Please note that this input file changes represent the only input file change in OpenFAST v1.0.0, but the MATLAB conversion scripts have not yet been updated.
- Fixed a bug in the calculation of wind loads on the tower whereby the tower displacement was used in place of the tower velocity
- Tower strikes detected by the models to calculate the influence of the tower on the wind local to the blade are now treated as fatal errors instead of severe errors
- Fixed minor bugs in the unsteady airfoil aerodynamics model
- The BeamDyn finite-element blade structural-dynamics module has undergone additional changes:
- The source-code has further undergone clean up, including changing the internal coordinate system to match IEC (with the local z axis along the pitch axis)
- Trapezoidal points are now correctly defined by blade stations instead of key points

- The tip rotation outputs were corrected as per GitHub issue #10 (<https://github.com/OpenFAST/openfast/issues/10>)
- The BeamDyn driver has been fixed for cases involving spinning blades
- BeamDyn no longer produces numerical “spikes” in single precision, so, it is no longer necessary to compile OpenFAST in double precision when using BeamDyn
- The ElastoDyn structural-dynamics model was slightly updated:
- The precision on some module-level outputs used as input to the BeamDyn module were increased from single to double to avoid numerical “spikes” when running BeamDyn in single precision
- Minor fixes were made to the error checking
- The ServoDyn control and electrical system module was slightly updated:
- Fixed the values of the generator torque and electrical power sent from ServoDyn to Bladed-style DLL controllers as per GitHub issue # 40 (<https://github.com/OpenFAST/openfast/issues/40>)
- Minor fixes were made to the error checking
- The OpenFAST driver/glue code has been updated:
- Correction steps have been added to the OpenFAST driver during the first few time steps to address initialization problems with BeamDyn (even with NumCrctn = 0)
- Fixed a bug in the Line2-to-Point mapping of loads as per GitHub issue #8 (<https://github.com/OpenFAST/openfast/issues/8>). Previously, the augmented mesh was being formed using an incorrect projection, thus causing strange transfer of loads in certain cases. This could cause issues in the coupling between ElastoDyn and AeroDyn and/or in the coupling between HydroDyn and SubDyn
- Added an otherwise undocumented feature for writing binary output without compression to support the new regression testing. The new format is available by setting OutFileFmt to 0 in the FAST primary input file.
- A new API has been added that provides a high level interface to run OpenFAST through a C++ driver code. The primary purpose of the C++ API is to help interface OpenFAST to external programs like CFD solvers that are typically written in C++.
- The TurbSim wind-inflow turbulence preprocessor was updated:
- The API spectra was corrected
- Several minor bugs were fixed.

OpenFAST: Looking forward

Our goal is to continually improve OpenFAST documentation and to increase the coverage of automated unit and regression testing. In order to increase testing coverage and to maintain robust software, we will require that

- new modules be equipped by the module developer(s) with sufficient module-specific unit and regression testing along with appropriate OpenFAST regression tests;
- bug fixes include appropriate unit tests;
- new features/capabilities include appropriate unit and regression tests. We are in the process of better instrumenting the BeamDyn module with extensive testing as a demonstration of requirements for new modules.

For unit testing, we will employ the pFUnit framework (<https://sourceforge.net/projects/pfunit>).

For the time being OpenFAST provides project and solution files to support users developing and compiling using Visual Studio. However, the team is continually working to automate the generation of Visual Studio build files via CMake in future releases.

Please contact Michael.A.Sprague@NREL.gov with questions regarding the OpenFAST development plan.

4.1.2 API changes between versions

This page lists the main changes in the OpenFAST API (input files) between different versions.

The changes are tabulated according to the module input file, line number, and flag name. The line number corresponds to the resulting line number after all changes are implemented. Thus, be sure to implement each in order so that subsequent line numbers are correct.

OpenFAST v3.2.0 to OpenFAST *dev*

Added in OpenFAST <i>dev</i>			
Module	Line	Flag Name	Example Value
AeroDyn driver	54*	WrVTK_Type	1 WrVTK_Type - VTK visualization data type: (switch) {1=surfaces; 2=lines; 3=both}

*Exact line number depends on number of entries in various preceeding tables.

OpenFAST v3.1.0 to OpenFAST v3.2.0

Added in OpenFAST v3.2.0			
Module	Line	Flag Name	Example Value
Turb-Sim	13	WrHAWCFF	False WrHAWCFF - Output full-field time-series data in HAWC form? (Generates RootName-u.bin, RootName-v.bin, RootName-w.bin, RootName.hawc)

Removed in OpenFAST v3.2.0			
Module	Line	Flag Name	Example Value
Turb-Sim	14	Clockwise	True Clockwise - Clockwise rotation looking downwind? (used only for full-field binary files - not necessary for AeroDyn)

OpenFAST v3.0.0 to OpenFAST v3.1.0

Added in OpenFAST v3.1.0			
Module	Line	Flag Name	Example Value
Servo-Dyn	60	Aero-ControlSec	————— AERODYNAMIC FLOW CONTROL —————
Servo-Dyn	61	AfCmode	0 AfCmode - Airfoil control mode {0: none, 1: cosine wave cycle, 4: user-defined from Simulink/Labview, 5: user-defined from Bladed-style DLL} (switch)
Servo-Dyn	62	AfC_Mean	0 AfC_Mean - Mean level for cosine cycling or steady value (-) [used only with AfCmode==1]
Servo-Dyn	63	AfC_Amp	0 AfC_Amp - Amplitude for cosine cycling of flap signal (-) [used only with AfCmode==1]
Servo-Dyn	64	AfC_Phase	0 AfC_Phase - Phase relative to the blade azimuth (0 is vertical) for cosine cycling of flap signal (deg) [used only with AfCmode==1]
Servo-Dyn	74	Cables-Section	————— CABLE CONTROL —————
Servo-Dyn	75	CCmode	0 CCmode - Cable control mode {0: none, 4: user-defined from Simulink/Labview, 5: user-defined from Bladed-style DLL} (switch)
Hydro-Dyn driver	6	WtrDens	1025 WtrDens - Water density (kg/m ³)
Hydro-Dyn driver	7	WtrDpth	200 WtrDpth - Water depth (m)
Hydro-Dyn driver	8	MSL2SWL	0 MSL2SWL - Offset between still-water level and mean sea level (m) [positive upward]
Open-FAST	21	MHK	0 MHK - MHK turbine type (switch) {0=Not an MHK turbine; 1=Fixed MHK turbine; 2=Floating MHK turbine}
Open-FAST	22	N/A	————— ENVIRONMENTAL CONDITIONS —————
Open-FAST	23	Gravity	9.80665 Gravity - Gravitational acceleration (m/s ²)
Open-FAST	24	AirDens	1.225 AirDens - Air density (kg/m ³)
Open-FAST	25	WtrDens	1025 WtrDens - Water density (kg/m ³)
Open-FAST	26	KinVisc	1.464E-05 KinVisc - Kinematic viscosity of working fluid (m ² /s)
Open-FAST	27	SpdSound	345 SpdSound - Speed of sound in air (m/s)
Open-FAST	28	Patm	103500 Patm - Atmospheric pressure (Pa) [used only for an MHK turbine cavitation check]
Open-FAST	29	Pvap	1700 Pvp - Vapour pressure of working fluid (Pa) [used only for an MHK turbine cavitation check]

*non-comment line count, excluding lines contained if NumCoords is not 0, and including all OPTIONAL lines in the UA coefficients table.

Modified in OpenFAST v3.1.0			
Module	Line	Flag Name	Example Value
Aero-Dyn	16	AirDens	“default” AirDens - Air density (kg/m ³)
Aero-Dyn	17	KinVisc	“default” KinVisc - Kinematic viscosity of working fluid (m ² /s)
Aero-Dyn	18	SpdSound	“default” SpdSound - Speed of sound in air (m/s)
Aero-Dyn	19	Patm	“default” Patm - Atmospheric pressure (Pa) [used only when CavitCheck=True]
Aero-Dyn	20	Pvap	“default” Pvap - Vapour pressure of working fluid (Pa) [used only when CavitCheck=True]
Hydro-Dyn	5	WtrDens	“default” WtrDens - Water density (kg/m ³)
Hydro-Dyn	6	WtrDpth	“default” WtrDpth - Water depth (meters)
Hydro-Dyn	7	MSL2SWL	“default” MSL2SWL - Offset between still-water level and mean sea level (meters) [positive upward; unused when WaveMod = 6; must be zero if PotMod=1 or 2]

Removed in OpenFAST v3.1.0			
Module	Line	Flag Name	Example Value
Aero-Dyn	21	FluidDepth	0.5 FluidDepth - Water depth above mid-hub height (m) [used only when CavitCheck=True]
Elasto-Dyn	7	N/A	_____ ENVIRONMENTAL CONDITION
Elasto-Dyn	8	Gravity	9.80665 Gravity - Gravitational acceleration (m/s ²)

- The AeroDyn driver input file was completely rewritten. You may consult the following examples for a single rotor and multiple rotors in addition to the [AeroDyn driver documentation](#).
- SubDyn
 - SubDyn Driver, applied loads input:

Added			
Module	Line	Flag Name	Example Value
SubDyn driver	21	[separator line]	----- LOADS -----
SubDyn driver	22	nAppliedLoads	1 nAppliedLoads - Number of applied loads at given nodes false
SubDyn driver	23	ALTable-Header	ALJointID Fx Fy Fz Mx My Mz UnsteadyFile
SubDyn driver	24	ALTableUnit	(-) (N) (N) (N) (Nm) (Nm) (Nm) (-)
SubDyn driver	25	ALTable-Line1	10 0.0 0.0 0.0 0.0 0.0 0.0 ""

- SubDyn: the lines at n+1 and n+2 below were inserted after line n.

Added			
Module	Line	Flag Name	Example Value
Sub-Dyn	n	OutCOSM	Output cosine matrices with the selected output member forces (flag)
Sub-Dyn	n+1	OutCBModes	Output Guyan and Craig-Bampton modes {0: No output, 1: JSON output}, (flag)
Sub-Dyn	n+2	OutFEM-Modes	Output first 30 FEM modes {0: No output, 1: JSON output} (flag)

OpenFAST v2.6.0 to OpenFAST v3.0.0

ServoDyn Changes

- The input file parser is updated to a keyword/value pair based input. Each entry must have a corresponding keyword with the same spelling as expected.
- The TMD submodule of ServoDyn is replaced by an updated Structural Control module (StC) with updated capabilities and input file.

Removed in OpenFAST v3.0.0			
Module	Line	Flag Name	Example Value
Servo-Dyn	60	na	----- TUNED MASS DAMPER -----
Servo-Dyn	61	CompNTMD	False CompNTMD - Compute nacelle tuned mass damper {true/false} (flag)
Servo-Dyn	62	NTMDfile	“NRELOffshrBslne5MW_ServoDyn_TMD.dat” NTMDfile - Name of the file for nacelle tuned mass damper (quoted string) [unused when CompNTMD is false]
Servo-Dyn	63	CompTTMD	False CompTTMD - Compute tower tuned mass damper {true/false} (flag)
Servo-Dyn	64	TTMDfile	“NRELOffshrBslne5MW_ServoDyn_TMD.dat” TTMDfile - Name of the file for tower tuned mass damper (quoted string) [unused when CompTTMD is false]

Added in OpenFAST v3.0.0			
Module	Line	Flag Name	Example Value
Servo-Dyn	60	na	----- STRUCTURAL CONTROL -----
Servo-Dyn	61	NumB-StC	0 NumBStC - Number of blade structural controllers (integer)
Servo-Dyn	62	BStC-files	“unused” BStCfiles - Name of the files for blade structural controllers (quoted strings) [unused when NumBStC==0]
Servo-Dyn	63	NumN-StC	0 NumNStC - Number of nacelle structural controllers (integer)
Servo-Dyn	64	NStC-files	“unused” NStCfiles - Name of the files for nacelle structural controllers (quoted strings) [unused when NumNStC==0]
Servo-Dyn	65	NumT-StC	0 NumTStC - Number of tower structural controllers (integer)
Servo-Dyn	66	TStC-files	“unused” TStCfiles - Name of the files for tower structural controllers (quoted strings) [unused when NumTStC==0]
Servo-Dyn	67	NumSSStC	0 NumSSStC - Number of substructure structural controllers (integer)
Servo-Dyn	68	SSStC-files	“unused” SSStCfiles - Name of the files for substructure structural controllers (quoted strings) [unused when NumSSStC==0]

OpenFAST v2.5.0 to OpenFAST v2.6.0

Many changes were applied to SubDyn input file format. You may consult the following example: (SubDyn's `Input File`): and the online SubDyn documentation.

Added in OpenFAST v2.6.0			
Module	Line	Flag Name	Example Value
Aero-Dyn 15		TwrTi	0.0000000E+00 6.0000000E+00 1.0000000E+00 1.0000000E-01 [additional column in <i>Tower Influence and Aerodynamics</i> table]
Sub-Dyn	8	GuyanLoadCorr.	False GuyanLoadCorection - Include extra moment from lever arm at interface and rotate FEM for floating
Sub-Dyn	15	GuyanDampMod	0 GuyanDampMod - Guyan damping {0=none, 1=Rayleigh Damping, 2=user specified 6x6 matrix }
Sub-Dyn	16	RayleighDamp	0.001, 0.003 RayleighDamp - Mass and stiffness proportional damping coefficients (Rayleigh Damping) [only if GuyanDampMod=1]
Sub-Dyn	17	GuyanDampSize	6 GuyanDampSize - Guyan damping matrix size (square, 6x6) [only if GuyanDampMod=2]
Sub-Dyn	18	GuyanDampMat	0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00
Sub-Dyn	-23	GuyanDampMat	0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00
Sub-Dyn	na	Cables-Section	————— CABLE PROPERTIES —————
Sub-Dyn	na	Cables-Section	0 NCablePropSets - Number of cable cable properties
Sub-Dyn	na	Cables-Section	PropSetID EA MatDens T0
Sub-Dyn	na	Cables-Section	(-) (N) (kg/m) (N)
Sub-Dyn	na	Rigid-Section	————— RIGID LINK PROPERTIES —————
Sub-Dyn	na	Rigid-Section	0 NRigidPropSets - Number of rigid link properties
Sub-Dyn	na	Rigid-Section	PropSetID MatDens
Sub-Dyn	na	Rigid-Section	(-) (kg/m)
46 Hydro-Dyn	52	NBody	1 NBody - Number of WAMIT bodies to be used (-) [≥ 1 , only used when PotMod=1. If NBodyMod=1, the WAMIT data contains a vector of size 6*NBody x 1 and matrices of size 6*NBody x 6*NBody; if NBodyMod>1, there are NBody sets of WAMIT data each with a vector of size 6 x 1 and matrices of size 6 x 6]

*non-comment line count, excluding lines contained if NumCoords is not 0, and including all OPTIONAL lines in the UA coefficients table.

Modified in OpenFAST v2.6.0			
Module	Line	Flag Name	Example Value
AeroDyn 15	9	TwrShadow	0 TwrShadow - Calculate tower influence on wind based on downstream tower shadow (switch) {0=none, 1=Powles model, 2=Eames model}
SubDyn	26	Joints	JointID JointXss JointYss JointZss JointType JointDirX JointDirY JointDirZ JointStiff
SubDyn	27	Joints	(-) (m) (m) (m) (-) (-) (-) (-) (Nm/rad)
SubDyn	na	Members	MemberID MJointID1 MJointID2 MPropSetID1 MPropSetID2 MType COSMID
SubDyn	na	Members	(-) (-) (-) (-) (-) (-) (-)
SubDyn	na	ConcentratedM	CMJointID JMass JMXX JMYJ JMZZ JMXJ JMXZ JMYZ MCGX MCGY MCGZ
SubDyn	na	ConcentratedM	(-) (kg) (kg*m^2) (kg*m^2) (kg*m^2) (kg*m^2) (kg*m^2) (kg*m^2) (m) (m) (m)
HydroDyn	48	ExtnMod	1 ExtnMod - Wave-excitation model {0: no wave-excitation calculation, 1: DFT, 2: state-space} (switch) [only used when PotMod=1; STATE-SPACE REQUIRES *.ssexctn INPUT FILE]
HydroDyn	49	RdtnMod	2 RdtnMod - Radiation memory-effect model {0: no memory-effect calculation, 1: convolution, 2: state-space} (switch) [only used when PotMod=1; STATE-SPACE REQUIRES *.ss INPUT FILE]

continues on next page

Table 4.1 – continued from previous page

Modified in OpenFAST v2.6.0			
Module	Line	Flag Name	Example Value
HydroDyn	50	RdtnTMax	60 RdtnTMax - Analysis time for wave radiation kernel calculations (sec) [only used when PotMod=1 and RdtnMod>0; determines RdtnDOmega=Pi/RdtnTMax in the cosine transform; MAKE SURE THIS IS LONG ENOUGH FOR THE RADIATION IMPULSE RESPONSE FUNCTIONS TO DECAY TO NEAR-ZERO FOR THE GIVEN PLATFORM!]
HydroDyn	51	RdtnDT	0.0125 RdtnDT - Time step for wave radiation kernel calculations (sec) [only used when PotMod=1 and ExctnMod>0 or RdtnMod>0; DT<=RdtnDT<=0.1 recommended; determines RdtnOmegaMax=Pi/RdtnDT in the cosine transform]

continues on next page

Table 4.1 – continued from previous page

Modified in OpenFAST v2.6.0			
Module	Line	Flag Name	Example Value
HydroDyn	54	PotFile	“Barge” PotFile - Root name of potential-flow model data; WAMIT output files containing the linear, nondimensionalized, hydrostatic restoring matrix (.hst), frequency-dependent hydrodynamic added mass matrix and damping matrix (.1), and frequency- and direction-dependent wave excitation force vector per unit wave amplitude (.3) (quoted string) [1 to NBody if NBodyMod>1] [MAKE SURE THE FREQUENCIES INHERENT IN THESE WAMIT FILES SPAN THE PHYSICALLY-SIGNIFICANT RANGE OF FREQUENCIES FOR THE GIVEN PLATFORM; THEY MUST CONTAIN THE ZERO- AND INFINITE-FREQUENCY LIMITS!]
HydroDyn	55	WAMITULEN	1 WAMITULEN - Characteristic body length scale used to redimensionalize WAMIT output (meters) [1 to NBody if NBodyMod>1] [only used when PotMod=1]
HydroDyn	56	PtfmRefxt	0.0 PtfmRefxt - The xt offset of the body reference point(s) from (0,0,0) (meters) [1 to NBody] [only used when PotMod=1]
HydroDyn	57	PtfmRefyt	0.0 PtfmRefyt - The yt offset of the body reference point(s) from (0,0,0) (meters) [1 to NBody] [only used when PotMod=1]

continues on next page

Table 4.1 – continued from previous page

Modified in OpenFAST v2.6.0			
Module	Line	Flag Name	Example Value
HydroDyn	58	PtfmRefzt	0.0 PtfmRefzt - The zt offset of the body reference point(s) from (0,0,0) (meters) [1 to NBody] [only used when PotMod=1. If NBodyMod=2,PtfmRefzt=0.0]
HydroDyn	59	PtfmRefztRot	0.0 PtfmRefztRot - The rotation about zt of the body reference frame(s) from xt/yt (degrees) [1 to NBody] [only used when PotMod=1]
HydroDyn	60	PtfmVol0	6000 PtfmVol0 - Displaced volume of water when the body is in its undisplaced position (m ³) [1 to NBody] [only used when PotMod=1; USE THE SAME VALUE COMPUTED BY WAMIT AS OUTPUT IN THE .OUT FILE!]
HydroDyn	61	PtfmCOBxt	0.0 PtfmCOBxt - The xt offset of the center of buoyancy (COB) from (0,0) (meters) [1 to NBody] [only used when PotMod=1]
HydroDyn	62	PtfmCOByt	0.0 PtfmCOByt - The yt offset of the center of buoyancy (COB) from (0,0) (meters) [1 to NBody] [only used when PotMod=1]
HydroDyn	69-74	AddF0	0 AddF0 - Additional preload (N, N-m) [If NBodyMod=1, one size 6*NBody x 1 vector; if NBodyMod>1, NBody size 6 x 1 vectors]
HydroDyn	75-80	AddCLin	0 0 0 0 0 0 AddCLin - Additional linear stiffness (N/m, N/rad, N-m/m, N-m/rad) [If NBodyMod=1, one size 6*NBody x 6*NBody matrix; if NBodyMod>1, NBody size 6 x 6 matrices]

continues on next page

Table 4.1 – continued from previous page

Modified in OpenFAST v2.6.0			
Module	Line	Flag Name	Example Value
HydroDyn	81-86	AddBLin	0 0 0 0 0 0 AddBLin - Additional linear damping(N/(m/s), N/(rad/s), N-m/(m/s), N-m/(rad/s)) [If NBodyMod=1, one size 6*NBody x 6*NBody matrix; if NBodyMod>1, NBody size 6 x 6 matrices]
HydroDyn	87-92	AddBQuad	0 0 0 0 0 0 AddBQuad - Additional quadratic drag(N/(m/s)^2, N/(rad/s)^2, N-m(m/s)^2, N-m/(rad/s)^2) [If NBodyMod=1, one size 6*NBody x 6*NBody matrix; if NBodyMod>1, NBody size 6 x 6 matrices]
HydroDyn	na	Simple Coef Tab	SimplCd SimplCdMG SimplCa SimplCaMG SimplCp SimplCpMG SimplAxCa SimplAxCaMG SimplAxCaMG SimplAxCp SimplAxCpMG
HydroDyn	na		(-) (-) (-) (-) (-) (-) (-) (-) (-) (-) (-) (-)
HydroDyn	na	Depth Coef Tab	Dpth DpthCd DpthCdMG DpthCa DpthCaMG DpthCp DpthCpMG DpthAxCa DpthAxCaMG DpthAxCaMG DpthAxCp DpthAxCpMG
HydroDyn	na		(m) (-) (-) (-) (-) (-) (-) (-) (-) (-) (-) (-)

continues on next page

Table 4.1 – continued from previous page

Modified in OpenFAST v2.6.0			
Module	Line	Flag Name	Example Value
HydroDyn	na	Member Coef Tab	MemberID MemberCd1 MemberCd2 Member- CdMG1 MemberCdMG2 MemberCa1 MemberCa2 MemberCaMG1 Mem- berCaMG2 MemberCp1 MemberCp2 Member- CpMG1 MemberCpMG2 MemberAxCd1 Mem- berAxCd2 MemberAx- CdMG1 MemberAx- CdMG2 MemberAxCa1 MemberAxCa2 Mem- berAxCaMG1 Mem- berAxCaMG2 Member- AxCp1 MemberAxCp2 MemberAxCpMG1 MemberAxCpMG2
HydroDyn	na		(-) (-) (-) (-) (-) (-) (-) (-) (-)) (-) (-) (-) (-) (-) (-) (-) (-) (-) (-) (-) (-) (-) (-) (-) (-)
HydroDyn	na	OutList names	<i>see OutlistParameters.xlsx for new and revised output channel names</i>

Removed in OpenFAST v2.6.0			
Mod- ule	Line	Flag Name	Example Value
Hydro- Dyn	68	na	_____ FLOATING PLATFORM FORCE FLAGS _____ [unused with WaveMod=6]
Hydro- Dyn	69	PtfmSgF	True PtfmSgF - Platform horizontal surge translation force (flag) or DEFAULT
Hydro- Dyn	70	Ptfm- SwF	True PtfmSwF - Platform horizontal sway translation force (flag) or DEFAULT
Hydro- Dyn	71	PtfmHvF	True PtfmHvF - Platform vertical heave translation force (flag) or DEFAULT
Hydro- Dyn	72	PtfmRF	True PtfmRF - Platform roll tilt rotation force (flag) or DEFAULT
Hydro- Dyn	73	PtfmPF	True PtfmPF - Platform pitch tilt rotation force (flag) or DEFAULT
Hydro- Dyn	74	PtfmYF	True PtfmYF - Platform yaw rotation force (flag) or DEFAULT

OpenFAST v2.4.0 to OpenFAST v2.5.0

- InflowWind
 - The input file parser is updated to a keyword/value pair based input. Each entry must have a corresponding keyword with the same spelling as expected. See [Section 4.1.3](#) for an overview.
 - Driver code includes ability to convert between wind types

Added in OpenFAST v2.5.0			
Module	Line	Flag Name	Example Value
IfW driver	6	[separator line]	===== File Conversion Options =====
IfW driver	7	WrHAWC	false WrHAWC - Convert all data to HAWC2 format? (flag)
IfW driver	8	WrBladed	false WrBladed - Convert all data to Bladed format? (flag)
IfW driver	9	WrVTK	false WrVTK - Convert all data to VTK format? (flag)
In-flowWind	7	VFlowAng	0 VFlowAng - Upflow angle (degrees) (not used for native Bladed format Wind-Type=7)

Modified in OpenFAST v2.5.0			
Module	Line	Flag Name / section	Example Value
MoorDyn	na	added CtrlChan column in LINE PROPERTIES table	

Renamed in OpenFAST v2.5.0				
Module	Line	Previous Name	New Name	Example Value
In-flowWind	17	File-name	File-Name_Uni	“Shr11_30.wnd” File-Name_Uni - Filename of time series data for uniform wind field. (-)
In-flowWind	18	RefHt	RefHt_Uni	90 RefHt_Uni - Reference height for horizontal wind speed (m)
In-flowWind	21	File-name	File-Name_BTS	“unused” File-Name_BTS - Name of the Full field wind file to use (.bts) (-)
In-flowWind	23	File-name	File-Name-Root	“unused” File-Name-Root - WindType=4: Rootname of the full-field wind file to use (.wnd, .sum); WindType=7: name of the intermediate file with wind scaling values
In-flowWind	35	RefHt	RefHt_Hawc	90 RefHt_Hawc - reference height; the height (in meters) of the vertical center of the grid (m)
In-flowWind	47	PLExp	PLExp_Hawc	10.2 PLExp_Hawc - Power law exponent (-) (used for PL wind profile type only)
In-flowWind	49	Init-Position(x)	XOffset	0 XOffset - Initial offset in +x direction (shift of wind box)

OpenFAST v2.3.0 to OpenFAST v2.4.0

Additional nodal output channels added for *AeroDyn15*, *BeamDyn*, and *ElastoDyn*.

Added in OpenFAST v2.4.0			
Module	Line	Flag Name	Example Value
Hydro-Dyn	53	ExctnMod	0 ExctnMod - Wave Excitation model {0: None, 1: DFT, 2: state-space} (-)
Open-FAST	44	CalcSteady	true CalcSteady - Calculate a steady-state periodic operating point before linearization? [unused if Linearize=False] (flag)
Open-FAST	45	TrimCase	3 TrimCase - Controller parameter to be trimmed {1: yaw; 2: torque; 3: pitch} [used only if CalcSteady=True] (-)
Open-FAST	46	TrimTol	0.0001 TrimTol - Tolerance for the rotational speed convergence [used only if CalcSteady=True] (-)
Open-FAST	47	TrimGain	0.001 TrimGain - Proportional gain for the rotational speed error (>0) [used only if CalcSteady=True] (rad/(rad/s) for yaw or pitch; Nm/(rad/s) for torque)
Open-FAST	48	Twr_Kdmp	0 Twr_Kdmp - Damping factor for the tower [used only if CalcSteady=True] (N/(m/s))
Open-FAST	49	Bld_Kdmp	0 Bld_Kdmp - Damping factor for the blades [used only if CalcSteady=True] (N/(m/s))
In-flowWind	48	InitPosition(x)	0.0 InitPosition(x) - Initial offset in +x direction (shift of wind box) [Only used with WindType = 5] (m)
Aero-Dyn	13	CompAA	False CompAA - Flag to compute AeroAcoustics calculation [only used when WakeMod=1 or 2]
Aero-Dyn	14	AA_InputFile	“unused” AA_InputFile - Aeroacoustics input file
Aero-Dyn	35	[separator line]	===== OLAF cOnvecting LAgrangian Filaments (Free Vortex Wake) Theory Options ===== [used only when WakeMod=3]
Aero-Dyn	36	OLAFInputFileName	“Elliptic_OLAF.dat” OLAFInputFileName - Input file for OLAF [used only when WakeMod=3]
Air-FoilTables	4*	BL_file	“unused” BL_file - The file name including the boundary layer characteristics of the profile. Ignored if the aeroacoustic module is not called.

Modified in OpenFAST v2.4.0				
Module	Line	New Flag Name	Example Value	Previous Flag Name/Value
Air-FoilTables	40*	filtCutOff	“DEFAULT” filtCutOff - Reduced frequency cut-off for low-pass filtering the AoA input to UA, as well as the 1st and 2nd deriv (-) [default = 0.5]	[default = 20]

*non-comment line count, excluding lines contained if NumCoords is not 0.

OpenFAST v2.2.0 to OpenFAST v2.3.0

Removed in OpenFAST v2.3.0			
Module	Line	Flag Name	Example Value
AeroDyn Airfoil Input File - Airfoil Tables	2	Ctrl	0 Ctrl ! Control setting (must be 0 for current AirfoilInfo)

Added in OpenFAST v2.3.0			
Module	Line	Flag Name	Example Value
AeroDyn Airfoil Input File - Airfoil Tables	2	UserProp	0 UserProp ! User property (control) setting
AeroDyn	37	AFTabMod	1 AFTabMod - Interpolation method for multiple airfoil tables {1=1D interpolation on AoA (first table only); 2=2D interpolation on AoA and Re; 3=2D interpolation on AoA and UserProp} (-)

OpenFAST v2.1.0 to OpenFAST v2.2.0

No changes required.

OpenFAST v2.0.0 to OpenFAST v2.1.0

Added in OpenFAST v2.1.0			
Module	Line	Flag Name	Example Value
BeamDyn driver	21	GlbRotBladeT0	True GlbRotBladeT0 - Reference orientation for BeamDyn calculations is aligned with initial blade root?

OpenFAST v1.0.0 to OpenFAST v2.0.0

Removed in OpenFAST v2.0.0			
Module	Line	Flag Name	Example Value
BeamDyn	5	analysis_type	analysis_type - 1: Static analysis; 2: Dynamic analysis

Added in OpenFAST v2.0.0			
Module	Line	Flag Name	Example Value
Aero-Dyn	22	Skew-Mod-Factor	“default” SkewModFactor - Constant used in Pitt/Peters skewed wake model {or “default” is $15/32 \cdot \pi$ } (-) [used only when SkewMod=2; unused when WakeMod=0]
Aero-Dyn	30	Section header	===== Dynamic Blade-Element/Momentum Theory Options ===== [used only when WakeMod=2]
Aero-Dyn	31	DBEMT_Mod	DBEMT_Mod - Type of dynamic BEMT (DBEMT) model {1=constant tau1, 2=time-dependent tau1} (-) [used only when WakeMod=2]
Aero-Dyn	32	tau1_const	tau1_const - Time constant for DBEMT (s) [used only when WakeMod=2 and DBEMT_Mod=1]
Beam-Dyn	5	QuasiStaticInit	True QuasiStaticInit - Use quasi-static pre-conditioning with centripetal accelerations in initialization (flag) [dynamic solve only]
Beam-Dyn	11	load_retries	DEFAULT load_retries - Number of factored load retries before quitting the simulation
Beam-Dyn	14	tngt_stf_fd	DEFAULT tngt_stf_fd - Flag to use finite differenced tangent stiffness matrix (-)
Beam-Dyn	15	tngt_stf_comp	DEFAULT tngt_stf_comp - Flag to compare analytical finite differenced tangent stiffness matrix (-)
Beam-Dyn	16	tngt_stf_pert	DEFAULT tngt_stf_pert - perturbation size for finite differencing (-)
Beam-Dyn	17	tngt_stf_diff_tol	DEFAULT tngt_stf_diff_tol - Maximum allowable relative difference between analytical and fd tangent stiffness (-)
Beam-Dyn	18	Rot-States	True RotStates - Orient states in the rotating frame during linearization? (flag) [used only when linearizing]

FAST v8.16 to OpenFAST v1.0.0

The transition from FAST v8 to OpenFAST is described in detail at [FAST v8 and the transition to OpenFAST](#).

Removed in OpenFAST v1.0.0			
Module	Line	Flag Name	Example Value
Open-FAST	18	CompSub	0 CompSub - Compute sub-structural dynamics (switch) {0=None; 1=Sub-Dyn}

Added in OpenFAST v1.0.0			
Module	Line	Flag Name	Example Value
Open-FAST	18	Comp-Sub	0 CompSub - Compute sub-structural dynamics (switch) {0=None; 1=SubDyn; 2=External Platform MCKF}
Aero-Dyn	12	Cavity-Check	False CavitCheck - Perform cavitation check? (flag)
Aero-Dyn	17	Patm	9999.9 Patm - Atmospheric pressure (Pa) [used only when CavitCheck=True]
Aero-Dyn	18	Pvap	9999.9 Pvap - Vapor pressure of fluid (Pa) [used only when CavitCheck=True]
Aero-Dyn	19	Fluid-Depth	9999.9 FluidDepth - Water depth above mid-hub height (m) [used only when CavitCheck=True]

4.1.3 Input file formats

OpenFAST uses two primary input file formats: *value column* where the first value on the line is read, and *key+value* where a value and keyword pair are read. Both formats are line number based where a specific input is expected on a specific line, with some exceptions.

Value column input files

Only the first column in a *value column* based input file is read. This is the historical format used by OpenFAST and its predecessors (the keyword was often referenced in the source code and documentation, but OpenFAST did not process the keyword or description). Everything after the first value read is simply ignored by the code. This allowed the user to keep old values while modifying things. So for example, an input line like

```
2      20      TMax      - Total run time (s)
```

would be read as 2 and the 20 and everything after it ignored.

This format and associated parsing methodology is somewhat limited in informing the user of errors in parsing, and limited the ability to pass entire input files as text strings from another code (such as a Python driver code).

Key + Value input files

The first two columns are read in *key + value* input files. One of these two columns must contain the **exact** keyword, and the other must contain the value that corresponds to it. For example, an input line

```
20      TMax      - Total run time (s)
```

is equivalent to

```
TMax      20      - Total run time (s)
```

One additional feature of this input file format is the ability to add an arbitrary number of comment lines wherever the user wishes. Any line starting with **!**, **#**, or **%** will be treated as a comment line and ignored. For example,

```
! This is a comment line that will be skipped
      % and this is also a comment line that will be skipped
# as is this comment line
```

(continues on next page)

(continued from previous page)

```

20    TMax          - Total run time (s)
! the first two columns in the above line will be read as the value + key

```

The parser for this format of input file also tracks which lines were comments, and which lines contained the value and key pair. If a keyname is not found the parser will return an error with information about which line it was reading from.

Modules using Key + Value Format

The following modules use the *key + value* format input files (all other modules use the *value column* format):

Module	Input file
AeroDyn	Main AD15 input file
AeroDyn	Airfoil files
HydroDyn	Main HD input file
InflowWind	Main IfW input file
InflowWind	Uniform wind input file
InflowWind	Bladed wind summary file
ServoDyn	Main ServoDyn input file
ServoDyn	Structural control submodule input file
ServoDyn	Structural control submodule prescribed force input file
SubDyn	SubDyn SSI matrix input files

Note that key + value format and value column input files can be identical if the value is stated before the key.

Reasons for change

The main reason for the change in the input file parsing was to allow for the passing of a complete input file in memory from a wrapper code into OpenFAST or a module. For example, when including the AeroDyn module into a Python code, the input file can be passed in directly in memory without writing to disk first. This helps reduce the IO overhead in optimization loops where the module might be called many times sequentially with very small changes to the input file. *NOTE: this is still a work in progress, so not all modules can be linked this way yet.*

To accomplish this, the file parser written by Marshall Buhl for parsing airfoil tables in AeroDyn 15 in FAST8 was used. This parser included the more robust *key + value* input format.

Troubleshooting input files

When troubleshooting an input file error, try the following procedure:

1. An error message containing a line number and variable name, the file format being parsed is a *key + value* format. Check that the key is spelled exactly as the input file. See [Section 4.1.3](#) below.
2. An error message containing only the variable name but no line number is a *value column* input file format. See [Section 4.1.3](#) below.
3. Turn on *echo* option in the input file and check the resulting *.ech* for which line the file parsing stopped at. This may help isolate where the input file parsing failed when no line number is given in the error message.
4. Compare the problematic input file with an input file of the same type from the regression test suite distributed with OpenFAST. See [section 3](#) for details on the regression tests, or check the repository at [r-test](#).

Workshop material, legacy documentation, and other resources are listed below.

- [Overview of OpenFAST at NAWEA WindTech 2019](#)
- [Workshop Presentations](#)
- [Old FAST v6 User's Guide](#)
- [FAST v8 README](#)
- [Implementation of Substructure Flexibility and Member-Level Load Capabilities for Floating Offshore Wind Turbines in OpenFAST](#)
- [FAST modularization framework for wind turbine simulation: full-system linearization](#)
- [Full-System Linearization for Floating Offshore Wind Turbines in OpenFAST](#)
- [FAST with Labview](#)
- [OutListParameters.xlsx](#) - Contains the full list of outputs for each module.

4.2 Module Documentation

This section contains documentation for the OpenFAST module-coupling environment and its underlying modules. Documentation covers usage of models, underlying theory, and in some cases module verification.

4.2.1 AeroDyn Users Guide and Theory Manual

This document offers a quick reference guide for the AeroDyn software program. It is intended to be used by the general user in combination with other OpenFAST manuals. The manual will be updated as new releases are issued and as needed to provide further information on advancements or modifications to the software. For reference, additional materials such as presentation slides, development plans, and publications can be downloaded from the list below.

- [Development and Validation of a New Blade Element Momentum Skewed-Wake Model within AeroDyn](#)
- [The Unsteady Aerodynamics Module for FAST 8](#)
- [Added-Mass Effects on a Horizontal-Axis Tidal Turbine Using FAST v8](#)
- [Predicting Cavitation on Marine and Hydrokinetic Turbine Blades with AeroDyn V15.04](#)
- [Development Plan for the Aerodynamic Linearization of OpenFAST](#)
- [AeroDyn Meshes and Related Calculations](#)
- [Calculation of Buoyancy on a Marine Hydrokinetic Turbine in AeroDyn](#)

The documentation here was derived from AeroDyn Manual for AeroDyn version 15.04 by J.Jonkman et al.

Introduction

AeroDyn is a time-domain wind turbine aerodynamics module that is coupled in the OpenFAST multi-physics engineering tool to enable aero-elastic simulation of horizontal-axis turbines. AeroDyn can also be driven as a standalone code to compute wind turbine aerodynamic response uncoupled from OpenFAST. When coupled to OpenFAST, AeroDyn can also be linearized as part of the linearization of the full coupled solution (linearization is not available in standalone mode). AeroDyn was originally developed for modeling wind turbine aerodynamics. However, the module equally applies to the hydrodynamics of marine hydrokinetic (MHK) turbines (the terms “wind turbine”, “tower”, “aerodynamics” etc. in this document imply “MHK turbine”, “MHK support structure”, “hydrodynamics” etc. for MHK turbines). Additional physics important for MHK turbines, not applicable to wind turbines, computed by AeroDyn include a cavitation check. This documentation pertains version of AeroDyn in the OpenFAST github repository. The AeroDyn version released of OpenFAST 1.0.0 is most closely related to AeroDyn version 15 in the legacy version numbering. AeroDyn version 15 was a complete overhaul from earlier version of AeroDyn. AeroDyn version 15 and newer follows the requirements of the FAST modularization framework.

AeroDyn calculates aerodynamic loads on both the blades and tower. Aerodynamic calculations within AeroDyn are based on the principles of actuator lines, where the three-dimensional (3D) flow around a body is approximated by local two-dimensional (2D) flow at cross sections, and the distributed pressure and shear stresses are approximated by lift forces, drag forces, and pitching moments lumped at a node in a 2D cross section. Analysis nodes are distributed along the length of each blade and tower, the 2D forces and moment at each node are computed as distributed loads per unit length, and the total 3D aerodynamic loads are found by integrating the 2D distributed loads along the length. When AeroDyn is coupled to OpenFAST, the blade and tower analysis node discretization may be independent from the discretization of the nodes in the structural modules. The actuator line approximations restrict the validity of the model to slender structures and 3D behavior is either neglected, captured through corrections inherent in the model (e.g., tip-loss, hub-loss, or skewed-wake corrections), or captured in the input data (e.g., rotational augmentation corrections applied to airfoil data).

AeroDyn assumes the turbine geometry consists of a one-, two-, or three-bladed rotor atop a single tower. While the undeflected tower is assumed to be straight and vertical, an undeflected blade may consider out-of-plane curvature and in-plane sweep. For blades, the 2D cross sections where the aerodynamic analysis take place may follow the out-of-plane curvature, but in-plane sweep is assumed to be accomplished by shearing, rather than rotation of the 2D cross section. Aerodynamic imbalances are possible through the use of geometrical differences between each blade.

When AeroDyn is coupled to OpenFAST, AeroDyn receives the instantaneous (possibly displaced/deflected) structural position, orientation, and velocities of analysis nodes in the tower, hub, and blades. As with curvature and sweep, the 2D cross sections where the blade aerodynamic analysis takes place will follow the out-of-plane deflection, but in-plane deflection is assumed to be accomplished by shearing, rather than rotation of the 2D cross section. AeroDyn also receives the local freestream (undisturbed) fluid velocities at the tower and blade nodes. (Fluid and structural calculations take place outside of the AeroDyn module and are passed as inputs to AeroDyn by the driver code.) The fluid and structural motions are provided at each coupling time step and then AeroDyn computes the aerodynamic loads on the blade and tower nodes and returns them back to OpenFAST as part of the aero-elastic calculation. In standalone mode, the inputs to AeroDyn are prescribed by a simple driver code, without aero-elastic coupling.

AeroDyn consists of four submodels: (1) rotor wake/induction, (2) blade airfoil aerodynamics, (3) tower influence on the fluid local to the blade nodes, and (4) tower drag. Nacelle, hub, and tail-vane fluid influence and loading, aeroacoustics, and wake and array effects between multiple turbines in a wind plant, are not yet available in AeroDyn v15 and newer.

For operating wind and MHK turbine rotors, AeroDyn calculates the influence of the wake via induction factors based on the quasi-steady Blade-Element/Momentum (BEM) theory, which requires an iterative nonlinear solve (implemented via Brent’s method). By quasi-steady, it is meant that the induction reacts instantaneously to loading changes. The induction calculation, and resulting inflow velocities and angles, are based on flow local to each analysis node of each blade, based on the relative velocity between the fluid and structure (including the effects of local inflow skew, shear, turbulence, tower flow disturbances, and structural motion, depending on features enabled). The Glauert’s empirical correction (with Buhl’s modification) replaces the linear momentum balance at high axial induction factors. In the BEM solution, Prandtl tip-loss, Prandtl hub-loss, and Pitt and Peters skewed-wake are all 3D corrections that can op-

tionally be applied. When the skewed-wake correction is enabled, it is applied after the BEM iteration. Additionally, the calculation of tangential induction (from the angular momentum balance), the use of drag in the axial-induction calculation, and the use of drag in the tangential-induction calculation are all terms that can optionally be included in the BEM iteration (even when drag is not used in the BEM iteration, drag is still used to calculate the nodal loads once the induction has been found). The wake/induction calculation can be bypassed altogether for the purposes of modeling rotors that are parked or idling, in which case the inflow velocity and angle are determined purely geometrically. During linearization analyses with AeroDyn coupled to OpenFAST and BEM enabled, the wake can be assumed to be frozen (i.e., the axial and tangential induces velocities, $-V_x a$ and $V_y a'$, are fixed at their operating-point values during linearization) or the induction can be recalculated during linearization using BEM theory. Dynamic wake that accounts for induction dynamics as a result of transient conditions are not yet available in AeroDyn v15 and newer.

The blade airfoil aerodynamics can be steady or unsteady, except in the case that a cavitation check is requested for MHK, in which case only steady aerodynamics are supported. In the steady model, the supplied static airfoil data — including the lift force, drag force, and optional pitching moment and minimum pressure coefficients versus angle of attack (AoA) — are used directly to calculate nodal loads. The [AirfoilPrep](#) preprocessor can be used to generate the needed static airfoil data based on uncorrected 2D data (based, e.g., on airfoil tests in a wind tunnel or [XFOIL](#)), including features to blend data between different airfoils, apply 3D rotational augmentation, and extrapolate to high AoA. The unsteady airfoil aerodynamic (UA) models account for flow hysteresis, including unsteady attached flow, trailing-edge flow separation, dynamic stall, and flow reattachment. The UA models can be considered as 2D dynamic corrections to the static airfoil response as a result of time-varying inflow velocities and angles. Three semi-empirical UA models are available: the original theoretical developments of Beddoes-Leishman (B-L), extensions to the B-L developed by González, and extensions to the B-L model developed by Minnema/Pierce. **While all of the UA models are documented in this manual, the original B-L model is not yet functional. Testing has shown that the González and Minnema/Pierce models produce reasonable hysteresis of the normal force, tangential force, and pitching-moment coefficients if the UA model parameters are set appropriately for a given airfoil, Reynolds number, and/or Mach number. However, the results will differ a bit from earlier versions of AeroDyn, (which was based on the Minnema/Pierce extensions to B-L) even if the default UA model parameters are used, due to differences in the UA model logic between the versions. We recommend that users run test cases with uniform wind inflow and fixed yaw error (e.g., through the standalone AeroDyn driver) to examine the accuracy of the normal force, tangential force, and pitching-moment coefficient hysteresis and to adjust the UA model parameters appropriately.** The airfoil-, Reynolds-, and Mach-dependent parameters of the UA models may be derived from the static airfoil data. These UA models are valid for small to moderate AoA under normal rotor operation; the steady model is more appropriate under parked or idling conditions. The static airfoil data is always used in the BEM iteration; when UA is enabled, it is applied after the BEM iteration and after the skewed-wake correction. The UA models are not set up to support linearization, so, UA must be disabled during linearization analyses with AeroDyn coupled to OpenFAST. The interpolation of airfoil data based on Reynolds number or aerodynamic-control setting (e.g., flaps) is not yet available in AeroDyn v15 and newer.

The influence of the tower on the fluid flow local to the blade is based on a potential-flow and/or a tower-shadow model. The potential-flow model uses the analytical potential-flow solution for flow around a cylinder to model the tower dam effect on upwind rotors. In this model, the freestream (undisturbed) flow at each blade node is disturbed based on the location of the blade node relative to the tower and the tower diameter, including lower velocities upstream and downstream of the tower, higher velocities to the left and right of the tower, and cross-stream flow. The Bak correction can optionally be included in the potential-flow model, which augments the tower upstream disturbance and improves the tower wake for downwind rotors based on the tower drag coefficient. The tower shadow model can also be enabled to account for the tower wake deficit on downwind rotors. This model includes an axial flow deficit on the freestream fluid at each blade node dependent on the location of the blade node relative to the tower and the tower diameter and drag coefficient, based on the work of Powles. Both tower-influence models are quasi-steady models, in that the disturbance is applied directly to the freestream fluid at the blade nodes without dynamics, and are applied within the BEM iteration.

The aerodynamic load on the tower is based directly on the tower diameter and drag coefficient and the local relative fluid velocity between the freestream (undisturbed) flow and structure at each tower analysis node (including the effects of local shear, turbulence, and structural motion, depending on features enabled). The tower drag load calculation is quasi-steady and independent from the tower influence on flow models.

The primary AeroDyn input file defines modeling options, environmental conditions (except freestream flow), airfoils, tower nodal discretization and properties, as well as output file specifications. Airfoil data properties are read from dedicated inputs files (one for each airfoil) and include coefficients of lift force, drag force, and optional pitching moment and minimum pressure versus AoA, as well as UA model parameters. (Minimum pressure coefficients versus AoA are also included in the airfoil input files in case that a cavitation check is requested.) Blade nodal discretization, geometry, twist, chord, and airfoil identifier are likewise read from separate input files (one for each blade).

[Section 4.2.1](#) describes the AeroDyn input files. [Section 4.2.1](#) discusses the output files generated by AeroDyn; these include an echo file, summary file, and the results file. [Section 4.2.1](#) provides modeling guidance when using AeroDyn. Example input files are included in [Section 4.2.1](#). A summary of available output channels are found [Section 4.2.1](#).

Input Files

The user configures the aerodynamic model parameters via a primary AeroDyn input file, as well as separate input files for airfoil and blade data. When used in standalone mode, an additional driver input file is required. The AeroDyn driver and driver input file are detailed in [Section 4.2.1](#). The driver file specifies initialization inputs normally provided to AeroDyn by OpenFAST, as well as the per-time-step inputs to AeroDyn.

As an example, the `driver.dvr` file is the main driver, the `input.dat` is the primary input file, the `blade.dat` file contains the blade geometry data, and the `airfoil.dat` file contains the airfoil angle of attack, lift, drag, moment coefficients, and pressure coefficients. Example input files are included in [Section 4.2.1](#).

No lines should be added or removed from the input files, except in tables where the number of rows is specified and comment lines in the AeroDyn airfoil data files.

Units

AeroDyn uses the SI system (kg, m, s, N). Angles are assumed to be in radians unless otherwise specified.

AeroDyn Primary Input File

The primary AeroDyn input file defines modeling options, environmental conditions (except freestream flow), airfoils, tower nodal discretization and properties, as well as output file specifications.

The file is organized into several functional sections. Each section corresponds to an aspect of the aerodynamics model. A sample AeroDyn primary input file is given in [Section 4.2.1](#).

The input file begins with two lines of header information which is for your use, but is not used by the software.

General Options

Set the Echo flag to TRUE if you wish to have AeroDyn echo the contents of the AeroDyn primary, airfoil, and blade input files (useful for debugging errors in the input files). The echo file has the naming convention of *OutRootFile.AD.ech*. OutRootFile is either specified in the I/O SETTINGS section of the driver input file when running AeroDyn standalone, or by the OpenFAST program when running a coupled simulation.

DTAero sets the time step for the aerodynamic calculations. For accuracy and numerical stability, we recommend that DTAero be set such that there are at least 200 azimuth steps per rotor revolution. However, when AeroDyn is coupled to OpenFAST, OpenFAST may require time steps much smaller than this rule of thumb. If UA is enabled while using very small time steps, you may need to recompile AeroDyn in double precision to avoid numerical problems in the UA routines. The keyword DEFAULT for DTAero may be used to indicate that AeroDyn should employ the time step prescribed by the driver code (OpenFAST or the standalone driver program).

Set `WakeMod` to 0 if you want to disable rotor wake/induction effects or 1 to include these effects using the (quasi-steady) BEM theory model. When `WakeMod` is set to 2, a dynamic BEM theory model (DBEMT) is used (also referred to as dynamic inflow or dynamic wake model). When `WakeMod` is set to 3, the free vortex wake model is used, also referred to as OLAF (see [Section 4.2.2](#)). `WakeMod` cannot be set to 2 or 3 during linearization analyses.

Set `AFAeroMod` to 1 to include steady blade airfoil aerodynamics or 2 to enable UA; `AFAeroMod` must be 1 during linearization analyses with AeroDyn coupled to OpenFAST.

Set `TwrPotent` to 0 to disable the potential-flow influence of the tower on the fluid flow local to the blade, 1 to enable the standard potential-flow model, or 2 to include the Bak correction in the potential-flow model.

Set the `TwrShadow` to 0 to disable the tower shadow model, 1 to enable the Powles tower shadow model, or 2 to use the Eames tower shadow model. These models calculate the influence of the tower on the flow local to the blade based on the downstream tower shadow model. If the tower influence from potential flow and tower shadow are both enabled, the two influences will be superimposed.

Set the `TwrAero` flag to TRUE to calculate fluid drag loads on the tower or FALSE to disable these effects.

During linearization analyses with AeroDyn coupled OpenFAST and BEM enabled (`WakeMod` = 1), set the `FrozenWake` flag to TRUE to employ frozen-wake assumptions during linearization (i.e. to fix the axial and tangential induces velocities, and, at their operating-point values during linearization) or FALSE to recalculate the induction during linearization using BEM theory.

Set the `CavitCheck` flag to TRUE to perform a cavitation check for MHK turbines or FALSE to disable this calculation. If `CavitCheck` is TRUE, `AFAeroMod` must be set to 1 because the cavitation check does not function with unsteady airfoil aerodynamics.

Set the `CompAA` flag to TRUE to run aero-acoustic calculations. This option is only available for `WakeMod` = 1 or 2. See [Section 4.2.3](#) for information on how to use this feature.

The `AA_InputFile` is used to specify the input file for the aeroacoustics sub-module. See [Section 4.2.3](#) for information on how to use this feature.

Environmental Conditions

Environmental conditions are now specified in driver input files but are left in the AeroDyn primary input file for legacy compatibility. Use the keyword `DEFAULT` to pass in values specified by the driver input file. Otherwise, values given in the AeroDyn primary input file will overwrite those given in the driver input file. `AirDens` specifies the fluid density and must be a value greater than zero; a typical value is around 1.225 kg/m³ for air (wind turbines) and 1025 kg/m³ for seawater (MHK turbines). `KinVisc` specifies the kinematic viscosity of the fluid (used in the Reynolds number calculation); a typical value is around 1.460E-5 m²/s for air (wind turbines) and 1.004E-6 m²/s for seawater (MHK turbines). `SpdSound` is the speed of sound in the fluid (used to calculate the Mach number within the unsteady airfoil aerodynamics calculations); a typical value is around 340.3 m/s for air. The last two parameters in this section are only used when `CavitCheck` = TRUE for MHK turbines. `Patm` is the atmospheric pressure above the free surface; typically around 101,325 Pa. `Pvap` is the vapor pressure of the fluid; for seawater this is typically around 2,000 Pa.

Blade-Element/Momentum Theory Options

The input parameters in this section are not used when `WakeMod` = 0.

`SkewMod` determines the skewed-wake correction model. Set `SkewMod` to 1 to use the uncoupled BEM solution technique without an additional skewed-wake correction. Set `SkewMod` to 2 to include the Pitt/Peters correction model. **The coupled model ``SkewMod= 3`` is not available in this version of AeroDyn.**

`SkewModFactor` is used only when `SkewMod` = 2. Enter a scaling factor to use in the Pitt/Peters correction model, or enter "default" to use the default value of $\frac{15\pi}{32}$.

Set `TipLoss` to `TRUE` to include the Prandtl tip-loss model or `FALSE` to disable it. Likewise, set `HubLoss` to `TRUE` to include the Prandtl hub-loss model or `FALSE` to disable it.

Set `TanInd` to `TRUE` to include tangential induction (from the angular momentum balance) in the BEM solution or `FALSE` to neglect it. Set `AIDrag` to `TRUE` to include drag in the axial-induction calculation or `FALSE` to neglect it. If `TanInd` = `TRUE`, set `TIDrag` to `TRUE` to include drag in the tangential-induction calculation or `FALSE` to neglect it. Even when drag is not used in the BEM iteration, drag is still used to calculate the nodal loads once the induction has been found,

`IndToler` sets the convergence threshold for the iterative nonlinear solve of the BEM solution. The nonlinear solve is in terms of the inflow angle, but `IndToler` represents the tolerance of the nondimensional residual equation, with no physical association possible. When the keyword `DEFAULT` is used in place of a numerical value, `IndToler` will be set to `5E-5` when `AeroDyn` is compiled in single precision and to `5E-10` when `AeroDyn` is compiled in double precision; we recommend using these defaults. `MaxIter` determines the maximum number of iterations steps in the BEM solve. If the residual value of the BEM solve is not less than or equal to `IndToler` in `MaxIter`, `AeroDyn` will exit the BEM solver and return an error message.

Dynamic Blade-Element/Momentum Theory Options

The input parameters in this section are used only when `WakeMod` = 2.

Set `DBEMT_Mod` to 1 for the constant- τ_1 model, set `DBEMT_Mod` to 2 to use a model where τ_1 varies with time, or set `DBEMT_Mod` to 3 to use a continuous-state model with constant τ_1 .

If `DBEMT_Mod`=1 (constant- τ_1 model) or `DBEMT_Mod`=3 (continuous-state constant- τ_1 model), set `tau1_const` to the time constant to use for DBEMT.

OLAF – cOnvecting LAgrangian Filaments (Free Vortex Wake) Theory Options

The input parameters in this section are used only when `WakeMod` = 3.

The settings for the free vortex wake model are set in the OLAF input file described in [Section 4.2.2](#). `OLAFInputFileName` is the filename for this input file.

Unsteady Airfoil Aerodynamics Options

The input parameters in this section are used only when `AFAeroMod` = 2.

`UAMod` determines the UA model. It has the following options:

- 1: the original theoretical developments of B-L (**not currently functional**),
- 2: the extensions to B-L developed by González
- 3: the extensions to B-L developed by Minnema/Pierce
- 4: a continuous-state model developed by Hansen, Gaunna, and Madsen (HGM)
- 5: a model similar to HGM with an additional state for vortex generation
- 6: Oye's dynamic stall model
- 7: Boeing-Vertol model

The models are described in [Section 4.2.1](#).

While all of the UA models are documented in this manual, the original B-L model is not yet functional. Testing has shown that the González and Minnema/Pierce models produce reasonable hysteresis of the normal force,

tangential force, and pitching-moment coefficients if the UA model parameters are set appropriately for a given airfoil, Reynolds number, and/or Mach number. However, the results will differ a bit from earlier versions of AeroDyn, (which was based on the Minnema/Pierce extensions to B-L) even if the default UA model parameters are used, due to differences in the UA model logic between the versions. We recommend that users run test cases with uniform inflow and fixed yaw error (e.g., through the standalone AeroDyn driver) to examine the accuracy of the normal force, tangential force, and pitching-moment coefficient hysteresis and to adjust the UA model parameters appropriately.

FLookup determines how the nondimensional separation distance value, f^* , will be calculated. When FLookup is set to TRUE, f^* is determined via a lookup into the static lift-force coefficient and drag-force coefficient data. **Using best-fit exponential equations (`FLookup = FALSE`) is not yet available, so `FLookup` must be `TRUE` in this version of AeroDyn.** Note, FLookup is not used when UAMod=4 or UAMod=5.

UASStartRad is the starting rotor radius where dynamic stall will be turned on. Enter a number between 0 and 1, representing a fraction of rotor radius, to indicate where unsteady aerodynamics should begin turning on. If this line is omitted from the input file, UASStartRad will default to 0 (turning on at the blade root). All blade nodes that are located at a rotor radius less than UASStartRad will have unsteady aerodynamics turned off for the entire simulation.

UAEndRad is the ending rotor radius where dynamic stall will be turned on. Enter a number between 0 and 1, representing a fraction of rotor radius, to indicate the last rotor radius where unsteady aerodynamics should be turned on. If this line is omitted from the input file, UAEndRad will default to 1 (the blade tip). All blade nodes that are located at a rotor radius greater than UAEndRad will have unsteady aerodynamics turned off for the entire simulation.

Airfoil Information

This section defines the airfoil data input file information. The airfoil data input files themselves (one for each airfoil) include tables containing coefficients of lift force, drag force, and optionally pitching moment, and minimum pressure versus AoA, as well as UA model parameters, and are described in [Section 4.2.1](#).

The first 5 lines in the AIRFOIL INFORMATION section relate to the format of the tables of static airfoil coefficients within each of the airfoil input files. InCol_Alf, InCol_Cl, InCol_Cd, InCol_Cm, and InCol_Cpmin are column numbers in the tables containing the AoA, lift-force coefficient, drag-force coefficient, pitching-moment coefficient, and minimum pressure coefficient, respectively (normally these are 1, 2, 3, 4, and 5, respectively). If pitching-moment terms are neglected with UseBlCm = FALSE, InCol_Cm may be set to zero, and if the cavitation check is disabled with CavitCheck = FALSE, InCol_Cpmin may be set to zero.

Specify the number of airfoil data input files to be used using NumAFfiles, followed by NumAFfiles lines of filenames. The file names should be in quotations and can contain an absolute path or a relative path e.g., "C:\airfoils\S809_CLN_298.dat" or "airfoils\S809_CLN_298.dat". If you use relative paths, it is relative to the location of the file in which it is specified. The blade data input files will reference these airfoil data using their line identifier, where the first airfoil file is numbered 1 and the last airfoil file is numbered NumAFfiles.

Rotor/Blade Properties

Set UseBlCm to TRUE to include pitching-moment terms in the blade airfoil aerodynamics or FALSE to neglect them; if UseBlCm = TRUE, pitching-moment coefficient data must be included in the airfoil data tables with InCol_Cm not equal to zero.

The blade nodal discretization, geometry, twist, chord, and airfoil identifier are set in separate input files for each blade, described in [Section 4.2.1](#). ADB1File(1) is the filename for blade 1, ADB1File(2) is the filename for blade 2, and ADB1File(3) is the filename for blade 3, respectively; the latter is not used for two-bladed rotors and the latter two are not used for one-bladed rotors. The file names should be in quotations and can contain an absolute path or a relative path. The data in each file need not be identical, which permits modeling of aerodynamic imbalances.

Tower Influence and Aerodynamics

The input parameters in this section pertain to the tower influence and/or tower drag calculations and are only used when `TwrPotent > 0`, `TwrShadow > 0`, or `TwrAero = TRUE`.

`NumTwrNds` is the user-specified number of tower analysis nodes and determines the number of rows in the subsequent table (after two table header lines). `NumTwrNds` must be greater than or equal to two; the higher the number, the finer the resolution and longer the computational time; we recommend that `NumTwrNds` be between 10 and 20 to balance accuracy with computational expense. For each node, `TwrElev` specifies the local elevation of the tower node above ground (or above MSL for offshore wind turbines or above the seabed for MHK turbines), `TwrDiam` specifies the local tower diameter, `TwrCd` specifies the local tower drag-force coefficient, and `TwrTI` specifies the turbulence intensity used in the Eames tower shadow model (`TwrShadow = 2`) as a fraction (rather than a percentage) of the wind fluctuation. `TwrElev` must be entered in monotonically increasing order—from the lowest (tower-base) to the highest (tower-top) elevation. Values of `TwrTI` between 0.05 and 0.4 are recommended. Values larger than 0.4 up to 1 will trigger a warning that the results will need to be interpreted carefully, but the code will allow such values for scientific investigation purposes. See [Fig. 4.1](#).

Outputs

Specifying `SumPrint` to `TRUE` causes `AeroDyn` to generate a summary file with name `<OutFileRoot>.AD.sum`. `<OutFileRoot>` is either specified in the I/O SETTINGS section of the driver input file when running `AeroDyn` standalone, or by the OpenFAST program when running a coupled simulation. See [Section 4.2.1](#) for summary file details. If `AFAeroMod=2`, the unsteady aero module will also generate a file called `<OutFileRoot>.UA.sum` that will list all of the UA parameters used in the airfoil tables. This allows the user to check what values are being used in case the code has computed the parameters without user input.

`AeroDyn` can output aerodynamic and kinematic quantities at up to nine nodes specified along the tower and up to nine nodes along each blade. For outputs at every blade node, see [Section 4.2.1](#).

`NB1Outs` specifies the number of blade nodes that output is requested for (0 to 9) and `B1OutNd` on the next line is a list `NB1Outs` long of node numbers between 1 and `NumB1Nds` (corresponding to a row number in the blade analysis node table in the blade data input files), separated by any combination of commas, semicolons, spaces, and/or tabs. All blades have the same output node numbers. `NTwOuts` specifies the number of tower nodes that output is requested for (0 to 9) and `TwOutNd` on the next line is a list `NTwOuts` long of node numbers between 1 and `NumTwrNds` (corresponding to a row number in the tower analysis node table above), separated by any combination of commas, semicolons, spaces, and/or tabs. The outputs specified in the `OutList` section determine which quantities are actually output at these nodes.

The `OutList` section controls output quantities generated by `AeroDyn`. Enter one or more lines containing quoted strings that in turn contain one or more output parameter names. Separate output parameter names by any combination of commas, semicolons, spaces, and/or tabs. If you prefix a parameter name with a minus sign, “-”, underscore, “_”, or the characters “m” or “M”, `AeroDyn` will multiply the value for that channel by -1 before writing the data. The parameters are written in the order they are listed in the input file. `AeroDyn` allows you to use multiple lines so that you can break your list into meaningful groups and so the lines can be shorter. You may enter comments after the closing quote on any of the lines. Entering a line with the string “END” at the beginning of the line or at the beginning of a quoted string found at the beginning of the line will cause `AeroDyn` to quit scanning for more lines of channel names. Blade and tower node-related quantities are generated for the requested nodes identified through the `B1OutNd` and `TwOutNd` lists above. If `AeroDyn` encounters an unknown/invalid channel name, it warns the users but will remove the suspect channel from the output file. Please refer to Appendix E for a complete list of possible output parameters.

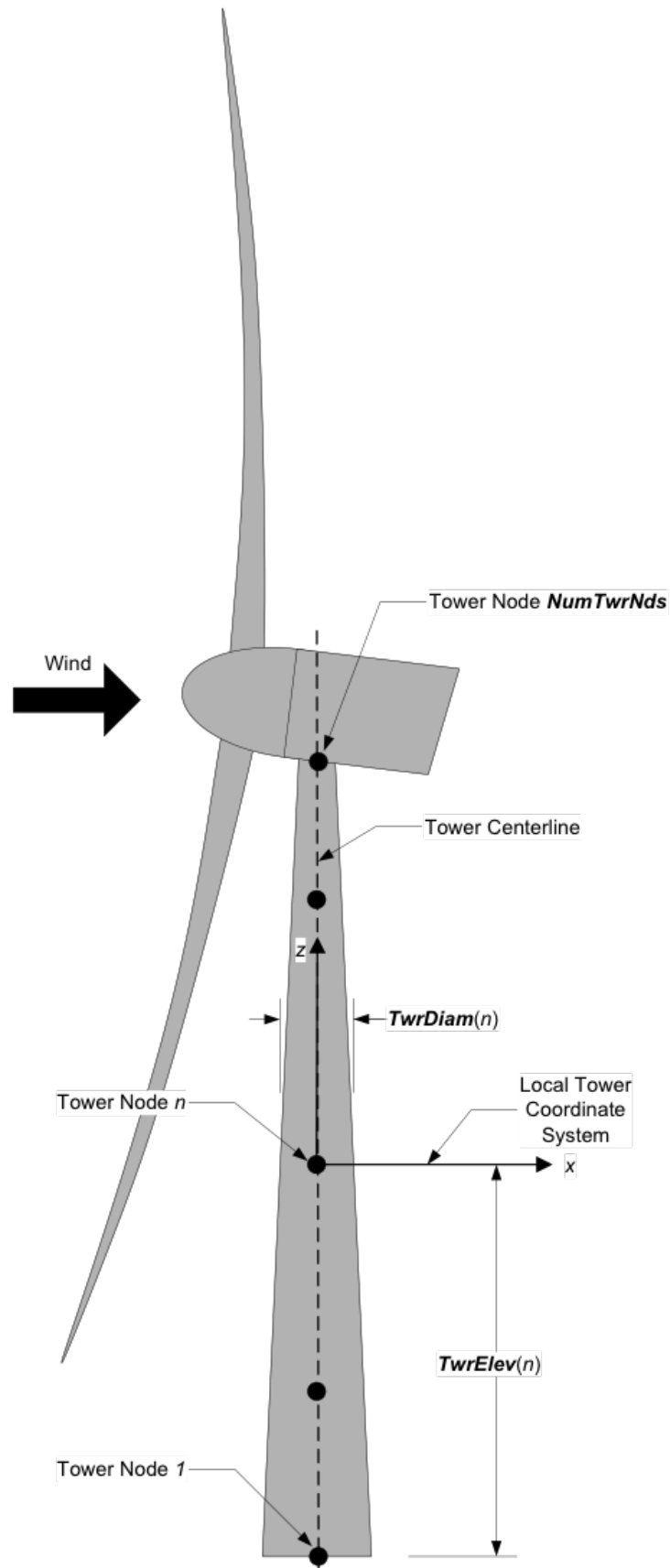


Fig. 4.1: AeroDyn Tower Geometry

Nodal Outputs

In addition to the named outputs in [Section 4.2.1](#) above, AeroDyn allows for outputting the full set blade node motions and loads (tower nodes unavailable at present). Please refer to the AeroDyn_Nodes tab in the Excel file `OutListParameters.xlsx` for a complete list of possible output parameters.

This section follows the *END* statement from normal Outputs section described above, and includes a separator description line followed by the following options.

BldNd_BladesOut specifies the number of blades to output. Possible values are 0 through the number of blades AeroDyn is modeling. If the value is set to 1, only blade 1 will be output, and if the value is 2, blades 1 and 2 will be output.

BldNd_BIOutNd specifies which nodes to output. This is currently unused.

The **OutList** section controls the nodal output quantities generated by AeroDyn. In this section, the user specifies the name of the channel family to output. The output name for each channel is then created internally by AeroDyn by combining the blade number, node number, and channel family name. For example, if the user specifies **AxInd** as the channel family name, the output channels will be named with the convention of **B β N###AxInd** where β is the blade number, and ### is the three digit node number.

Sample Nodal Outputs section

This sample includes the *END* statement from the regular outputs section.

```

1  END of input file (the word "END" must appear in the first 3 columns of this last
   ↳ OutList line)
2  ----- NODE OUTPUTS -----
3      3    BldNd_BladesOut  - Blades to output
4      99    BldNd_BIOutNd   - Blade nodes on each blade (currently unused)
5      OutList    - The next line(s) contains a list of output parameters. See
   ↳ OutListParameters.xlsx, AeroDyn_Nodes tab for a listing of available output channels,
   ↳ (-)
6  "VUndx"      - x-component of undisturbed wind velocity at each node
7  "VUndy"      - y-component of undisturbed wind velocity at each node
8  "VUndz"      - z-component of undisturbed wind velocity at each node
9  "VDisx"      - x-component of disturbed wind velocity at each node
10 "VDisy"      - y-component of disturbed wind velocity at each node
11 "VDisz"      - z-component of disturbed wind velocity at each node
12 "STVx"       - x-component of structural translational velocity at each node
13 "STVy"       - y-component of structural translational velocity at each node
14 "STVz"       - z-component of structural translational velocity at each node
15 "VRel"       - Relative wind speed at each node
16 "DynP"       - Dynamic pressure at each node
17 "Re"         - Reynolds number (in millions) at each node
18 "M"         - Mach number at each node
19 "Vindx"      - Axial induced wind velocity at each node
20 "Vindy"      - Tangential induced wind velocity at each node
21 "AxInd"      - Axial induction factor at each node
22 "TnInd"      - Tangential induction factor at each node
23 "Alpha"      - Angle of attack at each node
24 "Theta"      - Pitch+Twist angle at each node
25 "Phi"        - Inflow angle at each node
26 "Curve"      - Curvature angle at each node

```

(continues on next page)

(continued from previous page)

```

27 "Cl"      - Lift force coefficient at each node
28 "Cd"      - Drag force coefficient at each node
29 "Cm"      - Pitching moment coefficient at each node
30 "Cx"      - Normal force (to plane) coefficient at each node
31 "Cy"      - Tangential force (to plane) coefficient at each node
32 "Cn"      - Normal force (to chord) coefficient at each node
33 "Ct"      - Tangential force (to chord) coefficient at each node
34 "Fl"      - Lift force per unit length at each node
35 "Fd"      - Drag force per unit length at each node
36 "Mm"      - Pitching moment per unit length at each node
37 "Fx"      - Normal force (to plane) per unit length at each node
38 "Fy"      - Tangential force (to plane) per unit length at each node
39 "Fn"      - Normal force (to chord) per unit length at each node
40 "Ft"      - Tangential force (to chord) per unit length at each node
41 "Clrnc"   - Tower clearance at each node (based on the absolute distance to the
↳ nearest point in the tower from blade node B#N# minus the local tower radius, in the
↳ deflected configuration); please note that this clearance is only approximate because
↳ the calculation assumes that the blade is a line with no volume (however, the
↳ calculation does use the local tower radius); when blade node B#N# is above the tower
↳ top (or below the tower base), the absolute distance to the tower top (or base) minus
↳ the local tower radius, in the deflected configuration, is output
42 "Vx"      - Local axial velocity
43 "Vy"      - Local tangential velocity
44 "GeomPhi" - Geometric phi? If phi was solved using normal BEMT equations, GeomPhi = 1;
↳ otherwise, if it was solved geometrically, GeomPhi = 0.
45 "Chi"     - Skew angle (used in skewed wake correction) -- not available for OLAF
46 "UA_Flag" - Flag indicating if UA is turned on for this node. -- not available for OLAF
47 "CpMin"   - Pressure coefficient
48 "SgCav"   - Cavitation number
49 "SigCr"   - Critical cavitation number
50 "Gam"     - Gamma -- circulation on blade
51 "Cl_Static" - Static portion of lift force coefficient at each node, without unsteady
↳ effects -- not available for BEMT/DBEMT
52 "Cd_Static" - Static portion of drag force coefficient at each node, without unsteady
↳ effects -- not available for BEMT/DBEMT
53 "Cm_Static" - Static portion of pitching moment coefficient at each node, without
↳ unsteady effects -- not available for BEMT/DBEMT
54 "Uin"     - Axial induced velocity in rotating hub coordinates. Axial aligned with hub
↳ axis.      rotor plane polar hub rotating coordinates
55 "Uit"     - Tangential induced velocity in rotating hub coordinates. Tangential to the
↳ rotation plane. Perpendicular to blade azimuth.      rotor plane polar hub rotating
↳ coordinates
56 "Uir"     - Radial induced velocity in rotating hub coordinates. Radial outwards in
↳ rotation plane. Aligned with blade azimuth.      rotor plane polar hub rotating
↳ coordinates
57 END of input file (the word "END" must appear in the first 3 columns of this last
↳ OutList line)
58 -----

```

Airfoil Data Input File

The airfoil data input files themselves (one for each airfoil) include tables containing coefficients of lift force, drag force, and pitching moment versus AoA, as well as UA model parameters. In these files, any line whose first non-blank character is an exclamation point (!) is ignored (for inserting comment lines). The non-comment lines should appear within the file in order, but comment lines may be intermixed as desired for reading clarity. A sample airfoil data input file is given in [Section 4.2.1](#).

InterpOrd is the order the static airfoil data is interpolated when AeroDyn uses table look-up to find the lift-, drag-, and optional pitching-moment, and minimum pressure coefficients as a function of AoA. When **InterpOrd** is 1, linear interpolation is used; when **InterpOrd** is 3, the data will be interpolated with cubic splines; if the keyword **DEFAULT** is entered in place of a numerical value, **InterpOrd** is set to 1.

RelThickness is the non-dimensional thickness of the airfoil (thickness over chord ratio), expressed as a fraction (not a percentage), typically between 0.1 and 1. The parameter is currently used when **UAMod**=7, but might be used more in the future. The default value of 0.2 is provided for convenience.

NonDimArea is the nondimensional airfoil area (normalized by the local **B1Chord** squared), but is currently unused by AeroDyn.

NumCoords is the number of points to define the exterior shape of the airfoil, plus one point to define the aerodynamic center, and determines the number of rows in the subsequent table; **NumCoords** must be exactly zero or greater than or equal to three. For each point, the nondimensional *X* and *Y* coordinates are specified in the table, **X_Coord** and **Y_Coord** (normalized by the local **B1Chord**). The first point must always locate the aerodynamic center (reference point for the airfoil lift and drag forces, likely not on the surface of the airfoil); the remaining points should define the exterior shape of the airfoil. The airfoil shape is currently unused by AeroDyn, but when AeroDyn is coupled to OpenFAST, the airfoil shape will be used by OpenFAST for blade surface visualization when enabled.

BL_file is the name of the file containing boundary-layer characteristics of the profile. It is ignored if the aeroacoustic module is not used. This parameter may also be omitted from the file if the aeroacoustic module is not used.

Specify the number of Reynolds number- or aerodynamic-control setting-dependent tables of data for the given airfoil via the **NumTabs** setting. The remaining parameters in the airfoil data input files are entered separately for each table.

Re and **UserProp** are the Reynolds number (in millions) and aerodynamic-control (or user property) setting for the included table. These values are used only when the **AFTabMod** parameter in the primary AeroDyn input file is set to use 2D interpolation based on **Re** or **UserProp**. If 1D interpolation (based only on angle of attack) is used, only the first table in the file will be used.

Set **InclUAdata** to **TRUE** if you are including the UA model parameters. If this is set to **FALSE**, all of the UA model parameters will be determined by the code. Any lines that are missing from this section will have their values determined by the code, either using a default value or calculating it based on the polar coefficient data in the airfoil table:

- **alpha0** specifies the zero-lift AoA (in degrees);
- **alpha1** specifies the AoA (in degrees) larger than **alpha0** for which f equals 0.7; approximately the positive stall angle; This parameter is used when **flookup**=**false** and when determining a default value of **Cn1**.
- **alpha2** specifies the AoA (in degrees) less than **alpha0** for which f equals 0.7; approximately the negative stall angle; This parameter is used when **flookup**=**false** and when determining a default value of **Cn2**.
- **alphaUpper** specifies the AoA (in degrees) of the upper boundary of fully-attached region of the **cn** or **cl** curve. It is used to compute the separation function when **UAMod**=5.
- **alphaLower** specifies the AoA (in degrees) of the lower boundary of fully-attached region of the **cn** or **cl** curve. It is used to compute the separation function when **UAMod**=5. (The separation function will have a value of 1 between **alphaUpper** and **alphaLower**.)

- `eta_e` is the recovery factor and typically has a value in the range [0.85 to 0.95] for `UAMod = 1`; if the keyword `DEFAULT` is entered in place of a numerical value, `eta_e` is set to 0.9 for `UAMod = 1`, but `eta_e` is set to 1.0 for other `UAMod` values and whenever `Flookup = TRUE`;
- `C_nalpha` is the slope of the 2D normal force coefficient curve in the linear region;
- `T_f0` is the initial value of the time constant associated with Df in the expressions of Df and f' ; if the keyword `DEFAULT` is entered in place of a numerical value, `T_f0` is set to 3.0;
- `T_V0` is the initial value of the time constant associated with the vortex lift decay process, used in the expression of C_{vn} ; it depends on Reynolds number, Mach number, and airfoil; if the keyword `DEFAULT` is entered in place of a numerical value, `T_V0` is set to 6.0;
- `T_p` is the boundary-layer leading edge pressure gradient time constant in the expression for Dp and should be tuned based on airfoil experimental data; if the keyword `DEFAULT` is entered in place of a numerical value, `T_p` is set to 1.7;
- `T_VL` is the time constant associated with the vortex advection process, representing the nondimensional time in semi-chords needed for a vortex to travel from the leading to trailing edges, and used in the expression of C_{vn} ; it depends on Reynolds number, Mach number (weakly), and airfoil; valued values are in the range [6 to 13]; if the keyword `DEFAULT` is entered in place of a numerical value, `T_VL` is set to 11.0;
- `b1` is a constant in the expression of ϕ_α^c and ϕ_q^c ; this value is relatively insensitive for thin airfoils, but may be different for turbine airfoils; if the keyword `DEFAULT` is entered in place of a numerical value, `b1` is set to 0.14, based on experimental results;
- `b2` is a constant in the expression of ϕ_α^c and ϕ_q^c ; this value is relatively insensitive for thin airfoils, but may be different for turbine airfoils; if the keyword `DEFAULT` is entered in place of a numerical value, `b2` is set to 0.53, based on experimental results;
- `b5` is a constant in the expression of K_q''' , Cm_q^{nc} , and K_{mq} ; if the keyword `DEFAULT` is entered in place of a numerical value, `b5` is set to 5, based on experimental results;
- `A1` is a constant in the expression ϕ_α^c and ϕ_q^c ; this value is relatively insensitive for thin airfoils, but may be different for turbine airfoils; if the keyword `DEFAULT` is entered in place of a numerical value, `A1` is set to 0.3, based on experimental results;
- `A2` is a constant in the expression ϕ_α^c and ϕ_q^c ; this value is relatively insensitive for thin airfoils, but may be different for turbine airfoils; if the keyword `DEFAULT` is entered in place of a numerical value, `A2` is set to 0.7, based on experimental results;
- `A5` is a constant in the expression K_q''' , Cm_q^{nc} , and K_{mq} ; if the keyword `DEFAULT` is entered in place of a numerical value, `A5` is set to 1, based on experimental results;
- `S1` is the constant in the best fit curve of f for $\alpha_{00} \leq \text{AoA} \leq \alpha_{01}$ for `UAMod = 1` (and is unused otherwise); by definition, it depends on the airfoil;
- `S2` is the constant in the best fit curve of f for $\text{AoA} > \alpha_{01}$ for `UAMod = 1` (and is unused otherwise); by definition, it depends on the airfoil;
- `S3` is the constant in the best fit curve of f for $\alpha_{02} \leq \text{AoA} \leq \alpha_{00}$ for `UAMod = 1` (and is unused otherwise); by definition, it depends on the airfoil;
- `S4` is the constant in the best fit curve of f for $\text{AoA} < \alpha_{02}$ for `UAMod = 1` (and is unused otherwise); by definition, it depends on the airfoil;
- `Cn1` is the critical value of C'_n at leading-edge separation for positive AoA and should be extracted from airfoil data at a given Reynolds number and Mach number; `Cn1` can be calculated from the static value of C_n at either the break in the pitching moment or the loss of chord force at the onset of stall; `Cn1` is close to the condition of maximum lift of the airfoil at low Mach numbers;

- **Cn2** is the critical value of C'_n at leading-edge separation for negative AoA and should be extracted from airfoil data at a given Reynolds number and Mach number; Cn2 can be calculated from the static value of C_n at either the break in the pitching moment or the loss of chord force at the onset of stall; Cn2 is close to the condition of maximum lift of the airfoil at low Mach numbers;
- **St_sh** is the Strouhal's shedding frequency; if the keyword **DEFAULT** is entered in place of a numerical value, St_sh is set to 0.19;
- **Cd0** is the drag-force coefficient at zero-lift AoA;
- **Cm0** is the pitching-moment coefficient about the quarter-chord location at zero-lift AoA, positive for nose up;
- **k0** is a constant in the best fit curve of \hat{x}_{cp} and equals for $\hat{x}_{AC} - 0.25 \text{ UAMod} = 1$ (and is unused otherwise);
- **k1** is a constant in the best fit curve of \hat{x}_{cp} for $\text{UAMod} = 1$ (and is unused otherwise);
- **k2** is a constant in the best fit curve of \hat{x}_{cp} for $\text{UAMod} = 1$ (and is unused otherwise);
- **k3** is a constant in the best fit curve of \hat{x}_{cp} for $\text{UAMod} = 1$ (and is unused otherwise);
- **k1_hat** is a constant in the expression of C_c due to leading-edge vortex effects for $\text{UAMod} = 1$ (and is unused otherwise);
- **x_cp_bar** is a constant in the expression of \hat{x}_{cp}^ν for $\text{UAMod} = 1$ (and is unused otherwise); if the keyword **DEFAULT** is entered in place of a numerical value, x_cp_bar is set to 0.2; and
- **UACutOut** is the AoA (in degrees) in absolute value above which UA are disabled; if the keyword **DEFAULT** is entered in place of a numerical value, UACutOut is set to 45.
- **UACutOut_delta** is the AoA difference (in degrees) which, together with UACutOut determines when unsteady aero begins to turn off; if the keyword **DEFAULT** is entered in place of a numerical value, UACutOut_delta is set to 5 degrees. The unsteady solution is used at angles of attack less than $\text{UACutOut} - \text{UACutout_delta}$ degrees. Above UACutout, the steady solution is used (i.e., UA is disabled). The steady and unsteady solutions are blended between those two angles.
- **filtCutOff** is the cut-off reduced frequency of the low-pass filter applied to the AoA input to UA, as well as to the pitch rate and pitch acceleration derived from AoA within UA; if the keyword **DEFAULT** is entered in place of a numerical value, filtCutOff is set to 0.5. This non-dimensional value corresponds to a frequency of $\frac{U \times \text{filtCutOff}}{\pi \times \text{chord}}$ Hz.

NumAlf is the number of distinct AoA entries and determines the number of rows in the subsequent table of static airfoil coefficients; **NumAlf** must be greater than or equal to one (**NumAlf** = 1 implies constant coefficients, regardless of the AoA).

AeroDyn will interpolate on AoA using the data provided via linear interpolation or via cubic splines, depending on the setting of input **InterpOrd** above. If **AFTabMod** is set to 1, only the first airfoil table in each file will be used. If **AFTabMod** is set to 2, AeroDyn will find the airfoil tables that bound the computed Reynolds number, and linearly interpolate between the tables, using the logarithm of the Reynolds numbers. If **AFTabMod** is set to 3, it will find the bounding airfoil tables based on the **UserProp** field and linearly interpolate the tables based on it. Note that OpenFAST currently sets the **UserProp** input value to 0 unless the DLL controller is used and sets the value, so using this feature may require a code change.

For each AoA, you must set the AoA (in degrees), **alpha**, the lift-force coefficient, **Coefs(:,1)**, the drag-force coefficient, **Coefs(:,2)**, and optionally the pitching-moment coefficient, **Coefs(:,3)**, and minimum pressure coefficient, **Coefs(:,4)**, but the column order depends on the settings of **InCol_Alfa**, **InCol_Cl**, **InCol_Cd**, **InCol_Cm**, and **InCol_Cpmin** in the **AIRFOIL INFORMATION** section of the AeroDyn primary input file. AoA must be entered in monotonically increasing order—from lowest to highest AoA; the first row should be for AoA = -180 degrees and the last should be for AoA = +180 (unless **NumAlf** = 1, in which case AoA is unused). If pitching-moment terms are neglected with **UseBlCm** = **FALSE** in the **ROTOR/BLADE PROPERTIES** section of the AeroDyn primary input file, the column containing pitching-moment coefficients may be absent from the file. Likewise, if the cavitation check is

neglected with `CavitCheck = FALSE` in the GENERAL OPTIONS section of the AeroDyn primary input file, the column containing the minimum pressure coefficients may be absent from the file.

Blade Data Input File

The blade data input file contains the nodal discretization, geometry, twist, chord, and airfoil identifier for a blade. Separate files are used for each blade, which permits modeling of aerodynamic imbalances. A sample blade data input file is given in [Section 4.2.1](#).

The input file begins with two lines of header information which is for your use, but is not used by the software.

`NumBlNds` is the user-specified number of blade analysis nodes and determines the number of rows in the subsequent table (after two table header lines). `NumBlNds` must be greater than or equal to two; the higher the number, the finer the resolution and longer the computational time; we recommend that `NumBlNds` be between 10 and 20 to balance accuracy with computational expense. Even though `NumBlNds` is defined in each blade file, all blades must have the same number of nodes. For each node:

- `BlSpn` specifies the local span of the blade node along the (possibly preconed) blade-pitch axis from the root; `BlSpn` must be entered in monotonically increasing order—from the most inboard to the most outboard—and the first node must be zero, and when AeroDyn is coupled to OpenFAST, the last node should be located at the blade tip;
- `BlCrvAC` specifies the local out-of-plane offset (when the blade-pitch angle is zero) of the aerodynamic center (reference point for the airfoil lift and drag forces), normal to the blade-pitch axis, as a result of blade curvature; `BlCrvAC` is positive downwind; upwind turbines have negative `BlCrvAC` for improved tower clearance;
- `BlSwpAC` specifies the local in-plane offset (when the blade-pitch angle is zero) of the aerodynamic center (reference point for the airfoil lift and drag forces), normal to the blade-pitch axis, as a result of blade sweep; positive `BlSwpAC` is opposite the direction of rotation;
- `BlCrvAng` specifies the local angle (in degrees) from the blade-pitch axis of a vector normal to the plane of the airfoil, as a result of blade out-of-plane curvature (when the blade-pitch angle is zero); `BlCrvAng` is positive downwind; upwind turbines have negative `BlCrvAng` for improved tower clearance;
- `BlTwist` specifies the local aerodynamic twist angle (in degrees) of the airfoil; it is the orientation of the local chord about the vector normal to the plane of the airfoil, positive to feather, leading edge upwind; the blade-pitch angle will be added to the local twist;
- `BlChord` specifies the local chord length; and
- `BlAFID` specifies which airfoil data the local blade node is associated with; valid values are numbers between 1 and `NumAFfiles` (corresponding to a row number in the airfoil file table in the AeroDyn primary input file); multiple blade nodes can use the same airfoil data.

See [Fig. 4.2](#). Twist is shown in [Fig. 4.6](#) of [Section 4.2.1](#).

Output Files

AeroDyn produces three types of output files: an echo file, a summary file, and a time-series results file. The following sections detail the purpose and contents of these files.

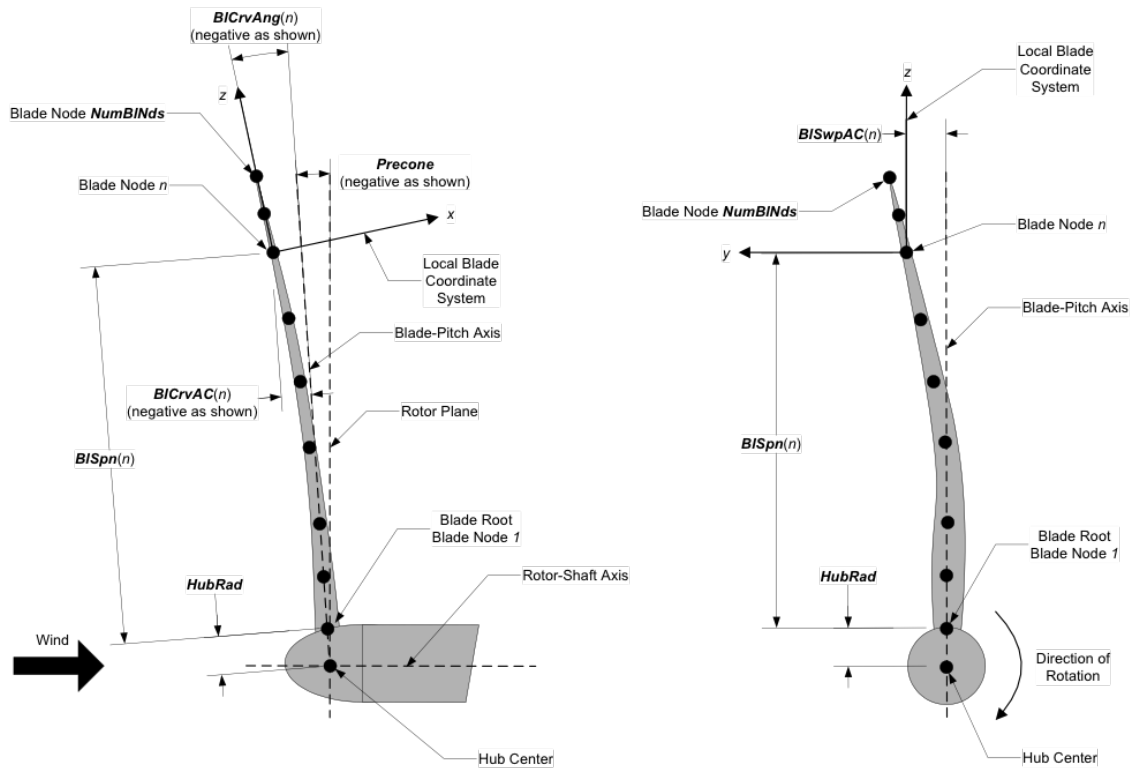


Fig. 4.2: AeroDyn Blade Geometry – Left: Side View; Right: Front View (Looking Downwind)

Echo Files

If you set the Echo flag to TRUE in the AeroDyn driver file or the AeroDyn primary input file, the contents of those files will be echoed to a file with the naming conventions, *OutFileRoot.ech* for the driver input file and *OutFileRoot.AD.ech* for the AeroDyn primary input file. *OutFileRoot* is either specified in the I/O SETTINGS section of the driver input file when running AeroDyn standalone, or by the FAST program when running a coupled simulation. The echo files are helpful for debugging your input files. The contents of an echo file will be truncated if AeroDyn encounters an error while parsing an input file. The error usually corresponds to the line after the last successfully echoed line.

Summary File

AeroDyn generates a summary file with the naming convention, *OutFileRoot.AD.sum* if the SumPrint parameter is set to TRUE. *OutFileRoot* is either specified in the I/O SETTINGS section of the driver input file when running AeroDyn standalone, or by the FAST program when running a coupled simulation. This file summarizes key information about your aerodynamics model, including which features have been enabled and what outputs have been selected.

Results Files

In standalone mode, the AeroDyn time-series results (a separate file for each case) are written to text-based files with the naming convention *OutFileRoot.#.out*, where *OutFileRoot* is specified in the I/O SETTINGS section of the driver input file and the '#' character is an integer number corresponding to a test case line found in the COMBINED-CASE ANALYSIS section. If AeroDyn is coupled to FAST, then FAST will generate a master results file that includes the AeroDyn results and AeroDyn will not write out its own results. The results are in table format, where each column is a data channel (the first column always being the simulation time), and each row corresponds to a simulation output time step. The data channels are specified in the OUTPUTS section of the AeroDyn primary input file. The column format of the AeroDyn-generated files is specified using the *OutFmt* parameter of the driver input file.

Modeling Considerations

AeroDyn was designed as an extremely flexible tool for modeling a wide-range of aerodynamic conditions and turbine configurations. This section provides some general guidance to help you construct models that are compatible with AeroDyn.

Please refer to the theory of Section 7 for detailed information about the implementation approach we have followed in AeroDyn.

Environmental Conditions

For air, typical values for *AirDens*, *KinVisc*, *SpdSound*, and *Patm* are around 1.225 kg/m³, 1.460E-5 m²/s, 340.3 m/s, and 101,325 Pa, respectively. For seawater, typical values for *AirDens*, *KinVisc*, and *Pvap* are around 1025 kg/m³, 1.004E-6 m²/s, and 2000 Pa, respectively.

Temporal and Spatial Discretization

For accuracy and numerical stability, we recommend that *DTAero* be set such that there are at least 200 azimuth steps per rotor revolution. However, when AeroDyn is coupled to FAST, FAST may require time steps much smaller than this rule of thumb. If UA is enabled while using very small time steps, you may need to recompile AeroDyn in double precision to avoid numerical problems in the UA routines.

For the blade and tower spatial discretization, using higher number of analysis nodes will result in a more accurate solution at the expense of longer computational time. When AeroDyn is coupled to FAST, the blade and tower analysis node discretization may be independent from the discretization of the nodes in the structural modules.

We recommend that *NumBlNds* be between 10 and 20 to balance accuracy with computational expense for the rotor aerodynamic load calculation. It may be beneficial to use a finer resolution of nodes where large gradients are expected in the aerodynamic loads e.g. near the blade tip. Aerodynamic imbalances are possible through the use of geometrical differences between each blade.

When the tower potential-flow (*TwrPotent* > 0), tower shadow (*TwrShadow* > 0), and/or the tower aerodynamic load (*TwrAero* = TRUE) models are enabled, we also recommend that *NumTwrNds* be between 10 and 20 to balance accuracy with computational expense. Normally the local elevation of the tower node above ground (or above MSL for offshore wind turbines or above the seabed for MHK turbines) (*TwrElev*), must be entered in monotonically increasing order from the lowest (tower-base) to the highest (tower-top) elevation. However, when AeroDyn is coupled to FAST, the tower-base node in AeroDyn cannot be set lower than the lowest point where wind is specified in the *InflowWind* module. To avoid truncating the lower section of the tower in AeroDyn, we recommend that the wind be specified in *InflowWind* as low to the ground (or MSL for offshore wind turbines or above the seabed for MHK turbines) as possible (this is a particular issue for full-field wind file formats).

Model Options Under Operational and Parked/Idling Conditions

To model an operational rotor, we recommend to include the dynamic BEM model (`WakeMod` = 2) and UA (`AFAeroMod` = 2). Normally, the Pitt and Peters skewed-wake (`SkewMod` = 2), Prandtl tip-loss (`TipLoss` = `TRUE`), Prandtl hub-loss (`HubLoss` = `TRUE`), and tangential induction (`TanInd` = `TRUE`) models should all be enabled, but `SkewMod` = 2 is invalid for very large yaw errors (much greater than 45 degrees). The nonlinear solve in the BEM solution is in terms of the inflow angle, but `IndTolEr` represents the tolerance of the nondimensional residual equation, with no physical association possible; we recommend setting `IndTolEr` to `DEFAULT`.

While all of the UA models are documented in this manual, the original B-L model is not yet functional. Testing has shown that the González and Minnema/Pierce models produce reasonable hysteresis of the normal force, tangential force, and pitching-moment coefficients if the UA model parameters are set appropriately for a given airfoil, Reynolds number, and/or Mach number. However, the results will differ a bit from earlier versions of AeroDyn, (which was based on the Minnema/Pierce extensions to B-L) even if the default UA model parameters are used, due to differences in the UA model logic between the versions. We recommend that users run test cases with uniform inflow and fixed yaw error (e.g., through the standalone AeroDyn driver) to examine the accuracy of the normal force, tangential force, and pitching-moment coefficient hysteresis and to adjust the UA model parameters appropriately.

To model a parked or idling rotor, we recommend to disable induction (`WakeMod` = 0) and UA (`AFAeroMod` = 1), in which case the inflow velocity and angle are determined purely geometrically and the airfoil data is determined statically.

The direct aerodynamic load on the tower often dominates the aerodynamic load on the rotor for parked or idling conditions above the cut-out wind speed, in which case we recommend that `TwrAero` = `TRUE`. Otherwise, `TwrAero` = `FALSE` may be satisfactory.

We recommend to include the influence of the tower on the fluid local to the blade for both operational and parked/idling rotors. We recommend that `TwrPotent` > 0 for upwind rotors and that `TwrPotent` = 2 or `TwrShadow` > 0 for downwind rotors.

Linearization

When coupled to FAST, AeroDyn can be linearized as part of the linearization of the full coupled solution. When induction is enabled (`WakeMod` = 1), we recommend to base the linearized solution on the frozen-wake assumption, by setting `FrozenWake` = `TRUE`. The UA models are not set up to support linearization, so, UA must be disabled during linearization by setting `AFAeroMod` = 1.

AeroDyn Driver

A standalone AeroDyn driver is provided to perform aerodynamic simulations of rigid turbines undergoing rigid body motion (fixed, sinusoidal, or arbitrary). The standalone AeroDyn driver code improves on the functionality previously available in the separate wind turbine rotor-performance tool `WT_Perf`. The driver also supports turbine configurations that are not currently supported by OpenFAST.

Examples of applications are:

- Simulation of horizontal/vertical axis wind turbine, kites, quad-rotors, multiple rotors and wings.
- Simulation with prescribed time series of wind speed, pitch, yaw, rotor speed.
- Combined case analyses to evaluate the turbine response at different operating conditions, for instance to compute the surfaces of power coefficient (C_P), thrust coefficient (C_T), and/or torque coefficient (C_Q) as a function of tip-speed ratio (TSR) and blade-pitch angle.
- Simulations with oscillatory motion of the tower base at different amplitudes and frequencies.

More details are provided below.

Compiling the driver

The compilation steps for the AeroDyn driver are similar to the ones of OpenFAST (see [Section 2](#)). When using the CMake workflow, the driver is compiled automatically when running `make`. To compile only the driver, use `make aerodyn_driver`. The driver will be located in the folder `/build/modules/aerodyn/`. A Visual Studio solution is available for Windows, in the folder `/vs-build/AeroDyn`.

Driver inputs and options

Main concepts

The driver supports:

- two kinds of turbine definitions: basic (HAWT), and advanced.
- two kinds of inflow definition: basic (uniform power law), and advanced (InflowWind)
- three types of analyses (*AnalysisType*): 1) one simulation of one or multiple turbines, 2) one simulation of one wind turbine under time-dependent inputs, 3) combined cases of one wind turbine

The current limitations are:

- The number of nodes per blades have to be the same for all blades and all rotors
- A turbine has only one (optional) tower
- Analysis Types 2 and 3 are limited to 1 turbine

More details are provided below, where the different sections of the input file are described.

Header and input configuration

The input file starts with a header, the user can place a relevant description of the model on the second line. The input configuration section follows. The user can toggle the flag *Echo* to write back the input file, as parsed by the driver, to disk. The *MHK* switch allows the user to specify whether or not the turbine is an MHK turbine. *MHK=0* denotes not an MHK turbine, *MHK=1* denotes a fixed MHK turbine, and *MHK=2* denotes a floating MHK turbine. Currently, only *MHK=0* can be specified, although users can still select a cavitation check. The driver supports three kinds of analyses, but not all turbine formats and inflow types are available for each analysis type:

- *AnalysisType=1*: Simulation of one or multiple rotors with basic (HAWT) or arbitrary geometries (HAWT/VAWT, quad-rotor, etc.), with basic or advanced wind inputs, and optional time-varying motion of the tower base, nacelle and individual pitch angles. Arbitrary motion or sinusoidal motion of the tower base are possible.
- *AnalysisType=2*: Simulation of one turbine with basic time-varying wind, nacelle motion, pitch. See “Time-dependent analysis” below.
- *AnalysisType=3*: Combined cases analyses of one turbine with basic steady wind. A table of cases is run sequentially by the driver. See “Combined-Case analysis” below.

The section follows with the definition of the total simulation time (*TMax*) and the time step used for the simulation (*DT*). These inputs are used for *AnalysisType=1* and *AnalysisType=2*. The user specifies the location of the AeroDyn primary file via the variable *AeroFile*. The path can be absolute or relative to the AeroDyn driver file.

An example of header and input configuration is given below:

```
----- AeroDyn Driver Input File -----
Three bladed wind turbine, using basic geometry input
----- Input Configuration -----
False          Echo          - Echo input parameters to "<rootname>.ech"?
```

(continues on next page)

(continued from previous page)

0	MHK	- MHK turbine type (switch) {0: not an MHK turbine, 1: ↪fixed MHK turbine, 2: floating MHK turbine}
3	AnalysisType	- {1: multiple turbines, one simulation, 2: one turbine, ↪one time-dependent simulation, 3: one turbine, combined-cases}
11.0	TMax	- Total run time [used only when AnalysisType/=3] (s)
0.5	DT	- Simulation time step [used only when AnalysisType/=3] (s)
"AD.dat"	AeroFile	- Name of the primary AeroDyn input file

Environmental conditions

Environmental conditions are specified here and passed to AeroDyn. *FldDens* (equivalent to *AirDens* in the AeroDyn primary input file) specifies the fluid density and must be a value greater than zero; a typical value is around 1.225 kg/m³ for air (wind turbines) and 1025 kg/m³ for seawater (MHK turbines). *KinVisc* specifies the kinematic viscosity of the fluid (used in the Reynolds number calculation); a typical value is around 1.460E-5 m²/s for air (wind turbines) and 1.004E-6 m²/s for seawater (MHK turbines). *SpdSound* is the speed of sound in the fluid (used to calculate the Mach number within the unsteady airfoil aerodynamics calculations); a typical value is around 340.3 m/s for air. The next two parameters in this section are only used when *CavitCheck* = *TRUE* for MHK turbines. *Patm* is the atmospheric pressure above the free surface; typically around 101,325 Pa. *Pvap* is the vapor pressure of the fluid; for seawater this is typically around 2,000 Pa. *WtrDpth* is the water depth from the seabed to the mean sea level (MSL).

Inflow data

The inflow can be provided in two ways:

- basic (*CompInflow*=0): uniform wind with a power law shear. The wind is defined using a reference height (*RefHt*), a power law exponent (*PLExp*), and the wind speed at reference height (*HWindSpeed*). With *AnalysisType*=2, the reference wind speed and power law are defined separately as time series (see “time-dependent analysis”). With *AnalysisType*=3, these parameters are provided in a separate table (see “Combined-Case analyses”). The reference height is used for all analyses types, since this height may be different than the hub height. The wind at a given node is determined using the following equation, where *Z* is the instantaneous elevation of the node above the ground for land-based wind turbines, above the mean sea level (MSL) for offshore wind turbines, or above the seabed for MHK turbines.

$$U(Z) = \text{HWindSpeed} \left(\frac{Z}{\text{RefHt}} \right)^{\text{PLExp}}$$

- advanced (*CompInflow*=1): the InflowWind module is used to compute the inflow, and all available options of InflowWind are then available. The user needs to provide the (relative or absolute) path of the InflowWind input file (*InflowFile*). This feature is limited to *AnalysisType*=1.

An example of inputs is given below:

```
----- Inflow Data -----
      0  CompInflow - Compute inflow wind velocities (switch) {0=Steady Wind;   
↪1=InflowWind}
"unused" InflowFile - Name of the InflowWind input file [used only when   
↪CompInflow=1]
      9.0  HWindSpeed - Horizontal wind speed [used only when CompInflow=0 and   
↪AnalysisType=1] (m/s)
      140  RefHt - Reference height for horizontal wind speed [used only when   
↪CompInflow=0] (m)
      0.10  PLExp - Power law exponent [used only when CompInflow=0 and   
↪AnalysisType=1] (-)
```

Turbine data

The user specifies the number of turbines as follows:

```

----- Turbine Data -----
1  NumTurbines - Number of turbines (should be 1 for AnalysisType=2 or AnalysisType=3)

```

As noted in the comment, the number of turbines should be 1 for *AnalysisType=2* and *AnalysisType=3*. After this section, the geometry and motion is provided for each turbine. Inputs for each turbine must have the suffix (*i*) where *i* is the turbine number (even with *NumTurbines=1*, then *i=1*). Outputs for each turbine will be written to distinct files, with the suffix *.Ti* where *i* is the turbine number (the suffix is not added when only 1 turbine is used).

An example of configuration with two wind turbines is shown in Fig. 4.3. The figure defines the different frames and origin associated with each turbine: the turbine base frame (t), nacelle frame (n), hub frame (h), and blade frames (b). The notations and conventions follow the OpenFAST frames, except that the turbine frame does not have its origin at the tower base. Prescribed motions of the turbine occur at the turbine origin. Yawing occurs around the z_n axis, the rotor rotates about the x_h axis, and blade pitching occurs around the individual z_b axes. The definitions of the different frames are standardized when using a basic (HAWT) input format definition, and are arbitrarily defined when using the advanced input format. More details are given in the next paragraph.

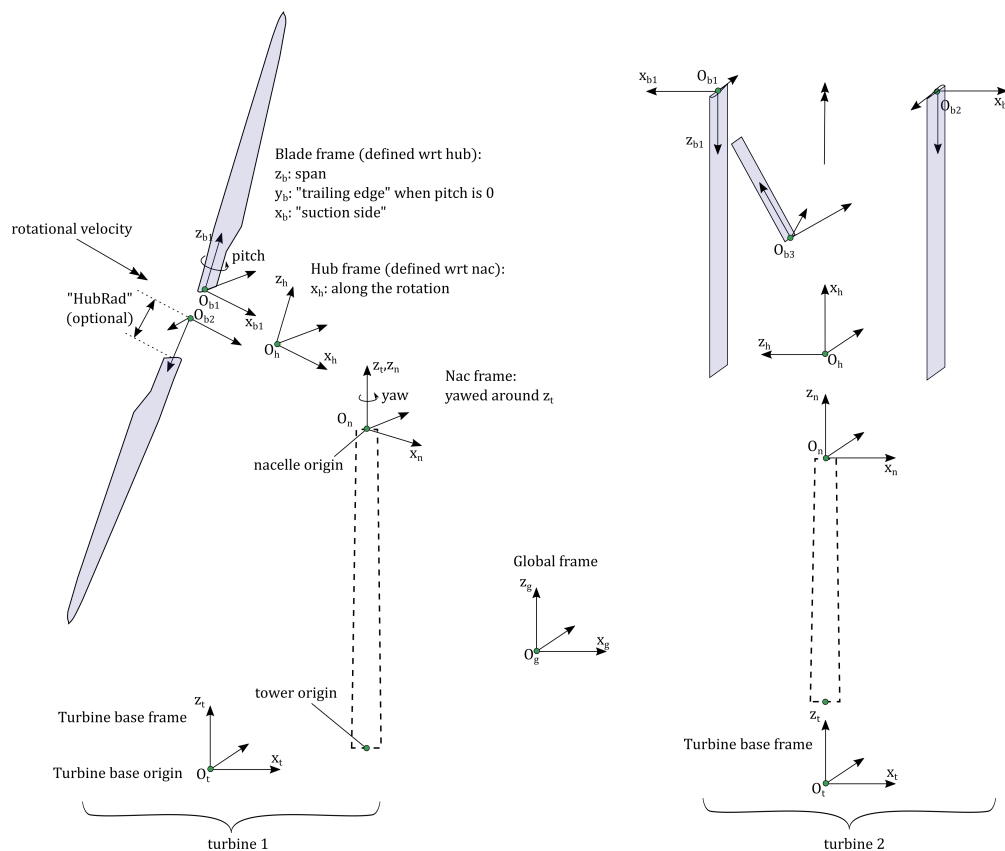


Fig. 4.3: Definition of multiple rotors.

Turbine geometry definition

Two turbine input formats are supported:

- basic (*BasicHAWTFormat=True*): Basic horizontal axis wind turbine (HAWT) format. In this format, the turbine geometry is entirely determined by the number of blades (*NumBlades*), the hub radius (*HubRad*), the hub height (*HubHt*), the overhang (*Overhang*), the shaft tilt (*ShftTilt*), the precone (*Precone*), and the vertical distance from the tower-top to the rotor shaft (*Twr2Shft*), as shown in Fig. 4.4. The definition of each parameter follows the ElastoDyn convention. For example, *HubRad* specifies the radius from the center-of-rotation to the blade root along the (possibly precone) blade-pitch axis and must be greater than zero. *HubHt* specifies the elevation of

the hub center above the ground for land-based wind turbines, above the mean sea level (MSL) for offshore wind turbines, or above the seabed for MHK turbines. *Overhang* specifies the distance along the (possibly tilted) rotor shaft between the tower centerline and hub center and is positive downwind (use a negative number for upwind rotors). *ShftTilt* is the angle (in degrees) between the rotor shaft and the horizontal plane, and positive *ShftTilt* means that the downwind end of the shaft is the highest (upwind turbines have negative *ShftTilt* for improved tower clearance). *Precone* is the angle (in degrees) between a flat rotor disk and the cone swept by the blades, positive downwind (upwind turbines have negative *Precone* for improved tower clearance).

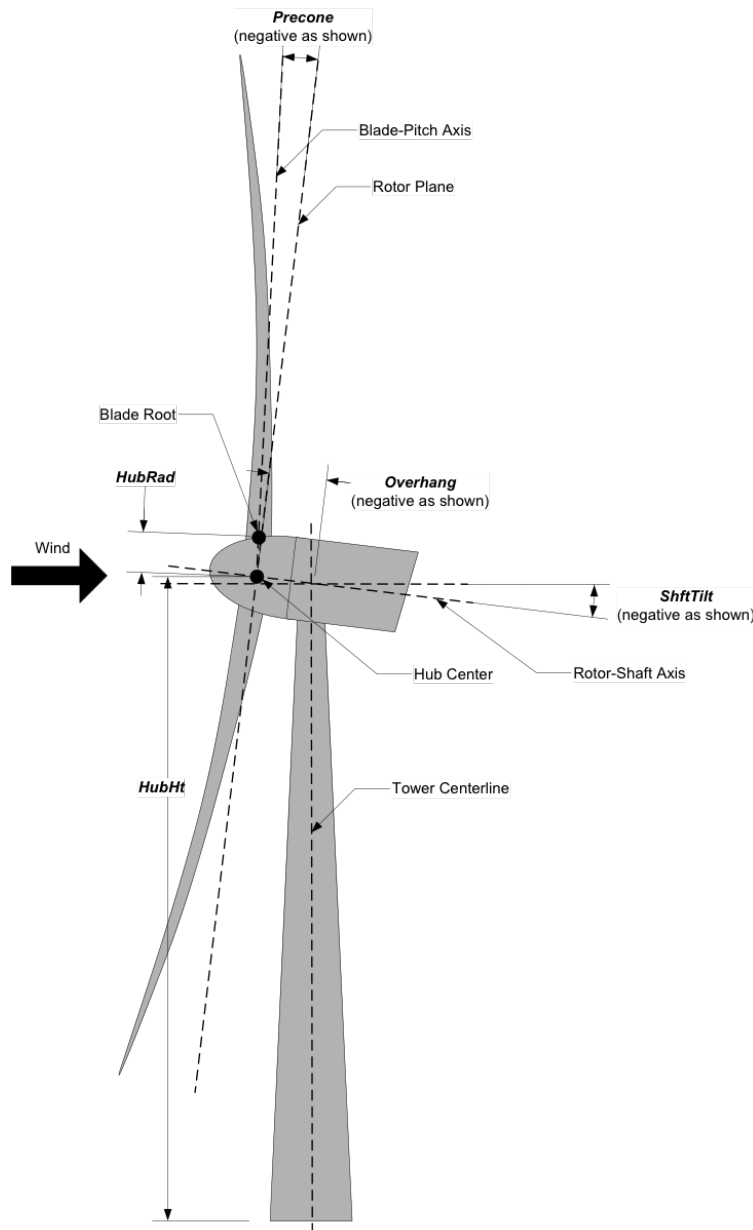


Fig. 4.4: Definition of basic turbine geometry.

Additionally, the user needs to provide the origin of the turbine base at $t=0$ (*BaseOriginInit*). An example of basic input is given below:


```

----- Turbine(1) Geometry -----
      True      BasicHAWTFormat(1) - Flag to switch between basic or generic input
↪format {True: next 7 lines are basic inputs, False: Base/Twr/Nac/Hub/Bld geometry and
↪motion must follow}
      0,0,0      BaseOriginInit(1) - Coordinates of turbine base in global coordinates (m)
      3          NumBlades(1)      - Number of blades (-)
      3.         HubRad(1)         - Hub radius (m)
      140.82513  HubHt(1)          - Hub height (m)
      -7         Overhang(1)       - Overhang (m)
      -6         ShftTilt(1)        - Shaft tilt (deg)
      -4         Precone(1)         - Blade precone (deg)
      3.09343    Twr2Shft(1)       - Vertical distance from the tower-top to the rotor
↪shaft (m)

```

- advanced (*BasicHAWTFormat=False*): The position and orientation of the tower base, nacelle, hub, and individual blades can be arbitrarily defined. This can be used for HAWT and any other turbine concepts. The definition of the different frames are given in Fig. 4.3. The position (*BaseOriginInit*) and orientation (*BaseOrientationInit*) of the turbine base frame are defined with respect to the global frame. Orientations are given using the values of three successive rotations (x-y-z Euler angle sequence). If the base undergoes a motion, the orientation of the base frame will consist of the time-varying rotations followed by these initial rotations.

A flag indicating whether the turbine has a tower is given on the next line (*HasTower*). This flag currently affects the VTK outputs and does not have an impact on AeroDyn yet. The user still has to provide tower input data in AeroDyn for each turbine (see Section 4.2.1). The next line indicates which projection AeroDyn is to use in its calculation. It is recommended to use *HAWTprojection=True* for HAWT, which is the default projection used in AeroDyn (projects on the coned-pitched axis). For other rotor concepts, set *HAWTprojection=False*. The following lines indicate the position and orientations of the tower, nacelle and hub.

The tower and the nacelle are defined with respect to the turbine base (t) origin and frame. The tower top is assumed to coincide with the nacelle origin. The tower stations defined in the AeroDyn input file are assumed to be given with respect to the tower origin, unlike OpenFAST which uses ground/MSL as a reference (see Section 4.2.1). The hub is defined with respect to the nacelle origin and frame (n).

The definitions of the blades follow, starting with the number of blades *NumBlades*. A rotor with zero blades is supported and can be used to model an isolated tower. If tower shadow/potential is used in AeroDyn, then the isolated tower will disturb the flow of the vortex wake when OLAF is used. When BEM is used, the flow of the blades of a given turbine are disturbed only by that turbine's tower. The inputs for turbine *i* and blade *j* are labelled (*i_j*). The origin (*BldOrigin_h*) and orientation (*BldOrientation_h*) of each blade are given with respect to the hub origin and frame (h). Hub radius inputs (*BldHubRad_Bl*) are provided for convenience. They will effectively offset the blades origin along the z_b axis. An example of input for an advanced geometry definition is given below. This example corresponds to typical values for a 3-bladed upwind HAWT, with 6 degrees of tilt (-6 in OpenFAST) and -4 degrees of precone (blades inclined upstream).

```

----- Turbine(1) Geometry -----
      False     BasicHAWTFormat(1) - Flag to switch between basic or generic input
↪format {True: next 7 lines are basic inputs, False: Base/Twr/Nac/Hub/Bld geometry and
↪motion must follow}
      0,0,0      BaseOriginInit(1)      - x,y,z coordinates of turbine base origin (m)
      0,0,0      BaseOrientationInit(1) - successive rotations (theta_x, theta_y, theta_
↪z) defining initial orientation of the base frame from the global frame (e.g. roll,
↪tilt, yaw) (deg)
      True       HasTower(1)            - True if turbine has a tower (flag)
      True       HAWTprojection(1)      - True if turbine is a horizontal axis turbine
↪(for AeroDyn projections) (flag)

```

(continues on next page)

(continued from previous page)

```

0,0,0          TwrOrigin_t(1)      - Coordinate of tower base in base coordinates
↳[used only when HasTower is True] (m)
0,0,137        NacOrigin_t(1)      - x,y,z coordinates of nacelle origin (and tower
↳top) from base, in base coordinates (m)
-6.96,0.,3.82  HubOrigin_n(1)      - x,y,z coordinates of hub origin from nacelle
↳origin, in nacelle coordinates (m)
0,6,0          HubOrientation_n(1)  - successive rotations (theta_x, theta_y, theta
↳z) defining initial orientation of the hub frame from the nacelle frame (e.g. roll,
↳tilt, yaw). The x axis needs to be aligned with the rotational speed. (deg)
----- Turbine(1) Blades -----
3              NumBlades(1)         - Number of blades for current rotor (-)
0,0,0          BldOrigin_h(1_1)     - Origin of blade 1 wrt. hub origin in hub
↳coordinates (m)
0,0,0          BldOrigin_h(1_2)     - Origin of blade 2 wrt. hub origin in hub
↳coordinates (m)
0,0,0          BldOrigin_h(1_3)     - Origin of blade 3 wrt. hub origin in hub
↳coordinates (m)
0,-4,0         BldOrientation_h(1_1) - successive rotations (theta_x, theta_y, theta_z)
↳defining initial orientation of the blade frame from the hub frame such that the "z"
↳is along span, "y" along trailing edge without pitch (azimuth, precone, pitch) (deg)
120,-4,0       BldOrientation_h(1_2) - successive rotations (theta_x, theta_y, theta_z)
↳defining initial orientation of the blade frame from the hub frame such that the "z"
↳is along span, "y" along trailing edge without pitch (azimuth, precone, pitch) (deg)
240,-4,0       BldOrientation_h(1_3) - successive rotations (theta_x, theta_y, theta_z)
↳defining initial orientation of the blade frame from the hub frame such that the "z"
↳is along span, "y" along trailing edge without pitch (azimuth, precone, pitch) (deg)
3.0            BldHubRad_bl(1_1)     - z-offset in blade coordinates of blade 1 where
↳radial input data start (m)
3.0            BldHubRad_bl(1_2)     - z-offset in blade coordinates of blade 2 where
↳radial input data start (m)
3.0            BldHubRad_bl(1_3)     - z-offset in blade coordinates of blade 3 where
↳radial input data start (m)

```

Turbine motion definition

The definition of the turbine motion is only used when *AnalysisType=1*, but must always be present in the input file.

The base motion is given in the same way for basic or advanced geometries. The motion of the base may be: fixed (*BaseMotionType=0*), sinusoidal (*BaseMotionType=1*) or arbitrary (*BaseMotionType=2*). The turbine base motion is applied at each time step before applying the initial position and orientation of the turbine base. A sinusoidal motion implies that one degree of freedom (*DegreeOfFreedom*) of the turbine base is moving according to a sine function of a given amplitude (*Amplitude*) and frequency (*Frequency*, in Hz), with zero phase. The 6 possible degrees of freedom correspond to translations or rotations of the base frame in global coordinates (g) (e.g. surge, sway, heave, roll, pitch, yaw). An arbitrary motion is specified via a CSV file (*BaseMotionFileName*) which contains 19 columns: time, 3 translations (global), three successive rotations (global), 3 translation velocities, 3 rotational velocities (omega, in global), 3 translational accelerations and 3 rotational accelerations (alpha, in global). Example of arbitrary input files are given in [Section 4.2.1](#). The time vector in the motion file has to be ascending, but does not need to be linear. Linear interpolation is used by the driver to determine inputs at a given time. The displacements/orientations, velocities, and accelerations are not checked internally for consistency.

An example of inputs for a sinusoidal surge motion is given below:

```

----- Turbine(1) Motion [used only when AnalysisType=1] -----
1          BaseMotionType(1)        - Type of motion prescribed for this base {0: fixed, 1:

```

(continues on next page)

(continued from previous page)

```

↪ Sinusoidal motion, 2: arbitrary motion} (flag)
1      DegreeOfFreedom(1)    - {1:xg, 2:yg, 3:zg, 4:theta_xg, 5:theta_yg, 6:theta_zg}
↪ [used only when BaseMotionType=1] (flag)
5.0    Amplitude(1)         - Amplitude of sinusoidal motion [used only when
↪ BaseMotionType=1] (m or rad)
0.1    Frequency(1)         - Frequency of sinusoidal motion [used only when
↪ BaseMotionType=1] (Hz)
"unused" BaseMotionFileName(1) - Filename containing arbitrary base motion (19
↪ columns: Time, x, y, z, theta_x, ..., alpha_z) [used only when BaseMotionType=2]

```

The different inputs for the basic and advanced geometries are given below:

- basic: The motion of a basic turbine consists of a constant nacelle yaw (*NacYaw*, positive rotation of the nacelle about the vertical tower axis, counterclockwise when looking downward), rotor speed (*RotSpeed*, positive clockwise looking downwind), and blade pitch (*BldPitch*, negative around z_b). Examples are given below:

```

0      NacYaw(1)             - Yaw angle (about z_t) of the nacelle (deg)
7      RotSpeed(1)           - Rotational speed of rotor in rotor coordinates (rpm)
1      BldPitch(1)           - Blades pitch (deg)

```

- advanced: When an advanced geometry is provided and when the number of blades is non-zero, the motion section contains options for the nacelle motion, rotor motion and individual blade pitch motion. The syntax for each of these motions consists of defining a type (fixed or time-varying), a value for the fixed case or a file for the time-varying case. The input files are CSV files containing time, position, speed and acceleration. Examples of files are given in [Section 4.2.1](#). The displacements/orientations, velocities, and accelerations are not checked internally for consistency. The time vector in the motion file has to be ascending, but does not need to be linear. Linear interpolation is used by the driver to determine inputs at a given time. The angular and rotational data in the CSV file are defined in rad and rad/s, whereas they are defined in deg and rpm in the driver input file. An example is given below for a fixed rotational speed:

```

0      NacMotionType(1)      - Type of motion prescribed for the nacelle {0: fixed,
↪ yaw, 1: time varying yaw angle} (flag)
0      NacYaw(1)             - Yaw angle (about z_t) of the nacelle [user only when
↪ NacMotionType=0] (deg)
"unused" NacMotionFileName(1) - Filename containing yaw motion [used only when
↪ NacMotionType=1]
0      RotMotionType(1)      - Type of motion prescribed for this rotor {0: constant,
↪ rotation, 1: time varying rotation} (flag)
6.0    RotSpeed(1)           - Rotational speed of rotor in rotor coordinates [used
↪ only when RotorMotionType=0] (rpm)
"unused" RotMotionFileName(1) - Filename containing rotor motion [used only when
↪ RotorMotionType=1]
0      BldMotionType(1)      - Type of pitch motion prescribed for the blades {0:
↪ fixed, 1: time varying pitch} (flag)
0      BldPitch(1_1)         - Blade 1 pitch [used only when BldMotionType=0] (deg)
0      BldPitch(1_2)         - Blade 2 pitch [used only when BldMotionType=0] (deg)
0      BldPitch(1_3)         - Blade 3 pitch [used only when BldMotionType=0] (deg)
"unused" BldMotionFileName(1_1) - Filename containing blade pitch motion [used only
↪ when BldMotionType=1]
"unused" BldMotionFileName(1_2) - Filename containing blade pitch motion [used only
↪ when BldMotionType=1]
"unused" BldMotionFileName(1_3) - Filename containing blade pitch motion [used only
↪ when BldMotionType=1]

```

Time-dependent analysis

Time-dependent analyses are used to vary a few standard variables during the simulation. The variables are: reference wind speed (*HWndSpeed*), power law exponent (*PLExp*), rotor speed (*RotSpd*), collective pitch (*Pitch*), and nacelle yaw (*Yaw*). The time series of each variable are provided in a CSV file (*TimeAnalysisFileName*). Time-dependent analyses are selected using *AnalysisType*=2. They are restricted to one turbine (*numTurbines*=1).

```
----- Time-dependent Analysis [used only when AnalysisType=2 and numTurbines=1] -----
"TimeSeries.csv" TimeAnalysisFileName - Filename containing time series (6 column: Time,
↳ HWndSpeed, PLExp, RotSpd, Pitch, Yaw).
```

Combined-case analyses

Combined-case analyses are used to run parametric studies in one single run. They are selected using *AnalysisType*=3, and are restricted to one turbine (*numTurbines*=1). The variables that can be changed for each simulation are: reference wind speed (*HWndSpeed*), power law exponent (*PLExp*), rotor speed (*RotSpd*), collective pitch (*Pitch*), nacelle yaw (*Yaw*), time step (*dT*), simulation time (*Tmax*), and sinusoidal motion parameters (degree of freedom, *DOF*, amplitude and frequency). When *DOF*=0, the turbine base is fixed.

```
----- Combined-Case Analysis [used only when AnalysisType=3 and numTurbines=1] -----
      4 NumCases      - Number of cases to run
HWndSpeed PLExp RotSpd Pitch Yaw dT Tmax DOF Amplitude Frequency
(m/s)      (-)  (rpm)  (deg) (deg) (s)  (s)  (-)  (m or rad) (Hz)
  8.      0.0    6.    0.    0.    1.0 100    0    0        0.0
  8.      0.0    6.    0.    0.    1.0 100    0    0        0.0
  9.      0.1    7.    1.    0.    0.5  50    1    5.0       0.1
  9.      0.2    8.    2.    0.    0.5  50    1    2.0       0.2
```

Outputs

The output section controls the format of the tabular output file and VTK files, similar to the OpenFAST outputs. The user can control the hub radius and nacelle dimension for the VTK visualization. The hub is represented as a sphere of radius (*VTKHubRad*), and the nacelle with a parallelepiped defined using an origin and three lengths parallel to the nacelle coordinates (*VTKNacDim*).

```
----- Output Settings -----
"ES15.8E2" OutFmt - Format used for text tabular output, excluding the time
↳ channel. Resulting field should be 10 characters. (quoted string)
2 OutFileFmt - Format for tabular (time-marching) output file (switch)
↳ {1: text file [<RootName>.out], 2: binary file [<RootName>.outb], 3: both}
0 WrVTK - VTK visualization data output: (switch) {0=none; 1=init;
↳ 2=animation}
2 VTKHubRad - HubRadius for VTK visualization (m)
-1,-1,-1,2,2,2 VTKNacDim - Nacelle Dimension for VTK visualization x0,y0,z0,Lx,Ly,Lz
↳ (m)
```

AeroDyn inputs for multiple turbines

No changes are required to the AeroDyn input files when one turbine is used. To minimize the impact of the multiple-turbines implementation, the driver currently uses only one AeroDyn input file for all turbines. This means that the AeroDyn options are currently the same for all rotors.

The definition of the blade files and tower inputs needs to be adapted when more than three blades are used and more than one turbine is used.

Blade files

The legacy AeroDyn format requires a minimum of three blade file names. For this reason, the blades of all rotors are currently indicated successively in the *ADBlFile* list. The list is populated by looping on turbines and turbine blades, with the blade index being the fastest index. For now, the number of stations have to be the same for all blades.

An example is given below for two turbines, the first one having 3 blades, the second 2 blades:

```
===== Rotor/Blade Properties
↳=====
True          UseBlCm      - Include aerodynamic pitching moment in calculations?
↳ (flag)
"AD_Turbine1_blade1.dat" ADBlFile(1) - Name of file containing distributed aerodynamic
↳ properties for Blade #1 (-)
"AD_Turbine1_blade1.dat" ADBlFile(2) - Name of file containing distributed aerodynamic
↳ properties for Blade #2 (-)
"AD_Turbine1_blade3.dat" ADBlFile(3) - Name of file containing distributed aerodynamic
↳ properties for Blade #3 (-)
"AD_Turbine2_blade1.dat" ADBlFile(4) - Name of file containing distributed aerodynamic
↳ properties for Blade #4 (-)
"AD_Turbine2_blade2.dat" ADBlFile(5) - Name of file containing distributed aerodynamic
↳ properties for Blade #5 (-)
```

Aerodynamic tower inputs

The entire tower input section of AeroDyn has to be reproduced for each turbine, including turbines that are set not to have a tower (*hasTower=False*). The number of stations may differ for each turbine. The tower stations defined in the AeroDyn input file are assumed to be given with respect to the tower origin, unlike OpenFAST which uses ground/MSL as a reference.

An example is given below for two turbines:

```
===== Turbine(1) Tower Influence and Aerodynamics
↳===== [used only when TwrPotent/=0,
↳ TwrShadow=True, or TwrAero=True]
2 NumTwrNds - Number of tower nodes used in the analysis (-) [used only when
↳ TwrPotent/=0, TwrShadow=True, or TwrAero=True]
TwrElev TwrDiam TwrCd TwrTI
(m)      (m)      (-)      (-)
0.0      2.0      1.0      0.1
10.0     1.0      1.0      0.1
===== Turbine(2) Tower Influence and Aerodynamics
↳===== [used only when TwrPotent/=0,
↳ TwrShadow=True, or TwrAero=True]
3 NumTwrNds - Number of tower nodes used in the analysis (-) [used only when
↳ TwrPotent/=0, TwrShadow=True, or TwrAero=True]
TwrElev TwrDiam TwrCd TwrTI
```

(continues on next page)

(continued from previous page)

(m)	(m)	(-)	(-)
0.0	4.0	1.0	0.1
15.0	3.0	1.0	0.1
30.0	2.0	1.0	0.1

Examples of driver input files

Working examples that use the different features of the driver are given in the r-test repository:

- [Dev branch](#) .
- [Main branch](#) .

Main Driver Input Files

An example of an AeroDyn driver for a basic inflow, basic HAWT, and combined case analyses is given below:

```

----- AeroDyn Driver Input File -----
Three bladed wind turbine, using basic geometry input
----- Input Configuration -----
False      Echo      - Echo input parameters to "<rootname>.ech"?
      3      AnalysisType - {1: multiple turbines, one simulation, 2: one turbine,
↪ one time-dependent simulation, 3: one turbine, combined cases}
      11.0    TMax      - Total run time [used only when AnalysisType/=3] (s)
      0.5     DT        - Simulation time step [used only when AnalysisType/=3] (s)
"./AD.dat"  AeroFile - Name of the primary AeroDyn input file
----- Inflow Data -----
      0      CompInflow - Compute inflow wind velocities (switch) {0=Steady Wind;
↪ 1=InflowWind}
"unused"    InflowFile - Name of the InflowWind input file [used only when
↪ CompInflow=1]
      9.0     HWindSpeed - Horizontal wind speed [used only when CompInflow=0 and
↪ AnalysisType=1] (m/s)
      140     RefHt      - Reference height for horizontal wind speed [used only
↪ when CompInflow=0] (m)
      0.10    PLExp      - Power law exponent [used only when CompInflow=0 and
↪ AnalysisType=1] (-)
----- Turbine Data -----
1          NumTurbines - Number of turbines
----- Turbine(1) Geometry -----
      True    BasicHAWTFormat(1) - Flag to switch between basic or generic input
↪ format {True: next 7 lines are basic inputs, False: Base/Twr/Nac/Hub/Bld geometry and
↪ motion must follow}
      0,0,0    BaseOriginInit(1) - Coordinate of tower base in base coordinates (m)
      3        NumBlades(1)      - Number of blades (-)
      3.        HubRad(1)        - Hub radius (m)
      140.82513 HubHt(1)         - Hub height (m)
      -7       Overhang(1)       - Overhang (m)
      -6       ShftTilt(1)       - Shaft tilt (deg)
      -4       Precone(1)       - Blade precone (deg)
      3.09343   Twr2Shft(1)      - Vertical distance from the tower-top to the rotor

```

(continues on next page)

(continued from previous page)

```

↳shaft (m)
----- Turbine(1) Motion [used only when AnalysisType=1] -----
1          BaseMotionType(1)      - Type of motion prescribed for this base {0:
↳fixed, 1: Sinusoidal motion, 2: arbitrary motion} (flag)
1          DegreeOfFreedom(1)     - {1:xg, 2:yg, 3:zg, 4:theta_xg, 5:theta_yg,
↳6:theta_zg} [used only when BaseMotionType=1] (flag)
5.0        Amplitude(1)           - Amplitude of sinusoidal motion [used only when
↳BaseMotionType=1] (m or rad)
0.1        Frequency(1)          - Frequency of sinusoidal motion [used only when
↳BaseMotionType=1] (Hz)
""         BaseMotionFileName(1) - Filename containing arbitrary base motion (19
↳columns: Time, x, y, z, theta_x, ..., alpha_z) [used only when BaseMotionType=2]
0          NacYaw(1)              - Yaw angle (about z_t) of the nacelle (deg)
7          RotSpeed(1)            - Rotational speed of rotor in rotor coordinates
↳(rpm)
1          BldPitch(1)            - Blade 1 pitch (deg)
----- Time-dependent Analysis [used only when AnalysisType=2, numTurbines=1] -----
"unused"   TimeAnalysisFileName - Filename containing time series (6 column: Time,
↳HWndSpeed, PLExp, RotSpd, Pitch, Yaw).
----- Combined-Case Analysis [used only when AnalysisType=3, numTurbines=1] -----
4 NumCases - Number of cases to run
HWndSpeed PLExp RotSpd Pitch Yaw dT Tmax DOF Amplitude Frequency
(m/s)      (-)   (rpm) (deg) (deg) (s) (s) (-) (-) (Hz)
8.0        0.0    6.    0.    0.    1.0 100 0 0 0
8.0        0.0    6.    0.    0.    1.0 100 0 0 0
9.0        0.1    7.    1.    0.    0.5 51 1 5.0 0.1
9.0        0.2    8.    2.    0.    0.51 52 1 2.0 0.2
----- Output Settings -----
"ES15.8E2" OutFmt - Format used for text tabular output, excluding the time
↳channel. Resulting field should be 10 characters. (quoted string)
2          OutFileFmt - Format for tabular (time-marching) output file (switch)
↳{1: text file [<RootName>.out], 2: binary file [<RootName>.outb], 3: both}
0          WrVTK - VTK visualization data output: (switch) {0=none; 1=init;
↳2=animation}
2          VTKHubRad - HubRadius for VTK visualization (m)
-1,-1,-1,2,2,2 VTKNacDim - Nacelle Dimension for VTK visualization x0,y0,z0,Lx,Ly,Lz
↳(m)

```

Motion input files

The time vector in the motion files has to be ascending, but does not need to be linear. Linear interpolation is used by the driver to determine inputs at a given time.

Arbitrary base motion file:

```

time_[s] , x_[m] , y_[m] , z_[m] , theta_x_[rad] , theta_y_[rad] , theta_z_
↳[rad] , xdot_[m/s] , ydot_[m/s] , zdot_[m/s] , omega_x_g_[rad/s] , omega_y_g_[rad/s] ,
↳omega_z_g_[rad/s] , xddot_[m^2/s] , yddot_[m^2/s] , zddot_[m^2/s] , alpha_x_g_[rad/s] ,
↳alpha_y_g_[rad/s] , alpha_z_g_[rad/s]
0.000000 , 0.000000 , 0.000000 , 0.000000 , 0.000000 , 0.000000 , 0.000000
↳ , 0.000000 , 0.000000 , 10.053096 , 0.000000 , 0.000000 , 0.

```

(continues on next page)

(continued from previous page)

```

→0.000000      , 0.000000      , 0.000000      , -0.000000      , 0.000000      ,
→0.000000      , 0.000000
0.100000 , 0.000000 , 0.000000 , 0.963507 , 0.000000      , 0.000000      , 0.000000
→ , 0.000000      , 0.000000      , 8.809596      , 0.000000      , 0.000000      , 0.
→0.000000      , 0.000000      , 0.000000      , -24.344157      , 0.000000      ,
→0.000000      , 0.000000

```

Yaw motion file:

```

time_[s] , yaw_[rad] , yaw_rate_[rad/s] , yaw_acc_[rad/s^2]
0.000000 , 0.000000 , 0.000000      , 0.000000
0.100000 , 0.007277 , 0.212647      , 4.029093

```

Rotor motion file:

```

time_[s] , azimuth_[rad] , omega_[rad/s] , rotacc_[rad/s^2]
0.000000 , 0.000000      , 0.000000      , 0.000000
0.100000 , 0.000000      , 0.000000      , 0.000000

```

Pitch motion file:

```

time_[s] , pitch_[rad] , pitch_rate_[rad/s] , pitch_acc_[rad/s^2]
0.000000 , 0.000000      , 0.000000      , 0.000000
0.100000 , 0.000000      , 0.000000      , 0.000000
0.200000 , 0.000000      , 0.000000      , 0.000000

```

AeroDyn Theory

This theory manual is work in progress, please refer to the AeroDyn 14 manual for more details [ad-MH05]. Many changes have occurred since AeroDyn 14 (e.g. BEM formulation, coordinate system used in the BEM equations, dynamic stall, dynamic BEM), but these changes are not yet documented here.

Steady BEM

The steady blade element momentum (BEM) equations are solved as a constrained equation, and the formulation follows the description from Ning [ad-Nin14].

Tower shadow models

Powles tower shadow model (**TwrShadow=1**) is given by:

$$u_{TwrShadow} = -\frac{C_d}{\sqrt{\bar{r}}} \cos\left(\frac{\pi/2\bar{y}}{\sqrt{\bar{r}}}\right)^2$$

where $\bar{r} = \sqrt{\bar{x}^2 + \bar{y}^2}$.

Eames tower shadow model (**TwrShadow=2**) is given by:

$$u_{TwrShadow} = -\frac{C_d}{TI \bar{x} \sqrt{2\pi}} \exp\left(-\frac{1}{2} \left(\frac{\bar{y}}{TI \bar{x}}\right)^2\right)$$

where TI is the turbulence intensity at the tower node.

Unsteady aerodynamics

The Unsteady Aerodynamic (UA) models account for flow hysteresis, including unsteady attached flow, trailing-edge flow separation, dynamic stall, and flow reattachment. *Dynamic stall* refers to rapid aerodynamic changes that may bring about or delay stall behavior [ad-Bra17]. Rapid changes in wind speed (for example, when the blades pass through the tower shadow) cause a sudden detachment and then reattachment of air flow along the airfoil. Such effects at the blade surface cannot be predicted with steady state aerodynamics, but may affect turbine operation, not only when the blades encounter tower shadow, but also during operation in skewed flows and turbulent wind conditions. Dynamic stall effects occur on time scales of the order of the time for the relative wind at the blade to traverse the blade chord, approximately $c/\Omega r$. For large wind turbines, this might be on the order of 0.5 seconds at the blade root to 0.001 seconds at the blade tip. Dynamic stall can result in high transient forces as the wind speed increases, but stall is delayed.

Theory

The different dynamic stall models implemented in AeroDyn are presented below.

Notations and Definitions

See Section 4.2.1 for a comprehensive description of all the inputs present in the profile input file (including some of the ones repeated below).

The airfoil section coordinate system and main variables are presented in Fig. 4.5 and further described below:

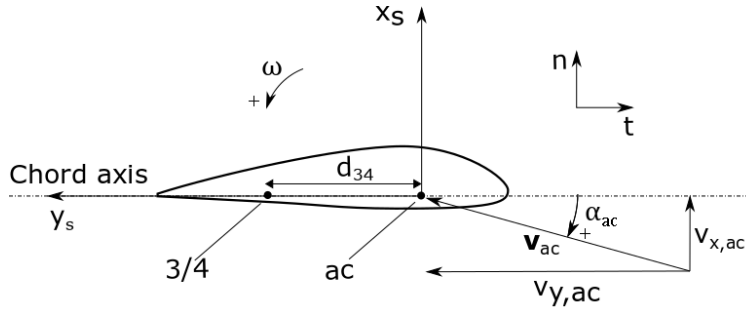


Fig. 4.5: Definition of airfoil section coordinate system used in the unsteady aerodynamics module

- Aerodynamic Center (AC): point of the airfoil cross section where the aerodynamic forces and moment are assumed to act. Usually close to the 1/4 chord point for a regular airfoil and at the center for a circular cross section
- “3/4” chord point: in the original formulation this point refers to the point on the chord axis located 3/4 chord behind the leading edge. This concept is here generalized to the point located mid-way between the aerodynamic center and the trailing edge, to account for aerodynamic center positions that differ strongly from a 1/4 chord point. The notation 3/4 is kept in this document.
- ω : rotational speed of the airfoil section (pitching/torsional rate) positive around z.
- \mathbf{v}_{ac} : velocity vector at the aerodynamic center $\mathbf{v}_{ac} = [v_{x,ac}, v_{y,ac}]$ (coordinates assumed to be expressed in the airfoil section coordinate system)
- \mathbf{v}_{34} : velocity vector at the 3/4 chord point $\mathbf{v}_{34} = [v_{x,34}, v_{y,34}]$ (coordinates assumed to be expressed in the airfoil section coordinate system) The velocity is obtained from the velocity at the 1/4 chord point and the rotational speed of the section: $\mathbf{v}_{34} = \mathbf{v}_{ac} + \omega d_{34} \hat{\mathbf{x}}_s$ where d_{34} is the distance between the aerodynamic center and the 3/4 chord point.

- U_{ac} : velocity norm at the aerodynamic center. $U_{ac} = \|\mathbf{v}_{ac}\| = \sqrt{v_{x,ac}^2 + v_{y,ac}^2}$
- α_{ac} : angle of attack at the aerodynamic center $\alpha_{ac} = \text{atan2}(v_{x,ac}, v_{y,ac})$
- α_{34} : angle of attack at the 3/4 chord point $\alpha_{34} = \text{atan2}(v_{x,34}, v_{y,34})$
- \mathbf{x} : the vector of states used by the continuous formulations
- c : airfoil chord
- $C_l^{st}, C_d^{st}, C_m^{st}$: static airfoil coefficients
- α_0 : angle of attack at zero lift, $C_l^{st}(\alpha_0) = 0$
- α_1 : angle of attack close to positive stall.
- α_2 : angle of attack close to negative stall.
- $C_{l,\alpha}$: slope of the steady lift curve about α_0 .
- $f_s^{st}(\alpha)$: is the steady separation function, determined from the lift curve $C_l^{st}(\alpha)$ (see below, and e.g. [ad-HGAM04])
- A_1, A_2, b_1, b_2 : are four constants, characteristic of the propagation of the wake vorticity (Wagner constants)

Time constants:

- $T_u(t) = \frac{c}{2U_{ac}(t)} \in [0.001, 50]$: Time for the flow to go over half the airfoil section. The value is plateaued to avoid unphysical values.
- $T_{f,0}$: Dimensionless time constant associated with leading edge separation. Default is 3.
- $T_{p,0}$: Dimensionless time constant for the boundary-layer, leading edge pressure gradient. Default is 1.7

Separation function:

The steady separation function, f_s^{st} , is defined as the separation point on a flat plate for a potential Kirchhoff flow [ad-HGAM04]:

$$\text{Close to } \alpha_0, f_s^{st}(\alpha) = \min \left\{ \left[2 \sqrt{\frac{C_l^{st}(\alpha)}{C_{l,\alpha}(\alpha - \alpha_0)}} - 1 \right]^2, 1 \right\}, \quad \text{away from } \alpha_0, f_s^{st}(\alpha) = 0$$

When $\alpha = \alpha_0$, $f_s^{st}(\alpha_0) = 1$. Away from α_0 , the function drops progressively to 0. As soon as the function reaches 0 on both sides of α_0 , then f_s^{st} is kept at the constant value 0.

Note that for UAMod=5, a different separation function is formed. We define an offset for the C_n function, cn_offset , where $C_{n,offset} = \frac{C_n(\alpha^{Lower}) + C_n(\alpha^{Upper})}{2}$. Then, the separation function is a value between 0 and 1, given by the following equation:

$$f_s^{st}(\alpha) = \left[2 \max \left\{ \frac{1}{4}, \sqrt{\frac{C_n^{st}(\alpha) - C_{n,offset}}{C_n^{fullyAttached}(\alpha) - C_{n,offset}}} \right\} - 1 \right]^2$$

with the fully-attached C_n curve defined as C_n between α^{Lower} and α^{Upper} and linear functions outside of that range:

$$C_n^{fullyAttached}(\alpha) = \begin{cases} C_n(\alpha^{Upper}) + C_n^{slope}(\alpha^{Upper}) \cdot (\alpha - \alpha^{Upper}) & \alpha > \alpha^{Upper} \\ C_n(\alpha) & \alpha^{Lower} \leq \alpha \leq \alpha^{Upper} \\ C_n(\alpha^{Lower}) + C_n^{slope}(\alpha^{Lower}) \cdot (\alpha - \alpha^{Lower}) & \alpha < \alpha^{Lower} \end{cases}$$

Note that to avoid numerical issues at the ± 180 degree boundary, this function changes slope when the separation function is 0 above α^{Upper} and below α^{Lower} . This allow the fully-attached linear sections to be periodic and avoid numerical issues with large magnitudes of angle of attack.

Inviscid and fully separated lift coefficient:

The inviscid lift coefficient is $C_{l,\text{inv}} = C_{l,\alpha}(\alpha - \alpha_0)$. The fully separated lift coefficient may be modelled in different ways ([ad-Bra17]). In most engineering models, the slope of the fully separated lift coefficient around α_0 is $C_{l,\alpha}/2$. In the Unsteady AeroDynamics sub-module, the fully separated lift coefficient is derived from the steady separation function as:

$$C_{l,\text{fs}}(\alpha) = \frac{C_l^{st}(\alpha) - C_{l,\alpha}(\alpha - \alpha_0)f_s^{st}(\alpha)}{1 - f_s^{st}(\alpha)} \text{ when } f_s^{st} \neq 1, \quad C_{l,\text{fs}}(\alpha) = \frac{C_l^{st}(\alpha)}{2} \text{ when } f_s^{st} = 1$$

Beddoes-Leishman type models (UAMod=2,3)

The Beddoes-Leishman model account for attached flows and trailing edge stall [ad-LB89].

Two variants are implemented in the Unsteady Aerodynamic module. These two (compressible) models are currently described in the following reference: [ad-DH19]. The models use C_n and C_c as main physical quantities. The models use discrete states and cannot be used with linearization.

Beddoes-Leishman 4-states model (UAMod=4)

The 4-states (incompressible) dynamic stall model from Hansen-Gaunaa-Madsen (HGM) is described in [ad-HGAM04] and enabled using UAMod=4. The model uses C_l as main physical quantity. Linearization of the model will be available in the future.

State equation: The state equation of the model is:

$$\begin{aligned} \dot{x}_1 &= -T_u^{-1}b_1 x_1 + T_u^{-1}b_1 A_1 \alpha_{34} \\ \dot{x}_2 &= -T_u^{-1}b_2 x_2 + T_u^{-1}b_2 A_2 \alpha_{34} \\ \dot{x}_3 &= -T_p^{-1}x_3 + T_p^{-1}C_l^p \\ \dot{x}_4 &= -T_f^{-1}x_4 + T_f^{-1}f_s^{st}(\alpha_F), \quad x_4 \in [0, 1] \end{aligned}$$

with

$$\begin{aligned} \alpha_E(t) &= \alpha_{34}(t)(1 - A_1 - A_2) + x_1(t) + x_2(t) \\ C_L^p(t) &= C_{l,\alpha}(\alpha_E(t) - \alpha_0) + \pi T_u(t)\omega(t) \\ \alpha_F(t) &= \frac{x_3(t)}{C_{l,\alpha}} + \alpha_0 \end{aligned}$$

Output equation: The unsteady airfoil coefficients $C_{l,\text{dyn}}$, $C_{d,\text{dyn}}$, $C_{m,\text{dyn}}$ are obtained from the states as follows:

$$\begin{aligned} C_{l,\text{dyn}}(t) &= x_4(\alpha_E - \alpha_0)C_{l,\alpha} + (1 - x_4)C_{l,\text{fs}}(\alpha_E) + \pi T_u \omega \\ C_{d,\text{dyn}}(t) &= C_d(\alpha_E) + (\alpha_{ac} - \alpha_E)C_{l,\text{dyn}} + [C_d(\alpha_E) - C_d(\alpha_0)] \Delta C_{d,f}'' \\ C_{m,\text{dyn}}(t) &= C_m(\alpha_E) - \frac{\pi}{2} T_u \omega \end{aligned}$$

with:

$$\Delta C_{d,f}'' = \frac{\sqrt{f_s^{st}(\alpha_E)} - \sqrt{x_4}}{2} - \frac{f_s^{st}(\alpha_E) - x_4}{4}, \quad x_4 \geq 0$$

Beddoes-Leishman 5-states model (UAMod=5)

The 5-states (incompressible) dynamic stall model is similar to the Beddoes-Leishman 4-states model (UAMod=4), but adds a 5th state to represent vortex generation. It is enabled using UAMod=5. The model uses C_n and C_c as main physical quantities. Linearization of the model will be available in the future.

Oye model (UAMod=6)

Oye's dynamic stall model is a one-state (continuous) model, formulated in [ad-Oye91] and described e.g. in [ad-Bra17]. The model attempts to capture trailing edge stall. Linearization of the model will be available in the future.

State equation: Oye's dynamic stall model uses one state, $x = [f_s]$ where f_s is the unsteady separation function. The state equation is a first-order differential equation:

$$\frac{df_s(t)}{dt} = -\frac{1}{T_f} f_s(t) + \frac{1}{T_f} f_s^{st}(\alpha_{34}(t))$$

where $T_f = T_{f,0} T_u$ is the time constant of the flow separation and f_s^{st} is the steady state separation function described in Section 4.2.1. The value $T_{f,0}$ is usually chosen around 6 (different from the default value). It is readily seen that f_s reaches the value f_s^{st} when the system is in a steady state (i.e. when $\frac{df_s(t)}{dt} = 0$).

Output equation: The unsteady lift coefficient is computed as a linear combination of the inviscid lift coefficient, $C_{l,inv}$, and the fully separated lift coefficient $C_{l,fs}$. Both of these lift coefficients are determined from the steady lift coefficient, usually provided as tabulated data, noted $C_l^{st}(\alpha)$, where the superscript *st* stands for “steady”. The unsteady lift coefficient is modelled as:

$$C_{l,dyn}(\alpha_{34}, t) = f_s(t) C_{l,inv}(\alpha_{34}) + (1 - f_s(t)) C_{l,fs}(\alpha_{34})$$

where α_{34} is the instantaneous angle of attack at the 3/4 chord. f_s is seen to act as a relaxation factor between the two flow situations.

Boeing-Vertol model (UAMod=7)

The Boeing-Vertol is mentioned in the following paper [ad-MB11]. Details of the model were omitted in this reference, so the documentation presented here is inspired from the implementation done in the vortex code CACTUS, which was reproduced to quasi-identity in AeroDyn. Linearization is not possible with this model.

The model as presented in [ad-MB11] is an output-only model, where the dynamic angle of attack is determined using the quasi steady angle of attack and the rate of change of the angle of attack:

$$\alpha_{dyn} = \alpha_{34} - k_1 \gamma \sqrt{|\dot{\alpha} T_u|}$$

where k_1 and γ are constants of the model. In practice, the implementation is different for the lift and drag coefficients, and for negative and positive stall. The model needs a discrete state to calculate the rate of change of the angle of attack and two discrete states to keep track of whether the model is activated or not.

Airfoil constants:

The constants k_1 , for positive and negative rates of angle of attack, are set to:

$$k_{1,p} = 1, \quad k_{1,n} = 1/2$$

The extent of the transition region is computed as:

$$\Delta\alpha_{\max} = \frac{0.9 \min(|\alpha_1 - \alpha_0|, |\alpha_2 - \alpha_0|)}{\max(k_{1,p}, k_{1,n})}$$

where α_1 and α_2 are the angle of attack at positive and negative stall respectively (taken as the values from the airfoil input file). The factor 0.9 is a margin to prevent the effective angle of attack to reach α_0 during stall.

Intermediate variables:

The variables γ for the lift and drag are computed as function of the thickness to chord ratio of the airfoil t_c and the Mach number Ma (assumed to be 0 in the current implementation):

$$\begin{aligned}\gamma_L &= (1.4 - 6\delta) \left[1 - \frac{Ma - (0.4 + 5\delta)}{0.9 + 2.5\delta - (0.4 + 5\delta)} \right] \\ \gamma_D &= (1 - 2.5\delta), & \text{if } Ma < 0.2 \\ \gamma_D &= (1 - 2.5\delta) \left[1 - \frac{Ma - 0.2}{(0.7 + 2.5\delta - 0.2)} \right], & \text{otherwise}\end{aligned}$$

where $\delta = 0.06 - t_c$.

Update of discrete states (and intermediate variables):

The rate of change of the angle of attack is computed as:

$$\dot{\alpha} = \frac{\alpha_{34}(t + \Delta t) - \alpha_{34}(t)}{\Delta t}$$

An additional state was introduced to avoid sudden jump of $\dot{\alpha}$, by storing its value. Rates that are beyond a fraction of $\pi \Delta t$ are replaced with the values at the previous time step. This feature is not present in the CACTUS implementation.

The dynamic angle of attack offsets (lags) for the lift and drag are computed as:

$$\begin{aligned}\Delta\alpha_L &= k_1 \min \left(\gamma_L \sqrt{|\dot{\alpha} T_u|}, \Delta\alpha_{\max} \right) \\ \Delta\alpha_D &= k_1 \min \left(\gamma_D \sqrt{|\dot{\alpha} T_u|}, \Delta\alpha_{\max} \right)\end{aligned}$$

The value of k_1 is taken as $k_{1,n}$ if $\dot{\alpha}(\alpha_{34} - \alpha_0) < 0$, and taken as $k_{1,p}$ otherwise. The lagged angle of attacks for the lift and drag are:

$$\begin{aligned}\alpha_{\text{Lag},L} &= \alpha_{34} - \Delta\alpha_L \text{sign}(\dot{\alpha}) \\ \alpha_{\text{Lag},D} &= \alpha_{34} - \Delta\alpha_D \text{sign}(\dot{\alpha})\end{aligned}$$

The distances to positive and negative stall are computed as follows. If $\dot{\alpha}(\alpha_{34} - \alpha_0) < 0$ and the dynamic stall is active:

$$\Delta_n = \alpha_2 - \alpha_{\text{Lag},D}, \quad \Delta_p = \alpha_{\text{Lag},D} - \alpha_1$$

If $\dot{\alpha}(\alpha_{34} - \alpha_0) < 0$ and the dynamic stall is not active:

$$\Delta_n = 0, \quad \Delta_p = 0$$

If $\dot{\alpha}(\alpha_{34} - \alpha_0) \geq 0$:

$$\Delta_n = \alpha_2 - \alpha_{34}, \quad \Delta_p = \alpha_{34} - \alpha_1$$

The effective angle of attack for the lift coefficient is taken as the lagged angle of attack:

$$\alpha_{e,L} = \alpha_{\text{Lag},L}$$

The effective angle of attack for the drag coefficient is obtained from the lagged angle of attack and the deltas to stall:

$$\begin{aligned}\alpha_{e,D} &= \alpha_{\text{Lag},D}, & \text{if } \Delta_n > T \text{ or } \Delta_p > T \\ \alpha_{e,D} &= \alpha_{34} + (\alpha_{\text{Lag},D} - \alpha_{34}) \frac{\Delta_n}{T}, & \text{if } \Delta_n > 0 \text{ and } \Delta_n < T \\ \alpha_{e,D} &= \alpha_{34} + (\alpha_{\text{Lag},D} - \alpha_{34}) \frac{\Delta_p}{T}, & \text{if } \Delta_p > 0 \text{ and } \Delta_p < T \\ \alpha_{e,D} &= \alpha_{34}, & \text{otherwise}\end{aligned}$$

where $T = 2\Delta\alpha_{\max}$ is the extent of the “transition” region.

The lift dynamic stall state is activated if $\dot{\alpha}(\alpha_{34} - \alpha_0) \geq 0$ and if the angle of attack is above α_1 or below α_2 . The state is turned off if $\dot{\alpha}(\alpha_{34} - \alpha_0) < 0$ and the effective angle of attack is below α_1 and above α_2 .

The drag dynamic stall state is activated if any of the condition below are true:

$$\Delta_n > T \text{ or } \Delta_p > T$$

$$\Delta_n > 0 \text{ and } \Delta_n < T$$

$$\Delta_p > 0 \text{ and } \Delta_p < T$$

The state is turned off otherwise.

Calculation of outputs: The calculation of the dynamic lift and drag coefficients is done as follows

$$\begin{aligned} C_{l,\text{dyn}} &= \frac{C_l^{\text{st}}(\alpha_{e,L})}{\alpha_{e,L} - \alpha_0}, & \text{if dynamic stall active for } C_l \\ C_{l,\text{dyn}} &= C_l^{\text{st}}(\alpha_{34}), & \text{otherwise} \\ C_{d,\text{dyn}} &= C_d^{\text{st}}(\alpha_{e,D}) \end{aligned}$$

Recalculation of intermediate variables are necessary to obtain $\alpha_{e,L}$ and $\alpha_{e,D}$. The moment coefficient is calculated based on values at the aerodynamic center and mid-chord (“50”):

$$C_{m,\text{dyn}} = C_m^{\text{st}}(\alpha_{ac}) + \cos \alpha_{50} [C_l^{\text{st}}(\alpha_{34}) - C_l^{\text{st}}(\alpha_{50})] / 4$$

where α_{50} is computed the same way as α_{34} (using the velocity at the aerodynamic center and the rotational rate of the airfoil) but using the distance from the aerodynamic center to the mid-chord (see [Section 4.2.1](#)).

Inputs

See [Section 4.2.1](#) for a description of the inputs necessary in the AeroDyn primary file (e.g. UAMod)

See [Section 4.2.1](#) for a more comprehensive description of all the inputs present in the profile input file. Their default values are described in [Section 4.2.1](#)

See [Section 4.2.1](#) for a list of notations and definitions specific to unsteady aerodynamic inputs.

An example of profile data (containing some of the unsteady aerodynamic parameters) is available [here](#) (here).

Calculating Default Airfoil Coefficients

The default value for `cd0` is the minimum value of the C_d curve between ± 20 degrees angle of attack. α_{cd0} is defined to be the angle of attack where `cd0` occurs.

After computing `cd0`, the C_n curve is computed by

$$C_n(\alpha) = C_l(\alpha) \cos \alpha + (C_d(\alpha) - c_{d0}) \sin \alpha$$

The slope of the C_n curve is computed as follows:

$$C_n^{\text{Slope}} \left(\frac{\alpha_{i+1} + \alpha_i}{2} \right) = \frac{C_n(\alpha_{i+1}) - C_n(\alpha_i)}{\alpha_{i+1} - \alpha_i}$$

$C_{n,\text{smooth}}^{\text{Slope}}$ is a smoothed version of C_n^{Slope} , calculated using a triweight kernel with a window of 2 degrees.

$$C_l^{\text{Slope}} \left(\frac{\alpha_{i+1} + \alpha_i}{2} \right) = \frac{C_l(\alpha_{i+1}) - C_l(\alpha_i)}{\alpha_{i+1} - \alpha_i}$$

Using $C_{n,smooth}^{Slope}$, **alphaUpper** and **alphaLower** are computed:

alphaUpper is the smallest angle of attack value between α_{cd0} and 20 degrees where the $C_{n,smooth}^{Slope}$ curve has started to decrease to 90% of its maximum slope.

$$C_{n,smooth}^{Slope}(\alpha^{Upper}) < 0.9 \max_{\alpha \in [\alpha_{cd0}, \alpha^{Upper}]} C_{n,smooth}^{Slope}(\alpha)$$

alphaLower is the largest angle of attack value between -20 degrees and α_{cd0} where the $C_{n,smooth}^{Slope}$ curve has started to decrease to 90% of its maximum slope.

$$C_{n,smooth}^{Slope}(\alpha^{Lower}) < 0.9 \max_{\alpha \in [\alpha^{Lower}, \alpha_{cd0}]} C_{n,smooth}^{Slope}(\alpha)$$

Cn1 is the value of $C_n(\alpha)$ at the smallest value of α where $\alpha \geq \alpha^{Upper}$ and the separation function, $f_{st}(\alpha) = 0.7$.

Cn2 is the value of $C_n(\alpha)$ at the largest value of α where $\alpha \leq \alpha^{Lower}$ and the separation function, $f_{st}(\alpha) = 0.7$.

Cn_offset is the average value of the C_n curve at **alphaUpper** and **alphaLower**:

$$C_n^{offset} = \frac{C_n(\alpha^{Lower}) + C_n(\alpha^{Upper})}{2}$$

C_nalpha is defined as the maximum slope of the smoothed C_n curve, $C_{n,smooth}^{Slope}$ between ± 20 degrees angle of attack.

C_lalpha is defined as the maximum slope of the (un-smoothed) C_l curve, C_l^{Slope} between ± 20 degrees angle of attack.

The default **alpha0** is computed as the zero-crossing of a line with a slope equal to **C_lalpha** that goes through the C_l curve at $\alpha = \frac{\alpha^{Upper} + \alpha^{Lower}}{2}$

$$\alpha_0 = \frac{\alpha^{Upper} + \alpha^{Lower}}{2} - \frac{C_l\left(\frac{\alpha^{Upper} + \alpha^{Lower}}{2}\right)}{C_{l,\alpha}}$$

Cm0 is the value of the C_m curve at **alpha0**: $C_{m,0} = C_m(\alpha_0)$. If the C_m polar values have not been included, $C_{m,0} = 0$.

alpha1 is the angle of attack above **alphaUpper** where the separation function, f_s^{st} is 0.7.

alpha2 is the angle of attack below **alphaLower** where the separation function, f_s^{st} is 0.7.

Cn1 is the value of the C_n curve at **alpha1**.

Cn2 is the value of the C_n curve at **alpha2**.

Outputs

Outputting variables of the dynamic stall models is possible, but requires to set preprocessor variable **UA_OUTS** and recompile the program (OpenFAST, AeroDyn Driver, or Unsteady Aero driver). The outputs are written in output files with extension *.UA.out. To activate these outputs with *cmake*, compile using **-DCMAKE_Fortran_FLAGS="-DUA_OUTS=ON"**

Driver

A driver is available to run simulations for a single airfoil, using sinusoidal variation of the angle of attack, or user defined time series of angle of attack, relative wind speed and pitch rate.

Using *cmake*, the driver is compiled using *make unsteadyaero_driver*, resulting as an executable in the *aerodyn* folder.

An example of driver input file is available here: [here](#). An example of time series input is available here: [here](#)

Appendix

AeroDyn Input Files

In this appendix we describe the AeroDyn input-file structure and provide examples.

1) Baseline AeroDyn Driver Input File: (`driver input file example`): The driver input file is only needed for the standalone version of AeroDyn and contains inputs normally generated by OpenFAST, and necessary to control the aerodynamic simulation for uncoupled models.

AeroDyn Driver Timeseries Input File (`driver timeseries input file example`): The timeseries input file for a case in the AeroDyn driver allows the parameters to vary with time. This feature can be useful for debugging the aerodynamic response outside of OpenFAST.

2) Multi-rotor AeroDyn Driver Input File (`driver input file example`):

3) AeroDyn Primary Input File (`primary input file example`):

The primary AeroDyn input file defines modeling options, environmental conditions (except freestream flow), airfoils, tower nodal discretization and properties, as well as output file specifications.

The file is organized into several functional sections. Each section corresponds to an aspect of the aerodynamics model.

The input file begins with two lines of header information which is for your use, but is not used by the software.

4) Airfoil Data Input File

(`profile data`):

(`profile coordinates`):

The airfoil data input files themselves (one for each airfoil) include tables containing coefficients of lift force, drag force, and pitching moment versus AoA, as well as UA model parameters. In these files, any line whose first non-blank character is an exclamation point (!) is ignored (for inserting comment lines). The non-comment lines should appear within the file in order, but comment lines may be intermixed as desired for reading clarity.

5) Blade Data Input File (`blade data input file example`):

The blade data input file contains the nodal discretization, geometry, twist, chord, and airfoil identifier for a blade. Separate files are used for each blade, which permits modeling of aerodynamic imbalances.

AeroDyn List of Output Channels

This is a list of all possible output parameters for the AeroDyn module. The names are grouped by meaning, but can be ordered in the OUTPUTS section of the AeroDyn input file as you see fit. $B\alpha N\beta$, refers to output node β of blade α , where α is a number in the range [1,3] and β is a number in the range [1,9], corresponding to entry β in the *BlOutNd* list. $TwN\beta$ refers to output node β of the tower and is in the range [1,9], corresponding to entry β in the *TwOutNd* list.

The local tower coordinate system is shown in Fig. 4.1 and the local blade coordinate system is shown in Fig. 4.6 below. Figure Fig. 4.6 also shows the direction of the local angles and force components.

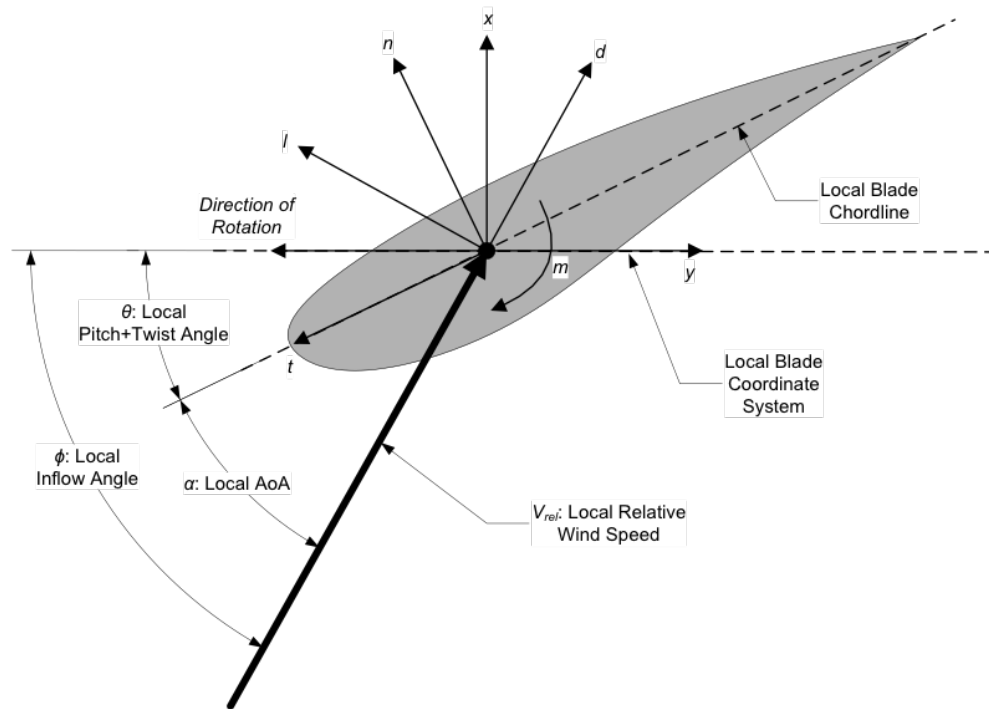


Fig. 4.6: AeroDyn Local Blade Coordinate System (Looking Toward the Tip, from the Root) – l: Lift, d: Drag, m: Pitching, x: Normal (to Plane), y: Tangential (to Plane), n: Normal (to Chord), and t: Tangential (to Chord)

4.2.2 OLAF User's Guide and Theory Manual (Free Vortex Wake in AeroDyn15)

Introduction

Over the past few decades, substantial reductions in the cost of wind energy have come from large increases in rotor size. One important consideration for such large turbines is increased blade flexibility. In particular, large blade deflections may lead to a swept area that deviates significantly from the rotor plane. Such deviations violate assumptions used by common aerodynamic models, such as the blade element momentum (BEM) method. Such methods rely on actuator-disk assumptions that are only valid for axisymmetric rotor loads contained in a plane. Large blade deflections may also cause near wake of the turbine to diverge from a uniform helical shape. Further, interactions between turbine blades and the local near wake may increase, thus violating assumptions of models that do not account for the position and dynamics of the near wake. Additionally, highly flexible blades will likely cause increased unsteadiness and three-dimensionality of aerodynamic effects, increasing the importance of accurate and robust dynamic stall models. There are many other complex wind turbine situations that violate simple engineering assumptions. Such situations include obtaining accurate aerodynamic loads for nonstraight blade geometries (e.g., built-in curvature or sweep); skewed flow caused by yawed inflow or turbine tilt; and large rotor motion as a result of placing the turbine atop a compliant offshore floating platform.

Channel Name(s)	Units	Description
<i>Tower</i>		
TwNβVUndx, TwNβVUndy, TwNβVUndz	(m/s), (m/s), (m/s)	Undisturbed wind velocity at TwNβ in the local tower coordinate system
TwNβSTVx, TwNβSTVy, TwNβSTVz	(m/s), (m/s), (m/s)	Structural translational velocity at TwNβ in the local tower coordinate system
TwNβVrel	(m/s)	Relative wind speed at TwNβ
TwNβDynP	(Pa)	Dynamic pressure at TwNβ
TwNβRe	(-)	Reynolds number (in millions) at TwNβ
TwNβM	(-)	Mach number at TwNβ
TwNβFdx, TwNβFdy	(N/m), (N/m)	Drag force per unit length at TwNβ in the local tower coordinate system
<i>Blade</i>		
BaAzimuth	(deg)	Azimuth angle of Ba
BaPitch	(deg)	Pitch angle of Ba
BaNβClrc ¹	(m)	Tower clearance at BaNβ ¹
BaNβVUndx, BaNβVUndy, BaNβVUndz	(m/s), (m/s), (m/s)	Undisturbed wind velocity at BaNβ in the local blade coordinate system
BaNβVDisx, BaNβVDisy, BaNβVDisz	(m/s), (m/s), (m/s)	Disturbed wind velocity at BaNβ in the local blade coordinate system
BaNβSTVx, BaNβSTVy, BaNβSTVz	(m/s), (m/s), (m/s)	Structural translational velocity at BaNβ in the local blade coordinate system
BaNβVrel	(m/s)	Relative wind speed at BaNβ
BaNβDynP	(Pa)	Dynamic pressure at BaNβ
BaNβRe	(-)	Reynolds number (in millions) at BaNβ
BaNβM	(-)	Mach number at BaNβ
BaNβVIndx, BaNβVIndy	(m/s), (m/s)	Axial and tangential induced wind velocity at BaNβ
BaNβAxInd, BaNβTnInd	(-), (-)	Axial and tangential induction factors at BaNβ
BaNβAlpha, BaNβTheta, BaNβPhi, BaNβCurve	(deg), (deg), (deg), (deg)	AoA, pitch+twist angle, inflow angle, and curvature angle at BaNβ
BaNβCl, BaNβCd, BaNβCm, BaNβCpmin, BaNβCx, BaNβCy ² , BaNβCn, BaNβCt	(-), (-), (-), (-) (-), (-), (-), (-)	Lift force, drag force, pitching moment, minimum pressure, normal force (to plane), tangential force (to plane) ² , normal force (to

¹ BaNβClrc is based on the absolute distance to the nearest point in the tower from BaNβ minus the local tower radius, in the deflected configuration. Please note that this clearance is only approximate because the calculation assumes that the blade is a line with no volume (however, the calculation does use the local tower radius). When BaNβ is above the tower top (or below the tower base), the absolute distance to the tower top (or base) minus the local tower radius, in the deflected configuration, is output.

Fig. 4.7: AeroDyn Output Channel List

Higher-fidelity aerodynamic models are necessary to account for the increased complexity of flexible and floating rotors. Although computational fluid dynamics (CFD) methods are able to capture such features, their computational cost limits the number of simulations that can be feasibly performed, which is an important consideration in load analysis for turbine design. FVW methods are less computationally expensive than CFD methods while modeling similarly complex physics. As opposed to the BEM methods, FVW methods do not rely on ad-hoc engineering models to account for dynamic inflow, skewed wake, tip losses, or ground effects. These effects are inherently part of the model. Numerous vorticity-based tools have been implemented, ranging from the early treatments by Rosenhead ([olaf-Ros31]), the formulation of vortex particle methods by Winckelmans and Leonard ([olaf-WL93]), to the recent mixed Eulerian-Lagrangian compressible formulations of Papadakis ([olaf-Pap14]). Examples of long-standing codes that have been applied in the field of wind energy are GENUVP ([olaf-Vou06]), using vortex particles methods, and AWSM ([olaf-vG03]), using vortex filament methods. Both tools have successfully been coupled to structural solvers. The method was extended by Branlard et al. ([olaf-BPG+15]) to consistently use vortex methods to perform aero-elastic simulations of wind turbines in sheared and turbulent inflow. Most formulations rely on a lifting-line representation of the blades, but recently, a viscous-inviscid representation was used in combination with a structural solver ([olaf-SGarciaSorensenS17]).

cOnvecting LAGRangian Filaments (OLAF) is a free vortex wake (FVW) module used to compute the aerodynamic forces on moving two- or three-bladed horizontal-axis wind turbines. This module has been incorporated into the National Renewable Energy Laboratory physics-based engineering tool, OpenFAST, which solves the aero-hydro-servo-elastic dynamics of individual wind turbines. OLAF is incorporated into the OpenFAST module, *AeroDyn15*, as an alternative to the traditional blade-element momentum (BEM) option, as shown in Figures Fig. 4.8 and Fig. 4.9.

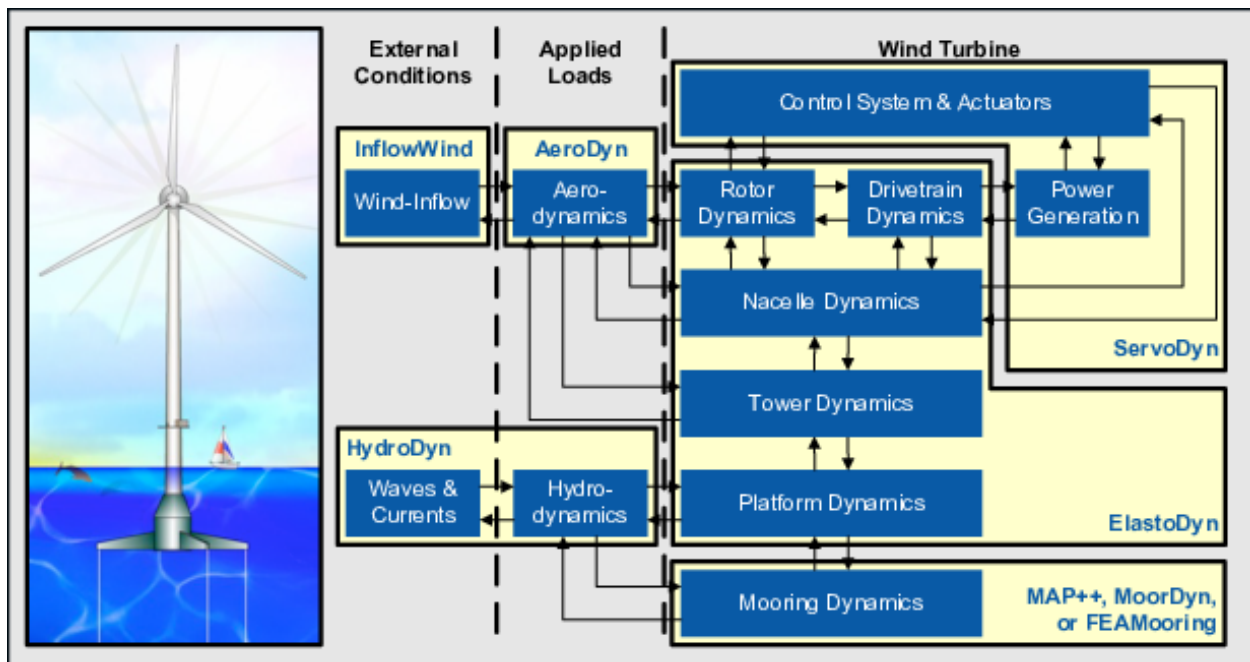


Fig. 4.8: OpenFAST schematic

Incorporating the OLAF module within OpenFAST allows for the modeling of highly flexible turbines along with the aero-hydro-servo-elastic response capabilities of OpenFAST. The OLAF module follows the requirements of the OpenFAST modularization framework ([olaf-Jon13, olaf-SJJ15]).

The OLAF module uses a lifting-line representation of the blades, which is characterized by a distribution of bound circulation. The spatial and time variation of the bound circulation results in free vorticity being emitted in the wake. OLAF solves for the turbine wake in a time-accurate manner, which allows the vortices to convect, stretch, and diffuse. The OLAF model is based on a Lagrangian approach, in which the turbine wake is discretized into Lagrangian markers. There are many methods of representing the wake with Lagrangian markers ([olaf-Bra17]). In this work, a hybrid lattice/filament method is used, as depicted in Figure Fig. 4.10.

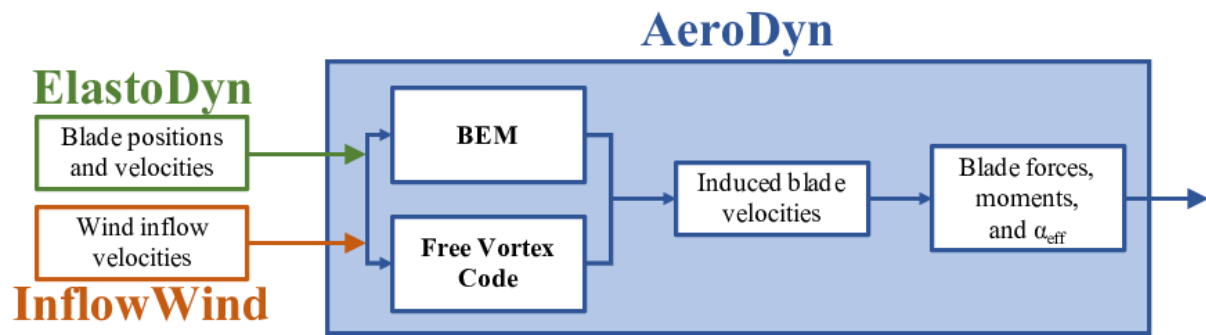
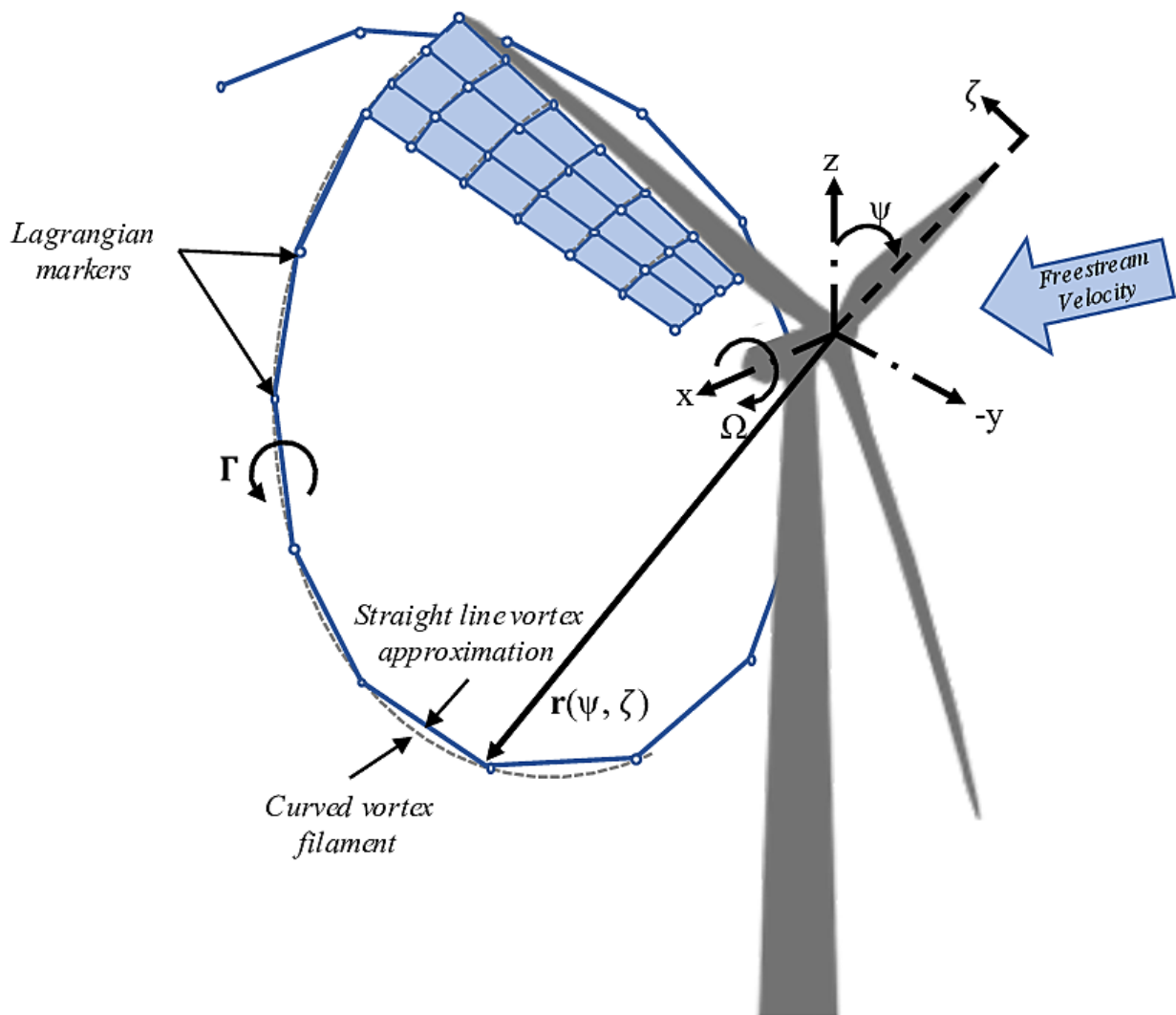
Fig. 4.9: OLAF and BEM integration with *AeroDyn15*

Fig. 4.10: Evolution of near-wake lattice, blade-tip vortex, and Lagrangian markers

Here, the position of the Lagrangian markers is defined in terms of wake age, ζ , and azimuthal position, ψ . A lattice method is used in the near wake of the blade. The near wake spans over a user-specified angle or distance for nonrotating cases. Though past research has indicated that a near-wake region of 30° is sufficient ([olaf-ALR02, olaf-Lei06]), it has been shown that a larger near wake is required for high thrust and other challenging conditions. After the near wake region, the wake is assumed to instantaneously roll up into a tip vortex and a root vortex, which are assumed to be the most dominant features for the remainder of the wake ([olaf-LBB02]). Each Lagrangian marker is connected to adjacent markers by straight-line vortex filaments, approximated to second-order accuracy ([olaf-GL02]). The wake is discretized based on the spanwise location of the blade sections and a specified time step (dt), which may be different from the time step of AeroDyn. After an optional initialization period, the wake is allowed to move and distort, thus changing the wake structure as the markers are convected downstream. To limit computational expense, the root and tip vortices are truncated after a specified distance (**WakeLength**) downstream from the turbine. The wake truncation violates Helmholtz's first law and hence introduces an erroneous boundary condition. To alleviate this, the wake is "frozen" in a buffer zone between a specified buffer distance, **FreeWakeLength**, and **WakeLength**. In this buffer zone, the markers convect at the average ambient velocity. In this way, truncation error is minimized~([olaf-LBB02]). The buffer zone is typically chosen as the convected distance over one rotor revolution.

As part of OpenFAST, induced velocities at the lifting line/blade are transferred to *AeroDyn15* and used to compute the effective blade angle of attack at each blade section, which is then used to compute the aerodynamic forces on the blades. The OLAF method returns the same information as the BEM method, but allows for more accurate calculations in areas where BEM assumptions are violated, such as those discussed above. As the OLAF method is more computationally expensive than BEM, both methods remain available in OpenFAST, and the user may specify in the *AeroDyn15* input file which method is used.

The OLAF input file defines the wake convection and circulation solution methods; wake size and length options; Lagrangian marker regularization (viscous core) method; and other simulation and output parameters. The extents of the near and far wakes are specified by a nondimensional length in terms of rotor diameter. Different regularization functions for the vortex elements are available. Additionally, different methods to compute the regularization parameters of the bound and wake vorticity may be selected. In particular, viscous diffusion may be accounted for by dynamically changing the regularization parameter. Wake visualization output options are also available.

This document is organized as follows. [Section 4.2.2](#) covers downloading, compiling, and running OLAF. [Section 4.2.2](#) describes the OLAF input file and modifications to the *AeroDyn15* input file. [Section 4.2.2](#) details the OLAF output file. [Section 4.2.2](#) provides an overview of the OLAF theory, including the free vortex wake method as well as integration into the *AeroDyn15* module. Example input files and a list of output channels are detailed in Appendices A, B, and C.

List of Symbols

BEM	blade-element momentum
CFD	computational fluid dynamics
DOE	U.S. Department of Energy
F_v	core radius factor
t	time
FVW	free vortex wake
N	number of rotor revolutions before wake cutoff condition
\vec{r}	vector between point of interest and vortex segment
$\vec{r}(\psi, \zeta)$	position vector of Lagrangian markers
r_c	core radius
r_{c0}	initial core radius
OLAF	cOnvecting LAgrangian Filaments
α	numerical constant = 1.25643
Γ	circulation strength
δ	measure of viscous diffusion
ϵ	measure of strain
$\Delta\psi$	step size for blade rotation
Ω	rotational speed of wind turbine
ζ	vortex wake age
ζ_0	vortex wake age offset
ν	kinematic viscosity
ψ	azimuth blade position

Working with OLAF

Running OLAF

As OLAF is a module of OpenFAST, the process of downloading, compiling, and running OLAF is the same as that for OpenFAST. Such instructions are available in the [Installing OpenFAST](#) documentation.

Note: To improve the speed of FVW module, the user may wish to compile with *OpenMP*. To do so, add the `-DOPENMP=ON` option with CMake.

Guidelines

Most options of OLAF can be left to default. The results will depend on the time discretization, wake length, and regularization parameters. We provide guidelines for these parameters in this section, together with a simple python code to compute these parameters. Please check this section again as we might further refine our guidelines with time.

Time step and wake length We recommend to set OLAF's time step (**DTfvw**) such that it corresponds to $\Delta\psi = 6$ degrees of one rotor revolution:

$$\Delta t = \frac{\Delta\psi_{\text{rad}}}{\Omega_{\text{rad/s}}} = \frac{\Delta\psi_{\text{deg}}}{6 \times \Omega_{\text{RPM}}}$$

If the structural solver requires a smaller time step, the time step for the glue code can be set to a different value than **DTfvw** as long as **DTfvw** is a multiple of the glue code time step.

We recommend to set the near wake length to the number of time steps needed to reach two rotor revolutions. For the far wake, we recommend 10 rotor revolutions. For the free far-wake, we recommend to set the distance to a value somewhere between 25% and 50% of the far wake length, (e.g. 3 revolutions).

The following python script computes the parameters according to these guidelines.

```
def OLAParams(omega_rpm, deltaPsiDeg=6, nNWrot=2, nFWrot=10, nFWrotFree=3,
nPerAzimuth=None):
    """
    Computes recommended time step and wake length based on the rotational speed in RPM

    INPUTS:
    - omega_rpm: rotational speed in RPM
    - deltaPsiDeg : azimuthal discretization in deg
    - nNWrot : number of near wake rotations
    - nFWrot : total number of far wake rotations
    - nFWrotFree : number of far wake rotations that are free

        deltaPsiDeg - nPerAzimuth
            5         72
            6         60
            7         51.5
            8         45
    """
    omega = omega_rpm*2*np.pi/60
    T = 2*np.pi/omega
    if nPerAzimuth is not None:
        dt_wanted = np.around(T/nPerAzimuth,3)
    else:
        dt_wanted = np.around(deltaPsiDeg/(6*omega_rpm),3)
        nPerAzimuth = int(2*np.pi / (deltaPsiDeg*np.pi/180))

    nNWPanel = nNWrot*nPerAzimuth
    nFWPanel = nFWrot*nPerAzimuth
    nFWPanelFree = nFWrotFree*nPerAzimuth

    print(dt_wanted, ' DTfvw')
    print(int(nNWPanel), ' nNWPanel ')
    print(int(nFWPanel), ' WakeLength ')
    print(int(nFWPanelFree), ' FreeWakeLength')

    return dt_wanted, nNWPanel, nFWPanel, nFWPanelFree
```

Regularization parameters

One critical parameter of vortex methods is the regularization parameter, also referred to as core radius. We currently recommend to set the regularization parameter as a fraction of the spanwise discretization, that is: **RegDetMethod=3**, **WakeRegFactor=0.6**, **WingRegFactor=0.6**. We will likely update these guidelines in the future.

We also recommend to have the regularization increasing with downstream distance: **WakeRegMethod=3**.

The factor with which the regularization parameter will increase with downstream distance can be set as **Core-SpreadEddyVisc=1000** for modern multi-MW turbines. Further guidelines will follow for this parameter in the future.

Input Files

No lines should be added or removed from the input files, except in tables where the number of rows is specified.

Units

OLAF uses the International System of Units (e.g., kg, m, s, N). Angles are assumed to be in degrees unless otherwise specified.

OLAF Primary Input File

The primary OLAF input file defines general free wake options, circulation model selection and specification, near- and far-wake length, and wake visualization options. Each section within the file corresponds to an aspect of the OLAF model. For most parameters, the user may specify the value “default” (with or without quotes), in which case a default value, defined below, is used by the program.

See [Section 4.2.2](#) for a sample OLAF primary input file.

General Options

IntMethod [switch] specifies which integration method will be used to convect the Lagrangian markers. There are four options: 1) fourth-order Runge-Kutta [1], 2) fourth-order Adams-Bashforth [2], 3) fourth-order Adams-Bashforth-Moulton [3], and 4) first-order forward Euler [5]. The default option is [5]. Currently only options [1] and [5] are implemented. These methods are specified in [Section 4.2.2](#).

DTfww [sec] specifies the time interval at which the module will update the wake. The time interval must be a multiple of the time step used by *AeroDyn15*. The blade circulation is updated at each intermediate time step based on the intermediate blades positions and wind velocities. The default value is dt_{aero} , where dt_{aero} is the time step used by *AeroDyn*.

FreeWakeStart [sec] specifies at what time the wake evolution is classified as “free.” Before this point is reached, the Lagrangian markers are simply convected with the freestream velocity. After this point, induced velocities are computed and affect the marker convection. If a time less than or equal to zero is given, the wake is “free” from the beginning of the simulation. The default value is 0.

FullCircStart [sec] specifies at what time the blade circulation reaches its full strength. If this value is specified to be > 0 , the circulation is multiplied by a factor of 0 at $t = 0$ and linearly increasing to a factor of 1 for $t > FullCircStart$. The default value is 0.

Circulation Specifications

CircSolvMethod [switch] specifies which circulation method is used. There are three options: 1) C_l -based iterative procedure [1], 2) no-flow through [2], and 3) prescribed [3]. The default option is [1]. These methods are described in [Section 4.2.2](#).

CircSolvConvCrit [-] specifies the dimensionless convergence criteria used for solving the circulation. This variable is only used if *CircSolvMethod* = [1]. The default value is 0.001, corresponding to 0.1% error in the circulation between two iterations.

CircSolvRelaxation [-] specifies the relaxation factor used to solve the circulation. This variable is only used if *CircSolvMethod* = [1]. The default value is 0.1.

CircSolvMaxIter [-] specifies the maximum number of iterations used to solve the circulation. This variable is only used if *CircSolvMethod* = [1]. The default value is 30.

PrescribedCircFile [quoted string] specifies the file containing the prescribed blade circulation. This option is only used if *CircSolvMethod* = [3]. The circulation file format is a delimited file with one header line and two columns. The first column is the dimensionless radial position [r/R]; the second column is the bound circulation value in [m²/s]. The radial positions do not need to match the AeroDyn node locations. A sample prescribed circulation file is given in [Section 4.2.2](#).

Wake Extent and Discretization Options

nNWPanel [-] specifies the number of FVW time steps (**DTfww**) for which the near-wake lattice is computed. In the future, this value will be defined as an azimuthal span in degrees or a downstream distance in rotor diameter.

WakeLength [D] specifies the length, in rotor diameters, of the far wake. The default value is 8.¹

FreeWakeLength [D] specifies the length, in rotor diameters, for which the turbine wake is convected as “free.” If *FreeWakeLength* is greater than *WakeLength*, then the entire wake is free. Otherwise, the Lagrangian markers located within the buffer zone delimited by *FreeWakeLength* and *WakeLength* are convected with the average velocity. The default value is 6.²

FWSshedVorticity [flag] specifies whether shed vorticity is included in the far wake. The default value is [False], specifying that the far wake consists only of the trailed vorticity from the root and tip vortices.

Wake Regularization and Diffusion Options

DiffusionMethod [switch] specifies which diffusion method is used to account for viscous diffusion. There are two options: 1) no diffusion [0] and 2) the core-spreading method [1]. The default option is [0].

RegDetMethod [switch] specifies which method is used to determine the regularization parameters. There are four options: 1) constant [0] and 2) optimized [1], 3) chord [2], and 4) span [3]. The optimized option determines all the parameters in this section for the user. The optimized option is still work in progress and not recommended. The constant option requires the user to specify all the parameters present in this section. The default option is [0]. When **RegDetMethod**==0, the regularization parameters is set constant:

$$r_{c,wake}(r) = \text{WakeRegParam}, \quad r_{c,blade}(r) = \text{BladeRegParam}$$

When **RegDetMethod**==2, the regularization parameters is set according to the local chord:

$$r_{c,wake}(r) = \text{WakeRegParam} \cdot c(r), \quad r_{c,blade}(r) = \text{BladeRegParam} \cdot c(r)$$

When **RegDetMethod**==3, the regularization parameters is set according to the spanwise discretization:

$$r_{c,wake}(r) = \text{WakeRegParam} \cdot \Delta r(r), \quad r_{c,blade}(r) = \text{BladeRegParam} \cdot \Delta r(r)$$

where *Deltar* is the length of the spanwise station.

RegFunction [switch] specifies the regularization function used to remove the singularity of the vortex elements, as specified in [Section 4.2.2](#). There are five options: 1) no correction [0], 2) the Rankine method [1], 3) the Lamb-Oseen method [2], 4) the Vatisstas method [3], and 5) the denominator offset method [4]. The functions are given in . The default option is [3].

WakeRegMethod [switch] specifies the method of determining viscous core radius (i.e., the regularization parameter). There are three options: 1) constant [1], 2) stretching [2], and 3) age [3]. The methods are described in [Section 4.2.2](#). The default option is [1].

¹ At present, this variable is called nFWPanel and specified as the number of far wake panels. This will be changed soon.

² At present, this variable is called nFWPanelFree and specified as the number of free far wake panels. This will be changed soon.

WakeRegParam [m, or -] specifies the wake regularization parameter, which is the regularization value used at the initialization of a vortex element. If the regularization method is “constant”, this value is used throughout the wake.

BladeRegParam [m, or -] specifies the bound vorticity regularization parameter, which is the regularization value used for the vorticity elements bound to the blades.

CoreSpreadEddyVisc [-] specifies the eddy viscosity parameter δ . The parameter is used for the core-spreading method (*DiffusionMethod* = [1]) and the regularization method with age (*WakeRegMethod* = [3]). The variable δ is described in [Section 4.2.2](#). The default value is 100.

Wake Treatment Options

TwrShadowOnWake [flag] specifies whether the tower potential flow and tower shadow have an influence on the wake convection. The tower shadow model, when activated in AeroDyn, always has an influence on the lifting line, hence the induction and loads on the blade. This option only concerns the wake. The default option is [False].

ShearVorticityModel [switch] specifies whether shear vorticity is modeled in addition to the sheared inflow prescribed by *InflowWind*. There are two options: 1) no treatment [0] and 2) mirrored vorticity [1]. The mirrored vorticity accounts for the ground effect. Dedicated options to account for the shear vorticity will be implemented at a later time. The shear velocity profile is handled by *InflowWind* irrespective of this input. The default option is [0].

Speedup Options

VelocityMethod [switch] specifies the method used to determine the velocity. There are four options: 1) N^2 Biot-Savart computation on the vortex segments [1], 2) Particle-Tree formulation [2], 3) N^2 Biot-Savart computation using a particle representation, 4) Segment-Tree formulation. The default option is [1]. Option [2] and [3] requires the specification of *PartPerSegment* (see below). Option [4] is expected to give results close to option [1] while offering significant speedup, and this option does not require the specification of *PartPerSegment*.

TreeBranchFactor [-] specifies the dimensionless distance, in branch radius, above which a multipole calculation is used instead of a direct evaluation. This option is only used in conjunction with the tree code (*VelocityMethod* = [2]).

PartPerSegment [-] specifies the number of particles that are used when a vortex segment is represented by vortex particles. The default value is 1.

Output Options

WrVTK [flag] specifies if Visualization Toolkit (VTK) visualization files are to be written out. *WrVTK* = [0] does not write out any VTK files. *WrVTK* = [1] outputs VTK files at time steps defined by *VTK_fps*. *WrVTK* = [2], outputs at time steps defined by *VTK_fps*, but ensures that a file is written at the beginning and the end of the simulation (typically used with *VTK_fps*=0 to output only at the end of the simulation). The outputs are written in the folder, *vtk_fwv*. The parameters *WrVTK*, *VTKCoord*, and *VTK_fps* are independent of the glue code VTK output options.

VTKBlades [-] specifies how many blade VTK files are to be written out. *VTKBlades* = *n* outputs VTK files for *n* blades, with 0 being an acceptable value. The default value is 1.

VTKCoord [switch] specifies in which coordinate system the VTK files are written. There are two options: 1) global coordinate system [1] and 2) hub coordinate system [2]. The default option is [1].

VTK_fps [1/sec] specifies the output frequency of the VTK files. The provided value is rounded to the nearest allowable multiple of the time step. The default value is $1/dt_{fvw}$. Specifying *VTK_fps* = [all], is equivalent to using the value $1/dt_{aero}$. If *VTK_fps* < 0, then no outputs are created, except if *WrVTK*=2.

nGridOut [-] specifies the number of grid outputs. The default value is 0. The grid outputs are fields (velocity, vorticity) that are exported on a regular Cartesian grid. They are defined using a table that follows on the subsequent

lines, with two lines of headers. The user needs to specify a name (**GridName**) used for the VTK output filename, a grid type (**GridType**), a start time (**TStart**), an end time (**TEnd**), a time interval (**DTOut**), and the grid extent in each directions, e.g. **XStart**, **XEnd**, **nX**. When **GridType** is 1, the velocity field is written to disk, when **GridType** is 2, both the velocity field and the vorticity field (computed using finite differences) are written. It is possible to export fields at a point (**nX=nY=nZ=1**), a line, a plane, or a 3D grid. When set to “default”, the start time is 0 and the end time is set to the end of the simulation. The outputs are done for $t_{Start} \leq t \leq t_{End}$. When the variable **DTOut** is set to “all”, the AeroDyn time step is used, when it is set to “default”, the OLAF time step is used. An example of input is given below:

3	nGridOut	Number of grid outputs									
GridName	GridType	TStart	TEnd	DTOut	XStart	XEnd	nX	YStart	YEnd	nY	nZ
(-)	(-)	(s)	(s)	(s)	(m)	(m)	(-)	(m)	(m)	(-)	(-)
"box"	2	default	default	all	-200	1000.	5	-150.	150.	20	1
"vert"	1	default	default	default	-200	1000.	100	0.	0.	1	1
"hori"	1	default	default	2.0	-200	1000.	100	-150.	150.	20	1

In this example, the first grid, named “box”, is exported at the AeroDyn time step, and consists of a box of shape 5x20x30 and dimension 1200x300x295. The grid contains both the velocity and vorticity. The two other grids are vertical and horizontal planes containing only the velocity.

AeroDyn15 Input File

Input file modifications

As OLAF is incorporated into the *AeroDyn15* module, a wake computation option has been added to the *AeroDyn15* input file and a line has been added. These additions are as follows.

WakeMod specifies the type of wake model that is used. *WakeMod* = [3] has been added to allow the user to switch from the traditional BEM method to the OLAF method.

FVWFile [string] specifies the OLAF module file, the path is relative to the AeroDyn file, unless an absolute path is provided.

Relevant sections

The BEM options (e.g. tip-loss, skew, and dynamic models) are read and discarded when *WakeMod* = [3]. The following sections and parameters remain relevant and are used by the vortex code:

- general options (e.g., airfoil and tower modeling);
- environmental conditions;
- dynamic stall model options;
- airfoil and blade information;
- tower aerodynamics; and
- outputs.

Output Files

The OLAF module itself does not produce its own output file. However, additional output channels are made available in *AeroDyn15*. As such, the *AeroDyn15* output file is briefly described as well as the outputs made available with OLAF. Visualization files are generated by using the parameter, **WrVTK**. This parameter is available in the OLAF input file, in which case the VTK files are written to the folder `vtk_fvw`, or the primary `.fst` file, in which case the VTK files are written to the folder `vtk`.

Velocity field outputs can be exported as VTK files. The user can control these outputs using **nGridOut** and the subsequent table.

Results File

OpenFAST generates a master results file that includes the *AeroDyn15* results. The results are in table format, where each column is a data channel, and each row corresponds to a simulation-output time step. The data channels are specified in the *OUTPUTS* section in the *AeroDyn15* primary input file. The column format of the AeroDyn-generated files is specified using the **OutFmt** parameter of the OpenFAST driver input file.

OLAF Theory

This section details the OLAF method and provides an overview of the computational method, followed by a brief explanation of its integration with OpenFAST.

Introduction - Vorticity Formulation

The vorticity equation for incompressible homogeneous flows in the absence of non-conservative force is given by Eq. (4.1)

$$\frac{d\vec{\omega}}{dt} = \frac{\partial \vec{\omega}}{\partial t} + \underbrace{(\vec{u} \cdot \nabla) \vec{\omega}}_{\text{convection}} = \underbrace{(\vec{\omega} \cdot \nabla) \vec{u}}_{\text{strain}} + \underbrace{\nu \Delta \vec{\omega}}_{\text{diffusion}} \quad (4.1)$$

Here, $\vec{\omega}$ is the vorticity, \vec{u} is the velocity, and ν is the viscosity. In free vortex wake methods, the vorticity equation is used to describe the evolution of the wake vorticity. Different approximations are introduced to ease its resolution, such as projecting the vorticity onto a discrete number of vortex elements (here vortex filaments), and separately treating the convection and diffusion steps, known as viscous-splitting. Several complications arise from the method; in particular, the discretization requires a regularization of the vorticity field (or velocity field) to ensure a smooth approximation.

The forces exerted by the blades onto the flow are expressed in vorticity formulation as well. This vorticity is bound to the blade and has a circulation associated with the lift force. A lifting-line formulation is used here to model the bound vorticity.

The different models of the implemented free vortex code are described in the following sections.

Discretization - Projection

The numerical method uses a finite number of states to model the continuous vorticity distribution. To achieve this, the vorticity distribution is projected onto basis function which is referred to as vortex elements. Vortex filaments are here used as elements that represents the vorticity field. A vortex filament is delimited by two points and hence assumes a direction formed by these two points. A vorticity tube is oriented along the unit vector \vec{e}_x of cross section dS and length l . It can then be approximated by a vortex filament of length l oriented along the same direction. The total vorticity of the tube and the vortex filaments are the same and related by:

$$\vec{\omega} dS = \vec{\Gamma} \quad (4.2)$$

where $\vec{\Gamma}$ is the circulation intensity of the vortex filament. If the vorticity tubes are complex and occupy a large volume, the projection onto vortex filaments is difficult and the projection onto vortex particle is more appropriate. Assuming the wake is confined to a thin vorticity layer which defines a velocity jump of know direction, it is possible to approximate the wake vorticity sheet as a mesh of vortex filaments. This is the basis of vortex filament wake methods. Vortex filaments are a singular representation of the vorticity field, as they occupy a line instead of a volume. To better represent the vorticity field, the filaments are “inflated”, a process referred to as regularization (see [Section 4.2.2](#)). The regularization of the vorticity field also regularizes the velocity field and avoids the singularities that would otherwise occur.

Lifting-Line Representation

The code relies on a lifting-line formulation. Lifting-line methods effectively lump the loads at each cross-section of the blade onto the mean line of the blade and do not account directly for the geometry of each cross-section. In the vorticity-based version of the lifting-line method, the blade is represented by a line of varying circulation. The line follows the motion of the blade and is referred to as “bound” circulation. The bound circulation does not follow the same dynamic equation as the free vorticity of the wake. Instead, the intensity is linked to airfoil lift via the Kutta-Joukowski theorem. Spanwise variation of the bound circulation results in vorticity being emitted into the the wake. This is referred to as “trailed vorticity”. Time changes of the bound circulation are also emitted in the wake, referred to as “shed” vorticity. The subsequent paragraphs describe the representation of the bound vorticity.

Lifting-Line Panels and Emitted Wake Panels

The lifting-line and wake representation is illustrated in [Fig. 4.11](#). The blade lifting-line is discretized into a finite number of panels, each of them forming a four sided vortex rings. The spanwise discretization follows the discretization of the AeroDyn blade input file. The number of spanwise panels, n_{LL} , is one less than the total number of AeroDyn nodes, **NumBLNds**. The sides of the panels coincide with the lifting-line and the trailing edge of the blade. The lifting-line is currently defined as the 1/4 chord location from the leading edge (LE). More details on the panelling is provided in [Section 4.2.2](#). At a given time step, the circulation of each lifting-line panel is determined according to one of the three methods developed in [Section 4.2.2](#). At the end of the time step, the circulation of each lifting-line panel is emitted into the wake, forming free vorticity panels. To satisfy the Kutta condition, the circulation of the first near wake panel and the bound circulation are equivalent (see [Fig. 4.11 b](#)). The wake panels model the thin shear layer resulting from the continuation of the blade boundary layer. This shear layer can be modelled using a continuous distribution of vortex doublets. A constant doublet strength is assumed on each panel, which in turn is equivalent to a vortex ring of constant circulation.

The current implementation stores the positions and circulations of the panel corner points. In the vortex ring formulation, the boundary between two panels corresponds to a vortex segment of intensity equal to the difference of circulation between the two panels. The convention used to define the segment intensity based on the panels intensity is shown in [Fig. 4.11 c](#). Since the circulation of the bound panels and the first row of near wake panels are equal, the vortex segments located on the trailing edge have no circulation.

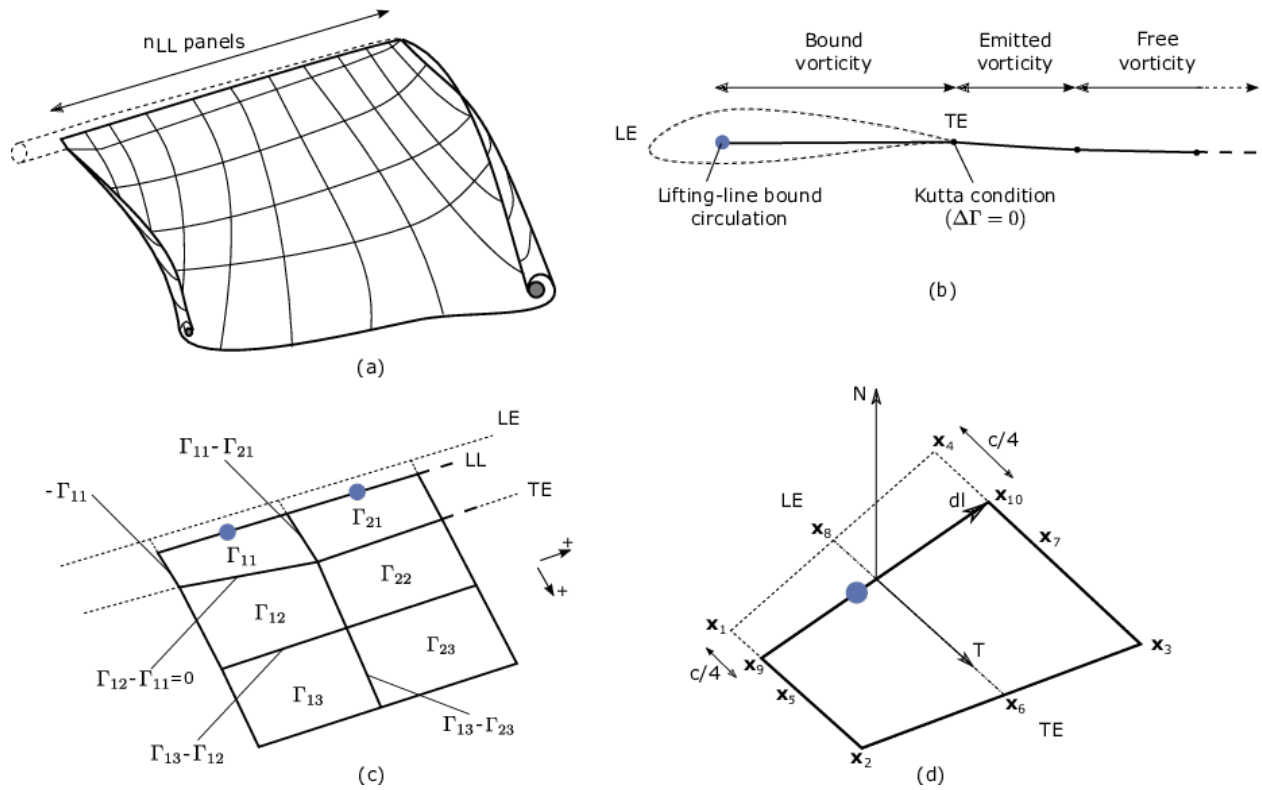


Fig. 4.11: Wake and lifting-line vorticity discretized into vortex ring panels. (a) Overview. (b) Cross-sectional view, defining the leading-edge, trailing edge, and lifting-line. (c) Circulation of panels and corresponding circulation for vorticity segments between panels. (d) Geometrical quantities for a lifting-line panel.

Panelling

The definitions used for the panelling of the blade are given in Fig. 4.11 d, following the notations of van Garrel ([olaf-vG03]). The leading edge and trailing edge (TE) locations are directly obtained from the AeroDyn mesh. At two spanwise locations, the LE and TE define the corner points: $\vec{x}_1, \vec{x}_2, \vec{x}_3$, and \vec{x}_4 . The current implementation assumes that the aerodynamic center, the lifting-line, and the 1/4 chord location all coincide. For a given panel, the lifting-line is then delimited by the points $\vec{x}_9 = 3/4 \vec{x}_1 + 1/4 \vec{x}_2$ and $\vec{x}_{10} = 3/4 \vec{x}_4 + 1/4 \vec{x}_3$. The mid points of the four panel sides are noted $\vec{x}_5, \vec{x}_6, \vec{x}_7$, and \vec{x}_8 . The lifting-line vector (\vec{dl}) as well as the vectors tangential (\vec{T}) and normal (\vec{N}) to the panel are defined as:

$$\vec{dl} = \vec{x}_{10} - \vec{x}_9, \quad \vec{T} = \frac{\vec{x}_6 - \vec{x}_8}{|\vec{x}_6 - \vec{x}_8|}, \quad \vec{N} = \frac{\vec{T} \times \vec{dl}}{|\vec{T} \times \vec{dl}|} \quad (4.3)$$

The area of the panel is obtained as $dA = |(\vec{x}_6 - \vec{x}_8) \times (\vec{x}_7 - \vec{x}_5)|$. For **CircSolvMethod**=[1], the control points are located on the lifting-line at the location $\vec{x}_9 + \eta_j \vec{dl}$. The factor η_j is determined based on the full-cosine approximation of van Garrel. This is based on the spanwise widths of the current panel, w_j , and the neighboring panels w_{j-1} and w_{j+1} :

$$\begin{aligned} \eta_1 &= \frac{w_1}{w_1 + w_2}, \\ \eta_j &= \frac{1}{4} \left[\frac{w_{j-1}}{w_{j-1} + w_j} + \frac{w_j}{w_j + w_{j+1}} + 1 \right], \quad j = 2..n-1, \\ \eta_n &= \frac{w_{n-1}}{w_{n-1} + w_n} \end{aligned}$$

For an equidistant spacing, this discretization places the control points at the middle of the lifting-line ($\eta = 0.5$). Theoretical circulation results for an elliptic wing with a cosine spacing are retrieved with such discretization since it places the control points closer to stronger trailing segments at the wing extremities (see e.g. [olaf-Ker00]).

Circulation Solving Methods

Three methods are implemented to determine the bound circulation strength. They are selected using the input **CircSolvMethod**, and are presented in the following sections.

CI-Based Iterative Method

The CI-based iterative method determines the circulation within a nonlinear iterative solver that makes use of the polar data at each control point located on the lifting line. The algorithm ensures that the lift obtained using the angle of attack and the polar data matches the lift obtained with the Kutta-Joukowski theorem. At present, it is the preferred method to compute the circulation along the blade span. It is selected with **CircSolvMethod**=[1]. The method is described in the work from van Garrel ([olaf-vG03]). The algorithm is implemented in an iterative approach using the following steps:

1. The circulation distribution from the previous time step is used as a guessed circulation, Γ_{prev} .
2. The velocity at each control points j is computed as the sum of the wind velocity, the structural velocity, and the velocity induced by all the vorticity in the domain, evaluated at the control point location.

$$\vec{v}_j = \vec{V}_0 - \vec{V}_{\text{elast}} + \vec{v}_{\omega, \text{free}} + \vec{v}_{\Gamma_{ll}}$$

$\vec{v}_{\omega, \text{free}}$ is the velocity induced by all free vortex filaments, as introduced in Eq. (4.10). The contribution of $\vec{v}_{\Gamma_{ll}}$ comes from the lifting-line panels and the first row of near wake panels, for which the circulation is set to Γ_{prev}

3. The circulation for all lifting-line panels j is obtained as follows.

$$\Gamma_{ll,j} = \frac{1}{2} C_{l,j}(\alpha_j) \frac{\left[(\vec{v}_j \cdot \vec{N})^2 + (\vec{v}_j \cdot \vec{T})^2 \right]^2 dA}{\sqrt{\left[(\vec{v}_j \times d\vec{l}) \cdot \vec{N} \right]^2 + \left[(\vec{v}_j \times d\vec{l}) \cdot \vec{T} \right]^2}}, \quad \text{with} \quad \alpha_j = \text{atan} \left(\frac{\vec{v}_j \cdot \vec{N}}{\vec{v}_j \cdot \vec{T}} \right)$$

The function $C_{l,j}$ is the lift coefficient obtained from the polar data of blade section j and α_j is the angle of attack at the control point.

4. The new circulation is set using the relaxation factor k_{relax} (**CircSolvRelaxation**):

$$\Gamma_{\text{new}} = \Gamma_{\text{prev}} + k_{\text{relax}} \Delta\Gamma, \quad \Delta\Gamma = \Gamma_{ll} - \Gamma_{\text{prev}}$$

5. Convergence is checked using the criterion k_{crit} (**CircSolvConvCrit**):

$$\frac{\max(|\Delta\Gamma|)}{\text{mean}(|\Gamma_{\text{new}}|)} < k_{\text{crit}}$$

If convergence is not reached, steps 2-5 are repeated using Γ_{new} as the guessed circulation Γ_{prev} .

No-flow-through Method

A Weissinger-L-based representation ([[olaf-Wei47](#)]) of the lifting surface is also available ([[olaf-BL94](#), [olaf-Gup06](#), [olaf-Rib07](#)]). In this method, the circulation is solved by satisfying a no-flow through condition at the 1/4-chord points. It is selected with **CircSolvMethod**=[\[2\]](#).

Prescribed Circulation

The final available method prescribes a constant circulation. A user specified spanwise distribution of circulation is prescribed onto the blades. It is selected with **CircSolvMethod**=[\[3\]](#).

Free Vorticity Convection

The governing equation of motion for a vortex filament is given by the convection equation of a Lagrangian marker:

$$\frac{d\vec{r}}{dt} = \vec{V}(\vec{r}, t) \tag{4.4}$$

where \vec{r} is the position of a Lagrangian marker. The Lagrangian markers are the end points of the vortex filaments. The Lagrangian convection of the filaments stretches the filaments and thus automatically accounts for strain in the vorticity equation.

At present, the Runge-Kutta 4th order (**IntMethod**=[\[1\]](#)) or first order forward Euler (**IntMethod**=[\[5\]](#)) methods are implemented to numerically solve the left-hand side of Eq. (4.4) for the vortex filament location. In the case of the first order Euler method, the convection is then simply: Eq. (4.5).

$$\vec{r} = \vec{r} + \vec{V} \Delta t \tag{4.5}$$

Free Vorticity Convection in Polar Coordinates

The governing equation of motion for a vortex filament is given by:

$$\frac{d\vec{r}(\psi, \zeta)}{dt} = \vec{V}[\vec{r}(\psi, \zeta), t] \quad (4.6)$$

Using the chain rule, Eq. (4.6) is rewritten as:

$$\frac{\partial \vec{r}(\psi, \zeta)}{\partial \psi} + \frac{\partial \vec{r}(\psi, \zeta)}{\partial \zeta} = \frac{\vec{V}[\vec{r}(\psi, \zeta), t]}{\Omega} \quad (4.7)$$

where $d\psi/dt = \Omega$ and $d\psi = d\zeta$ ([olaf-LBB02]). Here, $\vec{r}(\psi, \zeta)$ is the position vector of a Lagrangian marker, and $\vec{V}[\vec{r}(\psi, \zeta)]$ is the velocity.

Induced Velocity and Velocity Field

The velocity term on the right-hand side of Eq. (4.4) is a nonlinear function of the vortex position, representing a combination of the freestream and induced velocities ([olaf-Han08]). The induced velocities at point \vec{x} , caused by each straight-line filament, are computed using the Biot-Savart law, which considers the locations of the Lagrangian markers and the intensity of the vortex elements ([olaf-LBB02]):

$$d\vec{v}(\vec{x}) = \frac{\Gamma}{4\pi} \frac{d\vec{l} \times \vec{r}}{r^3} \quad (4.8)$$

Here, Γ is the circulation strength of the filament, $d\vec{l}$ is an elementary length along the filament, \vec{r} is the vector between a point on the filament and the control point \vec{x} , and $r = |\vec{r}|$ is the norm of the vector. The integration of the Biot-Savart law along the filament length, delimited by the points \vec{x}_1 and \vec{x}_2 leads to:

$$\vec{v}(\vec{x}) = F_\nu \frac{\Gamma}{4\pi} \frac{(r_1 + r_2)}{r_1 r_2 (r_1 r_2 + \vec{r}_1 \cdot \vec{r}_2)} \vec{r}_1 \times \vec{r}_2 \quad (4.9)$$

with $\vec{r}_1 = \vec{x} - \vec{x}_1$ and $\vec{r}_2 = \vec{x} - \vec{x}_2$. The factor F_ν is a regularization parameter, discussed in Section 4.2.2. r_0 is the filament length, where $\vec{r}_0 = \vec{x}_2 - \vec{x}_1$. The distance orthogonal to the filament is:

$$\rho = \frac{|\vec{r}_1 \times \vec{r}_2|}{r_0}$$

The velocity at any point of the domain is obtained by superposition of the velocity induced by all vortex filaments, and by superposition of the primary flow, \vec{V}_0 , (here assumed divergence free):

$$\vec{V}(\vec{x}) = \vec{V}_0(\vec{x}) + \vec{v}_\omega(\vec{x}), \quad \text{with} \quad \vec{v}_\omega(\vec{x}) = \sum_k \vec{v}_k(\vec{x}) \quad (4.10)$$

where the sum is over all the vortex filaments, each of intensity Γ_k . The intensity of each filament is determined by spanwise and time changes of the bound circulation, as discussed in Section 4.2.2. In tree-based methods, the sum over all vortex elements is reduced by lumping together the elements that are far away from the control points.

Regularization

Regularization and viscous diffusion

The singularity that occurs in Eq. (4.8) greatly affects the numerical accuracy of vortex methods. By regularizing the “1-over-r” kernel of the Biot-Savart law, it is possible to obtain a numerical method that converges to the Navier-Stokes equations. The regularization is used to improve the regularity of the discrete vorticity field, as compared to

the “true” continuous vorticity field. This regularization is usually obtained by convolution with a smooth function. In this case, the regularization of the vorticity field and the velocity field are the same. Some engineering models also perform regularization by directly introducing additional terms in the denominator of the Biot-Savart velocity kernel. The factor, F_ν , was introduced in Eq. (4.9) to account for this regularization.

In the convergence proofs of vortex methods, regularization and viscous diffusion are two distinct aspects. It is common practice in vortex filament methods to blur the notion of regularization with the notion of viscous diffusion. Indeed, for a physical vortex filament, viscous effects prevent the singularity from occurring and diffuse the vortex strength with time. The circular zone where the velocity drops to zero around the vortex is referred to as the vortex core. A length increase of the vortex segment will result in a vortex core radius decrease, and vice versa. Diffusion, on the other hand, continually spreads the vortex radially.

Because of the previously mentioned analogy, practitioners of vortex filament methods often refer to regularization as “viscous-core” models and regularization parameters as “core-radii.” Additionally, viscous diffusion is often introduced by modifying the regularization parameter in space and time instead of solving the diffusion from the vorticity equation. The distinction is made explicit in this document when clarification is required, but a loose terminology is used when the context is clear.

Determination of the regularization parameter

The regularization parameter is both a function of the physics being modeled (blade boundary layer and wake) and the choice of discretization. Contributing factors are the chord length, the boundary layer height, and the volume that each vortex filament is approximating. Currently the choice is left to the user (**RegDetMethod**=**[0]**). Empirical results for a rotating blade are found in the work of Gupta ([olaf-Gup06]). As a guideline, the regularization parameter may be chosen as twice the average spanwise discretization of the blade. This guideline is implemented when the user chooses **RegDetMethod**=**[1]**. Further refinement of this option will be considered in the future.

Implemented regularization functions

Several regularization functions have been developed ([olaf-Ran58, olaf-Scu75, olaf-VKM91]). At present, five options are available: 1) No correction, 2) the Rankine method, 3) the Lamb-Oseen method, 4) the Vatistas method, or 5) the denominator offset method. If no correction method is used, (**RegFunction**=**[0]**), $F_\nu = 1$. The remaining methods are detailed in the following sections. Here, r_c is the regularization parameter (**WakeRegParam**) and ρ is the distance to the filament. Both variables are expressed in meters.

Rankine

The Rankine method ([olaf-Ran58]) is the simplest regularization model. With this method, the Rankine vortex has a finite core with a solid body rotation near the vortex center and a potential vortex away from the center. If this method is used (**RegFunction**=**[1]**), the viscous core correction is given by Eq. (4.11).

$$F_\nu = \begin{cases} \rho^2/r_c^2 & 0 < \rho < 1 \\ 1 & \rho > 1 \end{cases} \quad (4.11)$$

Here, r_c is the viscous core radius of a vortex filament, detailed in Section 4.2.2.

Lamb-Oseen

If the Lamb-Oseen method is used [**RegFunction**=**[2]**], the viscous core correction is given by Eq. (4.12).

$$F_\nu = \left[1 - \exp\left(-\frac{\rho^2}{r_c^2}\right) \right] \quad (4.12)$$

Vatistas

If the Vatistas method is used [**RegFunction**=**[3]**], the viscous core correction is given by Eq. (4.13).

$$F_\nu = \frac{\rho^2}{(\rho^{2n} + r_c^{2n})^{1/n}} = \frac{(\rho/r_c)^2}{(1 + (\rho/r_c)^{2n})^{1/n}} \quad (4.13)$$

Here, ρ is the distance from a vortex segment to an arbitrary point ([olaf-Abe16]). Research from rotorcraft applications suggests a value of $n = 2$, which is used in this work ([olaf-BL93]).

Denominator Offset/Cut-Off

If the denominator offset method is used [**RegFunction**=**[4]**], the viscous core correction is given by Eq. (4.14)

$$\vec{v}(\vec{x}) = \frac{\Gamma}{4\pi} \frac{(r_1 + r_2)}{r_1 r_2 (r_1 r_2 + \vec{r}_1 \cdot \vec{r}_2) + r_c^2 r_0^2} \vec{r}_1 \times \vec{r}_2 \quad (4.14)$$

Here, the singularity is removed by introducing an additive factor in the denominator of Eq. (4.9), proportional to the filament length r_0 . In this case, $F_\nu = 1$. This method is found in the work of van Garrel ([olaf-vG03]).

Time Evolution of the Regularization Parameter–Core Spreading Method

There are four available methods by which the regularization parameter may evolve with time: 1) constant value, 2) stretching, 3) wake age, or 4) stretching and wake age. The three latter methods blend the notions of viscous diffusion and regularization. The notation r_{c0} used in this section corresponds to input file parameter value **WakeRegParam**.

Constant

If a constant value is selected, (**WakeRegMethod**=**[1]**), the value of r_c remains unchanged for all Lagrangian markers throughout the simulation and is taken as the value given with the parameter **WakeRegParam** in meters.

$$r_c(\zeta) = r_{c0} \quad (4.15)$$

Here, ζ is the vortex wake age, measured from its emission time.

Stretching

If the stretching method is selected, (**WakeRegMethod**=**[2]**), the viscous core radius is modeled by Eq. (4.16).

$$r_c(\zeta, \epsilon) = r_{c0}(1 + \epsilon)^{-1} \quad (4.16)$$

$$\epsilon = \frac{\Delta l}{l}$$

Here, ϵ is the vortex-filament strain, l is the filament length, and Δl is the change of length between two time steps. The integral in Eq. (4.16) represents strain effects.

This option is not yet implemented.

Wake Age / Core-Spreading

If the wake age method is selected, (**WakeRegMethod**=[3]), the viscous core radius is modeled by Eq. (4.17).

$$r_c(\zeta) = \sqrt{r_{c0}^2 + 4\alpha\delta\nu\zeta} \quad (4.17)$$

where $\alpha = 1.25643$, ν is kinematic viscosity, and δ is a viscous diffusion parameter (typically between 1 and 1,000). The parameter δ is provided in the input file as **CoreSpreadEddyVisc**. Here, the term $4\alpha\delta\nu\zeta$, accounts for viscous effects as the wake propagates downstream. The higher the background turbulence, the more diffusion of the vorticity with time, and the higher the value of δ should be. This method partially accounts for viscous diffusion of the vorticity while neglecting the interaction between the wake vorticity itself or between the wake vorticity and the background flow. It is often referred to as the core-spreading method. Setting **DiffusionMethod**=[1] is the same as using the wake age method (**WakeRegMethod**=[3]).

The time evolution of the core radius is implemented as:

$$\frac{dr_c}{dt} = \frac{2\alpha\delta\nu}{r_c(t)}$$

and $\frac{dr_c}{dt} = 0$ on the blades.

Stretching and Wake Age

If the stretching and wake-age method is selected (**WakeRegMethod**=[4]), the viscous core radius is modeled by Eq. (4.18).

$$r_c(\zeta, \epsilon) = \sqrt{r_{c0}^2 + 4\alpha\delta\nu\zeta(1 + \epsilon)^{-1}} \quad (4.18)$$

This option is not yet implemented.

Diffusion

The viscous-splitting assumption is used to solve for the convection and diffusion of the vorticity separately. The diffusion term $\nu\Delta\vec{\omega}$ represents molecular diffusion. This term allows for viscous connection of vorticity lines. Also, turbulent flows will diffuse the vorticity in a similar manner based on a turbulent eddy viscosity.

The parameter **DiffusionMethod** is used to switch between viscous diffusion methods. Currently, only the core-spreading method is implemented. The method is described in [Section 4.2.2](#) since it is equivalent to the increase of the regularization parameter with the wake age.

State-Space Representation and Integration with OpenFAST

State, Constraint, Input, and Output Variables

The OLAF module has been integrated into the latest version of OpenFAST via *AeroDyn15*, following the OpenFAST modularization framework ([[olaf-Jon13](#), [olaf-SJJ15](#)]). To follow the OpenFAST framework, the vortex code is written as a module, and its formulation comprises state, constraint, and output equations. The data manipulated by the module include the following vectors: constant parameters, \vec{p} ; inputs, \vec{u} ; constrained state, \vec{z} ; states, \vec{x} ; and outputs, \vec{y} . The vectors are defined as follows:

- Parameters, \vec{p} – a set of internal system values that are independent of the states and inputs. The parameters can be fully defined at initialization and characterize the system state and output equations.

- Inputs, \vec{u} – a set of values supplied to the module that, along with the states, are needed to calculate future states and the system output.
- Constraint states, \vec{z} – algebraic variables that are calculated using a nonlinear solver, based on values from the current time step.
- States, \vec{x} – a set of internal values of the module. They are influenced by the inputs and used to calculate future state values and output. Continuous states are employed, meaning that the states are differentiable in time and characterized by continuous time-differential equations.
- Outputs, \vec{y} – a set of values calculated and returned by the module that depend on the states, inputs, and/or parameters through output equations.

The parameters of the vortex code include:

- Fluid characteristics: kinematic viscosity, ν .
- Airfoil characteristics: chord c and polar data – $C_l(\alpha)$, $C_d(\alpha)$, $C_m(\alpha)$.
- Algorithmic methods and parameters, e.g., regularization, viscous diffusion, discretization, wake geometry, and acceleration.

The inputs of the vortex code are:

- Position, orientation, translational velocity, and rotational velocity of the different nodes of the lifting lines (\vec{r}_{ll} , Λ_{ll} , \vec{r}_{ll} , and $\vec{\omega}_{ll}$, respectively), gathered into the vector, $\vec{x}_{\text{elast},ll}$, for conciseness. These quantities are handled using the mesh-mapping functionality and data structure of OpenFAST.
- Disturbed velocity field at requested locations, written $\vec{V}_0 = [\vec{V}_{0,ll}, \vec{V}_{0,m}]$. Locations are requested for lifting-line points, \vec{r}_{ll} , and Lagrangian markers, \vec{r}_m . Based on the parameters, this disturbed velocity field may contain the following influences: freestream, shear, veer, turbulence, tower, and nacelle disturbance. The locations where the velocity field is requested are typically the location of the Lagrangian markers.

The constraint states are:

- The circulation intensity along the lifting lines, Γ_{ll} .

The continuous states are:

- The position of the Lagrangian markers, \vec{r}_m
- The vorticity associated with each vortex element, $\vec{\omega}_e$. For a projection of the vorticity onto vortex segments, this corresponds to the circulation, $\vec{\Gamma}_e$. For each segment, $\vec{\Gamma}_e = \Gamma_e \vec{dl}_e = \vec{\omega}_e dV_e$, with \vec{dl}_e and dV_e , the vortex segment length and its equivalent vortex volume.

The outputs are¹:

- The induced velocity at the lifting-line nodes, $\vec{v}_{i,ll}$
- The locations where the undisturbed wind is computed, \vec{r}_r (typically $\vec{r}_r = \vec{r}_m$).

¹ The loads on the lifting line are not an output of the vortex code; their calculation is handled by a separate submodule of *AeroDyn*.

State, Constraint, and Output Equations

An overview of the states, constraints, and output equations is given here. More details are provided in [Section 4.2.2](#). The constraint equation is used to determine the circulation distribution along the span of each lifting line. For the van Garrel method, this circulation is a function of the angle of attack along the blade and the airfoil coefficients. The angle of attack at a given lifting-line node is a function of the undisturbed velocity, $\vec{v}_{0,ll}$, and the velocity induced by the vorticity, $\vec{v}_{i,ll}$, at that point. Part of the induced velocity is caused by the vorticity being shed and trailed at the current time step, which in turn is a function of the circulation distribution along the lifting line. This constraint equation may be written as:

$$\vec{Z} = \vec{0} = \vec{\Gamma}_{ll} - \vec{\Gamma}_p\left(\vec{\alpha}(\vec{x}, \vec{u}), \vec{p}\right)$$

where $\vec{\Gamma}_p$ is the function that returns the circulation along the blade span, according to one of the methods presented in [Section 4.2.2](#).

The state equation specifies the time evolution of the vorticity and the convection of the Lagrangian markers:

$$\begin{aligned} \frac{d\vec{\omega}_e}{dt} &= \left[(\vec{\omega} \cdot \nabla) \vec{v} + \nu \nabla^2 \vec{\omega} \right]_e \\ \frac{d\vec{r}_m}{dt} &= \vec{V}(\vec{r}_m) = \vec{V}_0(\vec{r}_m) + \vec{v}_\omega(\vec{r}_m) = \vec{V}_0(\vec{r}_m) + \vec{V}_\omega(\vec{r}_m, \vec{r}_m, \vec{\omega}) \end{aligned} \quad (4.19)$$

Here,

- \vec{v}_ω is the velocity induced by the vorticity in the domain;
- $\vec{V}_\omega(\vec{r}, \vec{r}_m, \vec{\omega})$ is the function that computes this induced velocity at a given point, \vec{r} , based on the location of the Lagrangian markers and the intensity of the vortex elements;
- the subscript e indicates that a quantity is applied to an element; and
- the vorticity, $\vec{\omega}$, is recovered from the vorticity of the vortex elements by means of discrete convolutions.

For vortex-segment simulations, the viscous-splitting algorithm is used, and the convection step (Eq. (4.19)) is the main state equation being solved for. The vorticity stretching is automatically accounted for, and the diffusion is performed *a posteriori*. The velocity function, \vec{V}_ω , uses the Biot-Savart law. The output equation is:

$$\begin{aligned} \vec{y}_1 &= \vec{v}_{i,ll} = \vec{V}_\omega(\vec{r}_{ll}, \vec{r}_m, \vec{\omega}) \\ \vec{y}_2 &= \vec{r}_r \end{aligned}$$

Integration with AeroDyn15

The vortex code has been integrated as a submodule of the aerodynamic module of OpenFAST, *AeroDyn15*. The data workflow between the different modules and submodules of OpenFAST is illustrated in [Fig. 4.12](#). AeroDyn inputs such as BEM options (e.g., tip-loss factor), skew model, and dynamic inflow are discarded when the vortex code is used. The environmental conditions, tower shadow, and dynamic stall model options are used. This integration required a restructuring of the *AeroDyn15* module to isolate the parts of the code related to tower shadow modeling, induction computation, lifting-line-forces computations, and dynamic stall. The dynamic stall model is adapted when used in conjunction with the vortex code to ensure the effect of shed vorticity is not accounted for twice. The interface between *AeroDyn15* and the inflow module, *InflowWind*, was accommodated to include the additionally requested points by the vortex code.

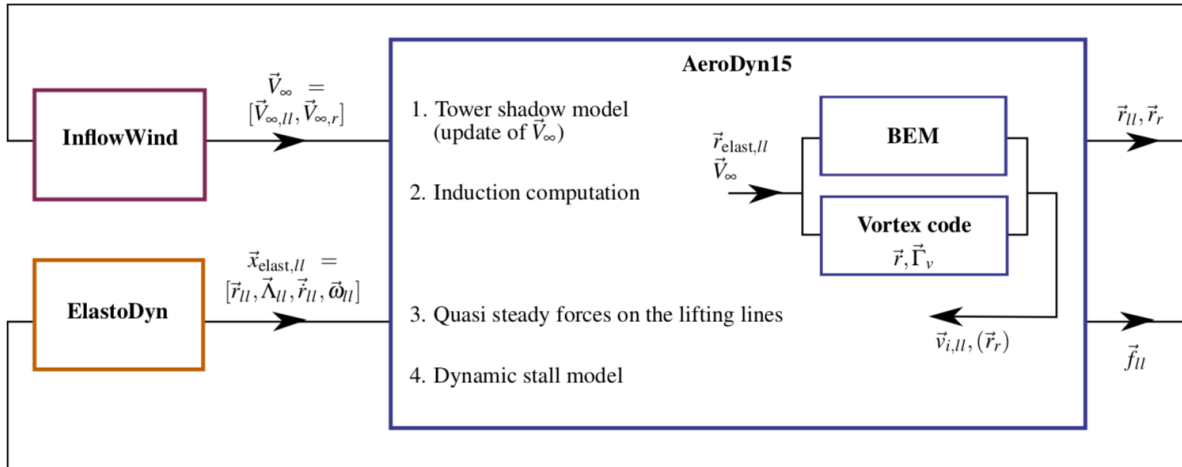


Fig. 4.12: OpenFAST-OLAF code integration workflow

Future Work

This first implementation phase focused on single-turbine capabilities, fulfilling the basic requirements for the design of large and novel rotor concepts. Future development work will turn toward the implementation of features enabling multiple-turbine simulations on medium-to-large-scale computational clusters. The reduction of the computational time will also be of focus. This may be achieved using tree techniques such as the fast multipole method. Further algorithmic options, such as vortex amalgamation in the far wake, will be considered to speed up the simulation. The framework presented in this manual is compatible with grid-free or grid-based vortex particle formulations. Such particle-based implementations will also be envisaged in the future. Further validation of the code against measurements and higher-order tools will be pursued. Applications to cases known to be challenging for the BEM algorithm will also be investigated, such as highly flexible rotors, offshore floating turbines, small-scale wind farms, multiple-rotor turbines, or kites.

The following list contains future work on OLAF software:

- Lagrangian particles
- Multiple turbines, integration into FAST.Farm
- Code speed-up
- Dedicated dynamic stall model

Appendix A: OLAF Primary Input File

Check the regression test cases for updates to this input file.

```

1  ----- OLAF (cOnvecting LAgrangian Filaments) INPUT FILE -----
2  ↪-----
3  Free wake input file for the Helix test case
4  ----- GENERAL OPTIONS -----
5  ↪-----
6  5      IntMethod      Integration method {1: Runge-Kutta 4th order, 5: Forward_
7  ↪Euler 1st order, default: 5} (switch)
8  0.2    DTfvw          Time interval for wake propagation. {default: dtaero} (s)

```

(continues on next page)

(continued from previous page)

```

6 default FreeWakeStart      Time when wake is free. (-) value = always free. {default: 0.
  ↳ 0} (s)
7 default FullCircStart      Time at which full circulation is reached. {default: 0.0} (s)
8 ----- CIRCULATION SPECIFICATIONS -----
  ↳ -----
9 1      CircSolvingMethod    Circulation solving method {1: Cl-Based, 2: No-Flow Through,
  ↳ 3: Prescribed, default: 1 }(switch)
10 default CircSolvConvCrit    Convergence criteria {default: 0.001} [only if
  ↳ CircSolvingMethod=1] (-)
11 default CircSolvRelaxation Relaxation factor {default: 0.1} [only if
  ↳ CircSolvingMethod=1] (-)
12 default CircSolvMaxIter     Maximum number of iterations for circulation solving
  ↳ {default: 30} (-)
13 "NA"   PrescribedCircFile    File containing prescribed circulation [only if
  ↳ CircSolvingMethod=3] (quoted string)
14 =====
15 ----- WAKE OPTIONS -----
  ↳ -----
16 ----- WAKE EXTENT AND DISCRETIZATION -----
  ↳ -----
17 150     nNWPanel            Number of near-wake panels [integer] (-)
18 400     WakeLength          Total wake distance [integer] (number of time steps)
19 default FreeWakeLength      Wake length that is free [integer] (number of time steps)
  ↳ {default: WakeLength}
20 False   FWShedVorticity     Include shed vorticity in the far wake {default: false}
21 ----- WAKE REGULARIZATIONS AND DIFFUSION -----
  ↳ -----
22 0       DiffusionMethod     Diffusion method to account for viscous effects {0: None, 1:
  ↳ Core Spreading, "default": 0}
23 2       RegDeterMethod      Method to determine the regularization parameters {0:
  ↳ Constant, 1: Optimized, 2: Chord-scaled, 3: dr-scaled, default: 0 }
24 2       RegFunction         Viscous diffusion function {0: None, 1: Rankine, 2: LambOseen,
  ↳ 3: Vatistas, 4: Denominator, "default": 3} (switch)
25 3       WakeRegMethod       Wake regularization method {1: Constant, 2: Stretching, 3:
  ↳ Age, default: 1} (switch)
26 0.25    WakeRegFactor       Wake regularization factor (m or -)
27 0.25    BladeRegFactor      Blade regularization factor (m or -)
28 1000    CoreSpreadEddyVisc  Eddy viscosity in core spreading methods, typical values 1-
  ↳ 1000
29 ----- WAKE TREATMENT OPTIONS -----
  ↳ -----
30 False   TwrShadowOnWake     Include tower flow disturbance effects on wake convection
  ↳ {default:false} [only if TwrPotent or TwrShadow]
31 0       ShearModel          Shear Model {0: No treatment, 1: Mirrored vorticity, default:
  ↳ 0}
32 ----- SPEEDUP OPTIONS -----
  ↳ -----
33 2       VelocityMethod      Method to determine the velocity {1:Biot-Savart Segment,
  ↳ 2:Particle tree, 3: Segment tree, default: 1}
34 1.5     TreeBranchFactor     Branch radius fraction above which a multipole calculation is
  ↳ used {default: 2.0} [only if VelocityMethod=2]
35 1       PartPerSegment      Number of particles per segment [only if VelocityMethod=2]

```

(continues on next page)

(continued from previous page)

```

=====
----- OUTPUT OPTIONS -----
↪ -----
38 1      WrVtk              Outputs Visualization Toolkit (VTK) (independent of .fst_
↪ option) {0: NoVTK, 1: Write VTK at each time step} (flag)
39 1      nVTKBlades         Number of blades for which VTK files are exported {0: No VTK_
↪ per blade, n: VTK for blade 1 to n} (-)
40 2      VTKCoord           Coordinate system used for VTK export. {1: Global, 2: Hub,
↪ "default": 1}
41 1      VTK_fps            Frame rate for VTK output (frames per second) {"all" for all_
↪ glue code timesteps, "default" for all OLAF timesteps} [used only if WrVTK=1]
42 0      nGridOut           Number of grid outputs
43 GridName GridType TStart TEnd DTOut XStart XEnd nX YStart YEnd _
↪ nY ZStart ZEnd nZ
44 (-)      (-)      (s)   (s)   (s)   (m)   (m)   (-)   (m)   (m)   _
↪ (-)      (m)      (m)   (-)
45 -----
↪ -----

```

Appendix B: Prescribed Circulation Input File

Check the regression tests for updated versions of this file.

```

1 r/R [-], Gamma [m^2/s]
2 0.048488, 0.000000
3 0.087326, 0.442312
4 0.126163, 6.909277
5 0.165000, 23.678557
6 0.203837, 55.650700
7 0.242674, 74.091529
8 0.281512, 84.205843
9 0.320349, 88.740429
10 0.359186, 89.730814
11 0.398023, 88.568114
12 0.436860, 87.114743
13 0.475698, 86.110557
14 0.514535, 85.705529
15 0.553372, 85.215829
16 0.592209, 84.547371
17 0.631047, 83.774329
18 0.669884, 82.889157
19 0.708721, 81.635600
20 0.747558, 79.788700
21 0.786395, 77.195200
22 0.825233, 73.765100
23 0.864070, 69.275900
24 0.902907, 62.965400
25 0.941744, 53.603300
26 0.980581, 39.854000

```

Appendix C: OLAF List of Output Channels

This is a list of all possible output parameters from the OLAF module. The names are grouped by meaning, but can be ordered in the OUTPUTS section of the *AeroDyn15* primary input file, as the user sees fit. $N\beta$ refers to output node, β , where β is a number in the range [1,9], corresponding to entry, β , in the **OutNd** list. $B\alpha$ is prefixed to each output name, where α is a number in the range [1,3], corresponding to the blade number.

Table 4.2: Available OLAF Output Channels

Channel Name(s)	Units	Description
$B\alpha N\beta Gam$	m^2/s	Circulation along the blade

4.2.3 Aeroacoustics Noise Model of OpenFAST

List of Acronyms

BPM	Brooks-Pope-Marcolini airfoil noise model
dB	decibels
dBA	A-weighted decibels
deg	degrees
Hz	hertz
IEA	International Energy Agency
kg	kilograms
kHz	kilohertz
LFC	low-frequency correction
m	meters
N	newtons
NREL	National Renewable Energy Laboratory
rad	radians
s	seconds
SPL	sound pressure level
TBL	turbulent boundary layer
TBL-TE	turbulent boundary layer – trailing edge
TNO	a Netherlands organization for applied scientific research

continues on next page

Table 4.3 – continued from previous page

TE	trailing edge
TI	turbulent inflow
TUM	Technical University of Munich

List of Symbols

l	low frequency	
h	high frequency	
p	airfoil pressure side	
s	airfoil suction side	
t	turbulence	
0	reference	
1	parallel to airfoil chord	
2	normal to airfoil chord	
3	blade spanwise direction	
α	angle of attack	[rad]
β^2	Prandtl-Glauert correction factor	[-]
δ	airfoil boundary layer thickness	[-]
δ^*	airfoil boundary layer displacement thickness	[-]
θ	airfoil boundary layer momentum thickness	[-]
Θ_e, Φ_e	angles between emitter and observer	[rad]
ρ	air density	[kg/m ³]
ω	radial frequency	[rad/s]
A_w	A-weight	[dB]
c	speed of sound	[m/s]
c_i	chord at blade spanwise position i	[m]

continues on next page

Table 4.4 – continued from previous page

d	blade span at station i	[m]
\overline{D}	directivity function	[-]
f	frequency	[Hz]
G	empirical function	[-]
h	height of the trailing edge thickness	[m]
H	airfoil kinematic shape factor	[-]
I	turbulence intensity	[-]
k	wave number	[m ⁻¹]
$\overline{k}, \widehat{k}$	nondimensional wave number	[-]
$\Delta K_1, K_1, K_2$	empirical parameters of the BPM model	[-]
l	spanwise extent of the separation zone from blade tip	[m]
L	lift force	[N]
L_t	length scale	[m]
M	Mach number	[-]
M_c	Mach number past the trailing edge	[-]
r_e	effective observer distance	[m]
Re	Reynolds number	[-]
S^2	Sears function	[-]
St	Strouhal number	[-]
t_x	relative thickness of the airfoil at chordwise position x	[-]
U	local inflow velocity	[m/s]
y	blade spanwise position	[m]
z	height above the ground	[m]
z_0	ground surface roughness	[m]

Introduction

The increasing penetration of wind energy into the electricity mix has been possible thanks to a constantly growing installed capacity, which has so far been mostly located on land. Land-based installations are, however, increasingly constrained by local ordinances and an often-limiting factor that comprises maximum allowable levels of noise. To further increase the number of land-based installations, it is important to develop accurate modeling tools to estimate the noise generated by wind turbines. This allows for a more accurate assessment of the noise emissions and the possibility to design quieter wind turbines.

Wind turbines emit two main sources of noise:

- Aeroacoustics noise from the interaction between rotor blades and the turbulent atmospheric boundary layer
- Mechanical noise from the nacelle component, mostly the gearbox, generator, and yaw mechanism.

This work targets the first class of noise generation and aims at providing a set of open-source models to estimate the aeroacoustics noise generated by an arbitrary wind turbine rotor. The models are implemented in Fortran and are fully coupled to the aeroservoelastic wind turbine simulator OpenFAST. The code is available in the GitHub repository of OpenFAST.¹ The code builds on the implementation of NAFNoise and the documentation presented in [aa-MM03] and [aa-Mor05]. OpenFAST is implemented as a modularization framework and the aeroacoustics model is implemented as a submodule of AeroDyn ([aa-MH05]).

The set of models is described in Section 4.2.3 and exercised on the noise estimate of the International Energy Agency (IEA) land-based reference wind turbine in Section 4.2.3. In Section 4.2.3, we also show a comparison to results obtained running the noise models implemented at the Technical University of Munich. This documentation closes with conclusions, an outlook on future work, and appendices, where the input files to OpenFAST are presented.

Aeroacoustics Noise Models

The aeroacoustics noise of wind turbine rotors emanates from pressure oscillations that are generated along the blades and propagate in the atmosphere. This source of noise has been historically simulated with models characterized by different fidelity levels. At lower fidelity, models correlated aeroacoustics noise with rotor thrust and torque ([aa-Low70, aa-Vit81]). At higher fidelity, three-dimensional incompressible computational fluid dynamics models are coupled with the Ffowcs Williams-Hawkings model to propagate pressure oscillations generated along the surface of the rotor blades to the far field ([aa-KGW+18]). The latter models are often only suitable to estimate noise at low frequency because capturing noise in the audible range, which is commonly defined between 20 (hertz) Hz and 20 kilohertz (kHz), requires a very fine space-time discretization with enormous computational costs.

For the audible range, a variety of models is available in the public domain, and [aa-SBCB18] offers the most recent literature review. These models have inputs that match the inputs and outputs of modern aeroservoelastic solvers, such as OpenFAST, and have therefore often been coupled together. Further, the computational costs of these acoustic models are similar to the costs of modern aeroservoelastic solvers, which has facilitated the coupling.

Models have targeted different noise generation mechanisms following the distinction defined by [aa-BPM89], and the mechanism of turbulent inflow noise. The latter represents a broadband noise source that is generated when a body of arbitrary shape experiences an unsteady lift because of the presence of an incident turbulent flow. For an airfoil, this phenomenon can be interpreted as leading-edge noise. Turbulent inflow noise was the topic of multiple investigations over the past decades and, as a result, multiple models have been published ([aa-SBCB18]). The BPM model includes five mechanisms of noise generation for an airfoil immersed in a flow:

1. Turbulent boundary layer – trailing edge (TBL-TE)
2. Separation stall
3. Laminar boundary layer – vortex shedding
4. Tip vortex

¹ <https://github.com/OpenFAST/openfast>

5. Trailing-edge bluntness – vortex shedding.

For the five mechanisms, semiempirical models were initially defined for the NACA 0012 airfoil. The BPM model is still a popular model for wind turbine noise prediction, and subsequent studies have improved the model by removing some of the assumptions originally adopted. Recent studies have especially focused on the TBL-TE mechanism, which is commonly the dominant noise source of modern wind turbines. As a result, each noise source defined in the BPM model now has a variety of permutations.

The following subsections describe the details of each mechanism and the models implemented in this model of OpenFAST.

Turbulent Inflow

A body of any arbitrary shape, when immersed in a turbulent flow, generates surface pressure fluctuations. Over the years, several formulations of the turbulent inflow noise model have been developed ([aa-SBCB18]). In this model of OpenFAST, the formulation defined in [aa-MGM04] is adopted. The formulation is based on the model of Amiet ([aa-Ami75, aa-PA76]) and is presented in Section 4.2.3. Additionally, the user can activate the correction defined by [aa-MH05], which builds upon the Amiet model and accounts for the thickness of the airfoils adopted along the blade span. This second model is named Simplified Guidati and is presented in Section 4.2.3.

Amiet model

The formulation is based on work from [aa-Ami75] and [aa-PA76], and it represents the blade as a flat plate and neglects the shape of the airfoil.

The model starts by first computing the wave number, k_1 , for a given frequency f :

$$k_1 = \frac{2f}{U_1} \quad (4.20)$$

where U_1 is the incident inflow velocity on the profile. From k_1 , the wave numbers \bar{k}_1 and \hat{k}_1 are computed:

$$\bar{k}_1 = \frac{k_1 c_i}{2} \quad (4.21)$$

$$\hat{k}_1 = \frac{k_1}{k_e} \quad (4.22)$$

where c_i is the local chord, and k_e is the wave number range of energy containing eddies, defined as:

$$k_e = \frac{3}{4L_t}. \quad (4.23)$$

L_t is the turbulent length scale, and many different formulations have been proposed over the years. As default implementation, L_t is defined following the formulation proposed in [aa-ZHS05]:

$$L_t = 25z^{0.35}z_0^{-0.063} \quad (4.24)$$

where z is the height above the ground of the leading edge of section i at a given instant, t , while z_0 is the surface roughness. Note that setting L_t appropriately is a challenge, and advanced users of this model may want to validate this formulation against experimental data.

The value of sound pressure level (SPL) is expressed in one-third octave bands at the given frequency, f , originated at the given blade station, i , which can be computed as:

$$\text{SPL}_{\text{TI}} = 10 \log_{10} \left(\rho^2 c^4 \frac{L_t d}{2r_e^2} M^5 I_1^2 \frac{\hat{k}_1^3}{(1 + \hat{k}_1^2)^{\frac{7}{3}}} \bar{D} \right) + 78.4 \quad (4.25)$$

where ρ is the air density, c the speed of sound, d the blade element span, r_e the effective distance between leading edge and observer, M the Mach number, I_1 the turbulence intensity of the airfoil inflow, and \overline{D} the directivity term. \overline{D} is different below (\overline{D}_l) and above (\overline{D}_h) a certain frequency, which is named “cut-off” and defined as:

$$f_{co} = \frac{10U_1}{\pi c_i}. \quad (4.26)$$

The formulations of \overline{D}_h and \overline{D}_l are presented in [Section 4.2.3](#).

The current implementation offers two approaches to estimate I_1 . The first one is through a user-defined grid of I_1 ; see [Section 4.2.3](#). The second option is to have the code reconstructing I_1 from the turbulent wind grid, where the code computes the airfoil relative position of each blade section, i , at every time instant and, given the rotor speed, reconstructs the inflow component, I_1 , of the turbulence intensity.

Two corrections to this model are also implemented. The first one comprises a correction for the angle of attack, α , in which the effect is neglected in the original formulation from [\[aa-Ami75\]](#) and Amiet and Peterson (1976). This correction is formulated as:

$$SPL_{TI} = SPL_{TI} + 10 \log_{10} (1 + 9a^2). \quad (4.27)$$

The second correction is called low-frequency correction (LFC), and is formulated as:

$$S^2 = \left(\frac{2\pi \bar{k}_1}{\beta^2} + \left(1 + 2.4 \frac{\bar{k}_1}{\beta^2} \right)^{-1} \right)^{-1} \quad (4.28)$$

$$LFC = 10S^2 M \bar{k}_1^2 \beta^{-2} \quad (4.29)$$

$$SPL_{TI} = SPL_{TI} + 10 \log_{10} \left(\frac{LFC}{1 + LFC} \right). \quad (4.30)$$

In (4.28) and (4.29), S^2 represents the squared Sears function, and β^2 is the Prandtl-Glauert correction factor, which is defined as:

$$\beta^2 = 1 - M^2. \quad (4.31)$$

It is worth stressing that numerous alternative formulations of the turbulent inflow noise model exist ([\[aa-SBCB18\]](#)), where the main differences comprise different definitions of L_t and k_1 .

Simplified Guidati

Sound spectra are often overpredicted by the Amiet model implemented here. Guidati ([\[aa-GBW+97\]](#)) derived a correction to the sound pressure levels by adding a term considering shape and camber of the airfoil profiles, but the method proved computationally too expensive for wind turbine simulations. Moriarty et al. ([\[aa-MGM05\]](#)) proposed a simplified model based on geometric characteristics of six wind turbine airfoils. The validity of the correction is limited to Mach numbers on the order of 0.1–0.2 and Strouhal number St below 75. St is defined based on airfoil chord and mean inflow velocity:

$$St = \frac{f c_i}{U_1}. \quad (4.32)$$

The formula for the correction to the noise spectra is provided in Eq. 4 in [\[aa-MGM05\]](#):

$$t = t_{1\%} + t_{10\%} \quad (4.33)$$

$$\Delta SPL_{TI} = - (1.123t + 5.317t^2) (2\pi St + 5) \quad (4.34)$$

where $t_{x\%}$ is the relative thickness of the profile at x position along the chord (i.e., 0% being the leading edge and 100% the trailing edge).

It should be highlighted here that a validation campaign was conducted in a wind tunnel on two-dimensional airfoils ([aa-MGM04]), returning a fairly poor match between the Simplified Guidati model and the experimental results. Therefore, a correction of +10 decibels (dB) on the SPL levels across the whole frequency spectrum was proposed. This correction is still implemented, but a validation at turbine level should assess the accuracy of the models for turbulent inflow. It should also be noted that the code currently does not check whether Mach and Strouhal numbers are within the range of validity of this model.

Turbulent Boundary Layer – Trailing Edge

Airfoils immersed in a flow develop a boundary layer, which at high Reynolds numbers is turbulent. When the turbulence passes over the trailing edge, noise is generated. This noise source was named TBL-TE in [aa-BPM89] and it is a relevant source of aeroacoustics noise for modern wind turbine rotors. Two formulations of TBL-TE noise are implemented in the code: (1) the original formulation from the BPM model, described in Section 4.2.3, and (2) a more recent model developed at the Dutch research institute, TNO, described in Section 4.2.3. Both models take as input the characteristics of the airfoil boundary layer. These must be provided by the user and are discussed in Section 4.2.3.

BPM

The SPL of the TBL-TE noise in the BPM model is made from three contributions:

$$\text{SPL}_{\text{TBL-TE}} = 10 \log_{10} \left(10^{\frac{\text{SPL}_p}{10}} + 10^{\frac{\text{SPL}_s}{10}} + 10^{\frac{\text{SPL}_\alpha}{10}} \right) \quad (4.35)$$

where the subscripts p , s , and α refer to the contributions of pressure side, suction side, and angle of attack, respectively. The equations describing the three contributions are described in great detail in Section 5.1.2, in [aa-BPM89], and are summarized here.

For the suction and pressure contributions, the equations are:

$$\text{SPL}_p = 10 \log_{10} \left(\frac{\delta_p^* M^5 d\bar{D}_h}{r_e^2} \right) + A \left(\frac{\text{St}_p}{\text{St}_1} \right) + (K_1 - 3) + \Delta K_1 \quad (4.36)$$

$$\text{SPL}_s = 10 \log_{10} \left(\frac{\delta_s^* M^5 d\bar{D}_h}{r_e^2} \right) + A \left(\frac{\text{St}_s}{\text{St}_1} \right) + (K_1 - 3). \quad (4.37)$$

The terms in the equations, which are also described in the nomenclature at the beginning of this document, list δ^* as the boundary layer displacement thickness on either side of the airfoil, St , as the Strouhal number based on δ^* , and A , A' , B , ΔK_1 , K_1 , and K_2 as empirical functions based on St .

For the angle-of-attack contribution, a distinction is made above and below the stall angle, which in the original BPM model is set equal to 12.5 degrees, whereas it is here assumed to be the actual stall angle of attack of the airfoil at blade station i . Below stall, SPL_α is equal to:

$$\text{SPL}_\alpha = 10 \log_{10} \left(\frac{\delta_s^* M^5 d\bar{D}_h}{r_e^2} \right) + B \left(\frac{\text{St}_s}{\text{St}_2} \right) + K_2. \quad (4.38)$$

At angles of attack above the stall point, the flow along the profile is fully separated and noise radiates from the whole chord. SPL_p and SPL_s are then set equal to $-\infty$, whereas SPL_α becomes:

$$\text{SPL}_\alpha = 10 \log_{10} \left(\frac{\delta_s^* M^5 d\bar{D}_l}{r_e^2} \right) + A' \left(\frac{\text{St}_s}{\text{St}_2} \right) + K_2. \quad (4.39)$$

Notably, above stall the low-frequency directivity \bar{D}_l is adopted in Eqs. 18 and 19 (see Section 4.2.3).

TNO model

The TNO model is a more recent model to simulate the noise emitted by the vortices shed at the trailing edge of the blades and was formulated by Parchen ([aa-Par98]). The implementation adopted here is the one described in Moriarty et al. (2005). The TNO model uses the spectrum of the wave number, \vec{k} , of unsteady surface pressures to estimate the far-field noise. The spectrum, P , is assumed to be:

$$P(k_1, k_3, \omega) = 4\rho_0^2 \frac{k_1^2}{k_1^2 + k_3^2} \int_0^{10 \frac{\omega}{Mc}} L_2 \overline{u_2^2} \left(\frac{\partial U_1}{\partial x_2} \right)^2 \phi_{22}(k_1, k_3, \omega) \phi_m(\omega - U_c(x_2) k_1) e^{(-2|\vec{k}|x_2)} dx_2. \quad (4.40)$$

In the equation, the indices 1, 2, and 3 refer to the directions parallel to the airfoil chord, normal to the airfoil chord, and along span, respectively; ϕ_{22} is the vertical velocity fluctuation spectrum; ϕ_m is the moving axis spectrum; and U_c is the convection velocity of the eddies along the trailing edge. Lastly, L_2 is the vertical correlation length, perpendicular to the chord length, which indicates the vertical extension of the vortices that convect over the trailing edge. In this work, L_2 is assumed equal to the mixing length, L_m (Moriarty et al. 2005). This decision is partially arbitrary, and dedicated research should better assess the correct integral length to be adopted within the TNO model.

From P , the far-field spectrum, $S(\omega)$, is computed as:

$$S(\omega) = \frac{d\overline{D}_h}{4\pi r_e^2} \int_0^\delta \frac{\omega}{ck_1} P(k_1, 0, \omega) dk_1. \quad (4.41)$$

The implementation of the TNO model is identical to the one described in [aa-MGM05]. The inputs to the model are generated from the boundary layer characteristics provided by the user (see Section 4.2.3).

Laminar Boundary Layer – Vortex Shedding

Another source of airfoil self-noise included in the BPM model is the noise generated by a feedback loop between vortices being shed at the trailing edge and instability waves in the laminar boundary layer. This noise is typically distributed on a narrow band of frequencies and occurs when the boundary layer of the airfoil remains laminar. This may occur in the inboard region of smaller wind turbines, where the Reynolds number can be smaller than 1 million, but hardly occurs in modern rotors that operate at a Reynolds number one order of magnitude larger. The formula to estimate the noise spectrum in a one-third-octave presentation is:

$$\begin{aligned} \text{SPL}_{LBLE-VS} = 10 \log_{10} \left(\frac{\delta_p M^5 d\overline{D}_h}{r_e^2} \right) + G_1 \left(\frac{St'}{St'_{\text{peak}}} \right) \\ + G_2 \left[\frac{\text{Re}_c}{(\text{Re}_c)_0} \right] + G_3(\alpha_*) \end{aligned} \quad (4.42)$$

where G represents empirical functions, St'_{peak} is the peak Strouhal number function of Re_c , which is the Reynolds number at chord, c_i . The subscript $_0$ refers to a reference Reynolds number that is a function of the angle of attack (Brooks et al. 1989).

Tip Vortex

The vortices generated at blade tips are another source of noise of the BPM model. Although rarely relevant in modern wind turbines, the possibility to include this noise source is offered. The sound pressure level is estimated as:

$$\text{SPL}_{\text{Tip}} = 10 \log_{10} \left(\frac{M^2 M_{\text{max}}^2 l^2 \overline{D}_h}{r_e^2} \right) - 30.5 \left(\log_{10} St'' + 0.3 \right)^2 + 126 \quad (4.43)$$

where $M_{\max} = M_{\max}(\alpha_{\text{tip}})$ is the maximum Mach number, measured near the blade tip within the separated flow region that is assumed to depend on α_{tip} , which is the angle of attack at the tip; l is the spanwise extent of the separation zone; and St''' is the Strouhal number based on l . For a round shape of the tip, l is estimated as:

$$l = c_i 0.008 \alpha_{\text{tip}} \quad (4.44)$$

where α_{tip} is the angle of attack of the tip region to the incoming flow. For a square tip, the BPM model estimates l based on the quantity, α'_{tip} , which is defined as:

$$\alpha'_{\text{tip}} = \left[\left(\frac{\frac{\partial L'}{\partial y}}{\left(\frac{\partial L'}{\partial y} \right)_{\text{ref}}} \right)_{y \rightarrow \text{tip}} \right] \alpha_{\text{tip}} \quad (4.45)$$

where L' is the lift per unit span along the blade at position y . For α'_{tip} between 0 and 2 degrees, l becomes:

$$l = c_i \left(0.0230 + 0.0169 \alpha'_{\text{tip}} \right), \quad (4.46)$$

while for α'_{tip} larger than 2 degrees, l is:

$$l = c_i \left(0.0378 + 0.0095 \alpha'_{\text{tip}} \right). \quad (4.47)$$

However, it must be noted that, unfortunately, α_{tip} is not a reliable output of standard aeroelastic models and the impossibility to accurately determine α_{tip} weakens the formulation of the tip vortex noise.

Trailing-Edge Bluntness – Vortex Shedding

Lastly, wind turbine blades are often characterized by a finite height of the trailing edge, which generates noise as a result of vortex shedding. The frequency and amplitude of this noise source depends on the geometry of the trailing edge and is typically characterized by a tonal nature. Adopting flatback and truncated airfoils far outboard along the blade may strengthen this noise source. When this noise source is activated, the user is asked to provide the distribution along the blade span of the blunt thickness of the trailing edge, h , and the solid angle between the suction and pressure sides of the airfoil, Ψ (see [Section 4.2.3](#)). h and Ψ are inputs to the equation:

$$\begin{aligned} \text{SPL}_{TEB-VS} = 10 \log_{10} \left(\frac{\delta_p^* M^5 d \bar{D}_h}{r_e^2} \right) + G_4 \left(\frac{h}{\delta_{\text{avg}}^*}, \Psi \right) \\ + G_5 \left(\frac{h}{\delta_{\text{avg}}^*}, \Psi, \frac{St''}{St''_{\text{peak}}} \right). \end{aligned} \quad (4.48)$$

In the equation, δ_{avg}^* is the average displacement thickness for both sides of the airfoil. Note that this noise source is very sensitive to h and Ψ , which, therefore, should be estimated accurately.

Directivity

The position of one or more observers is specified by the user, as described in [Section 4.2.3](#). The directivity from the BPM model is adopted in this implementation ([aa-BPM89]). The directivity term, \bar{D} , corrects the SPL depending on the relative position of the observer to the emitter. The position is described by the spanwise directivity angle, Φ_e , and by the chordwise directivity angle, Θ_e , which are schematically represented in [Fig. 4.13](#) and defined as:

$$\Phi_e = \text{atan} \left(\frac{z_e}{y_e} \right) \quad (4.49)$$

$$\Theta_e = \text{atan} \left(\frac{y_e \bullet \cos(\Phi_e) + z_e \bullet \sin(\Phi_e)}{x_e} \right) \quad (4.50)$$

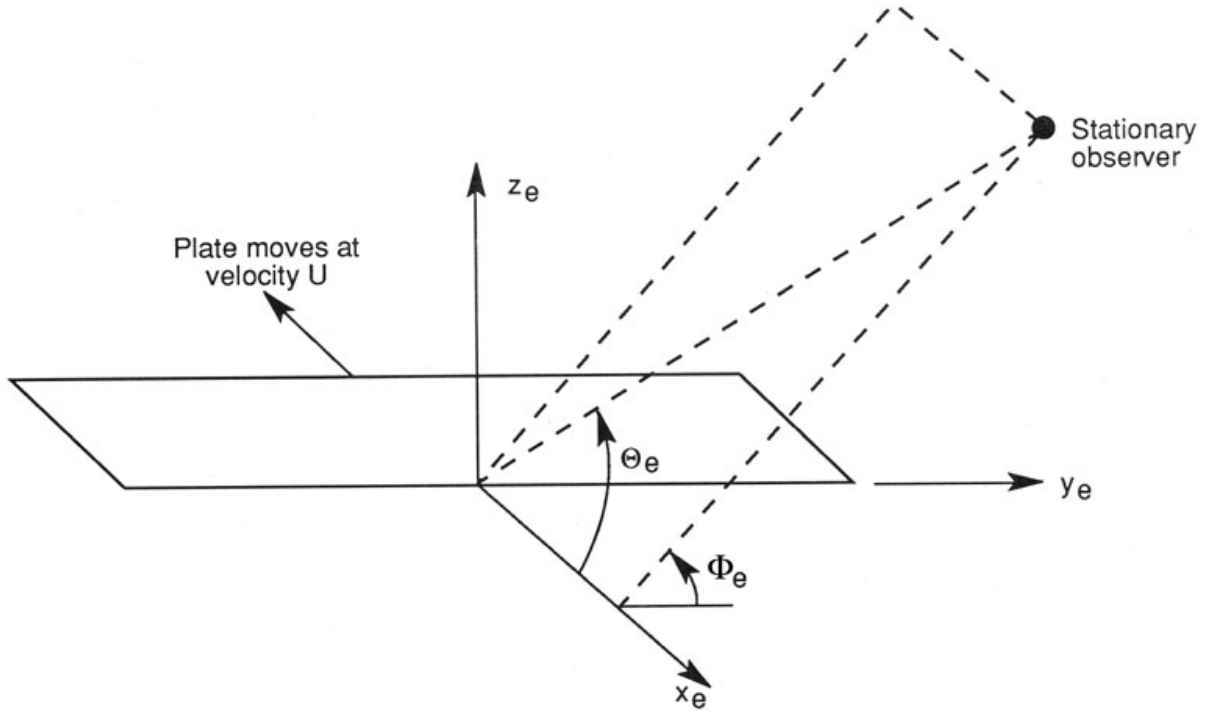


Fig. 4.13: Angles used in the directivity function ([aa-BPM89, aa-MM03])

The reference axis is located at each blade node and x_e is aligned with the chord, y_e is aligned with the span pointing to the blade tip, and z_e is aligned toward the airfoil suction side. Note that in OpenFAST the local airfoil-oriented reference system is used, and a rotation is applied.

Given the angles Θ_e and Φ_e , at high frequency, \bar{D} for the trailing edge takes the expression:

$$\bar{D}_{h-TE}(\Theta_e, \Phi_e) = \frac{2 \sin^2\left(\frac{\Theta_e}{2}\right) \sin^2 \Phi_e}{(1 + M \cos \Theta_e)(1 + (M - M_c) \cos \Theta_e)^2} \quad (4.51)$$

where M_c represents the Mach number past the trailing edge and that is here for simplicity assumed equal to 80% of free-stream M .

For the leading edge, and therefore for the turbulent inflow noise model, at high frequency, \bar{D} is:

$$\bar{D}_{h-LE}(\Theta_e, \Phi_e) = \frac{2 \cos^2\left(\frac{\Theta_e}{2}\right) \sin^2 \Phi_e}{(1 + M \cos \Theta_e)^3} \quad (4.52)$$

Note that this equation was not reported in the NREL Tech Report NREL/TP-5000-75731!

At low frequency, the equation is identical for both leading and trailing edges:

$$\bar{D}_l(\Theta_e, \Phi_e) = \frac{\sin^2 \Theta_e \sin^2 \Phi_e}{(1 + M \cos \Theta_e)^4}. \quad (4.53)$$

Each model distinguishes a different value between low and high frequency. For the TI noise model, the shift between low and high frequency is defined based on \bar{k}_1 . For the TBL-TE noise, the model differences instead shift between below and above stall, where \bar{D}_h and \bar{D}_l are used, respectively.

A-Weighting

The code offers the possibility to weigh the aeroacoustics outputs by A-weighting, which is an experimental coefficient that aims to take into account the sensitivity of human hearing to different frequencies. The A-weight, A_w , is computed as:

$$A_w = \frac{10 \log \left(1.562339 \frac{f^4}{(f^2 + 107.65265^2)(f^2 + 737.86223^2)} \right)}{\log 10} + \frac{10 \log \left(2.422881e16 \frac{f^4}{(f^2 + 20.598997^2)(f^2 + 12194.22^2)} \right)}{\log 10} \quad (4.54)$$

The A-weighting is a function of frequency and is added to the values of sound pressure levels:

$$SPL_{A_w} = SPL + A_w \quad (4.55)$$

Model Verification

Reference Wind Turbine

The noise model of OpenFAST is exercised by simulating the aeroacoustics noise emissions of the IEA Wind Task 37 land-based reference wind turbine ([aa-BTD+19]). The main characteristics of the reference wind turbine are presented in Table 4.5.

Table 4.5: Main Characteristics of the IEA Wind Task 37 Land-Based Reference Wind Turbine

Data	Value	Data	Value
Wind class	International Electrotechnical Commission 3A	Rated electrical power	3.37 megawatts
Rated aerodynamic power	3.6 megawatts	Drivetrain & generator efficiency	93.60%
Rotor diameter	130 meters	Hub height	110 meters
Cut-in wind speed	4 meters/second	Cut-out wind speed	25 meters/second
Rotor cone angle	3 degrees	Nacelle tilt angle	5 degrees
Max blade tip speed	80 meters/second	Rated tip-speed ratio	8.16
Maximum aerodynamic C_p	0.481	Rated rotor speed	11.75 revolutions per minute

The OpenFAST model of the wind turbine is available at <https://github.com/OpenFAST/r-test> and is optionally coupled to the Reference OpenSource Controller.²

Code-to-Code Comparison

A detailed code-to-code comparison was conducted to verify the implementation of the noise models linked to OpenFAST with the implementation available at the Wind Energy Institute of the Technical University of Munich, Germany. The latter is described in Sucameli ([aa-SBCB18]) and is implemented in the wind turbine design framework Cp-Max, which adopts the multibody-based aeroservoelastic solver Cp-Lambda.

The comparison is conducted for the main noise sources—turbulent inflow and the TBL-TE noise—for both the single airfoil profile and full turbine. This helped resolve a few implementation mistakes and small inconsistencies. The comparison is performed with a steady wind of 8 meters per second (m/s), no shear, a rated pitch angle of 1.17 degrees (deg), and a fixed rotor speed of 10.04 revolutions per minute (rpm). A fixed value of 0.1 is assumed for the incident turbulent intensity, I_1 .

Fig. 4.14 shows the predictions in terms of SPL for the Amiet model with the angle-of-attack correction from OpenFAST, the Simplified Guidati model generated by OpenFAST, and the Amiet model from Cp-Max.

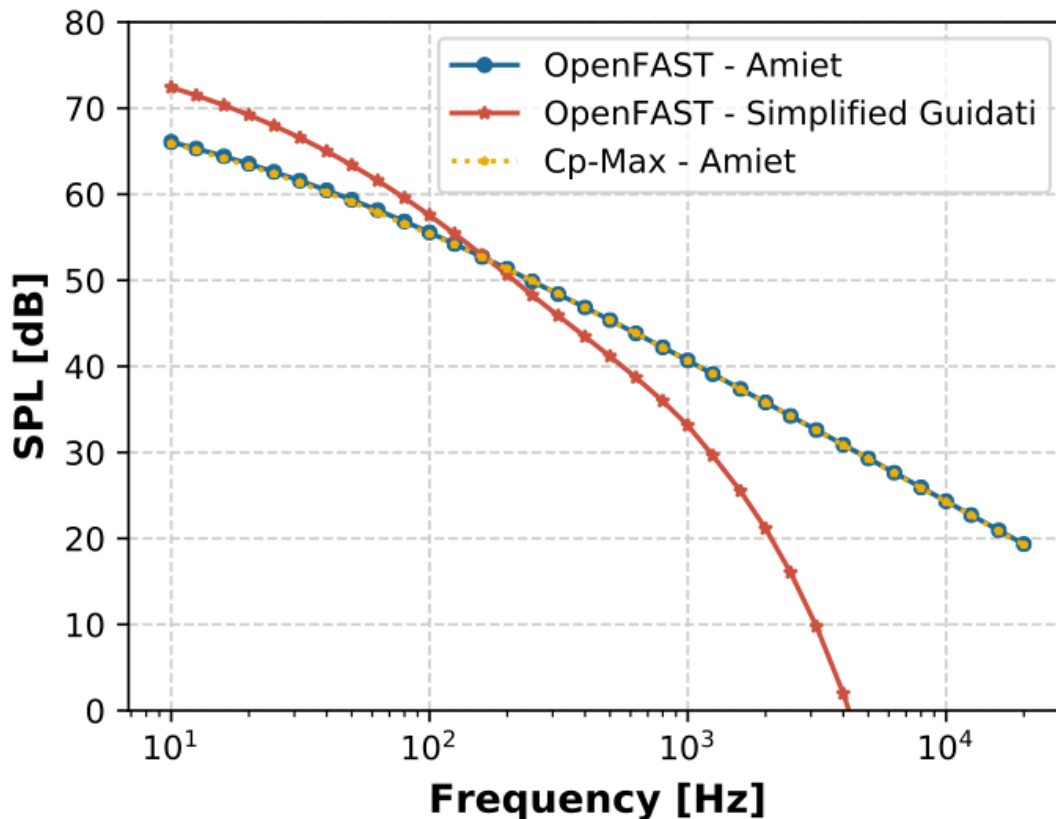


Fig. 4.14: Code-to-code comparison for the TI models

The two implementations of the turbulent inflow Amiet model return a perfect match between OpenFAST and Cp-Max.

² <https://github.com/NREL/ROSCO>

The chosen scenario sees the blade operating at optimal angles of attack and, therefore, the effect of the angle of attack correction is negligible. The plots also show the great difference between the Amiet model and the Simplified Guidati model. It may be useful to keep in mind that the Simplified Guidati model has, in the past, been corrected with a factor of +10 dB, which is applied here.

For the same inflow and rotor conditions, the BPM and TNO TBL-TE noise models are compared in Fig. 4.15. The match is again satisfactory, although slightly larger differences emerge that are attributed to differences in the angles of attack between the two aeroelastic solvers and in different integration schemes in the TNO formulations.

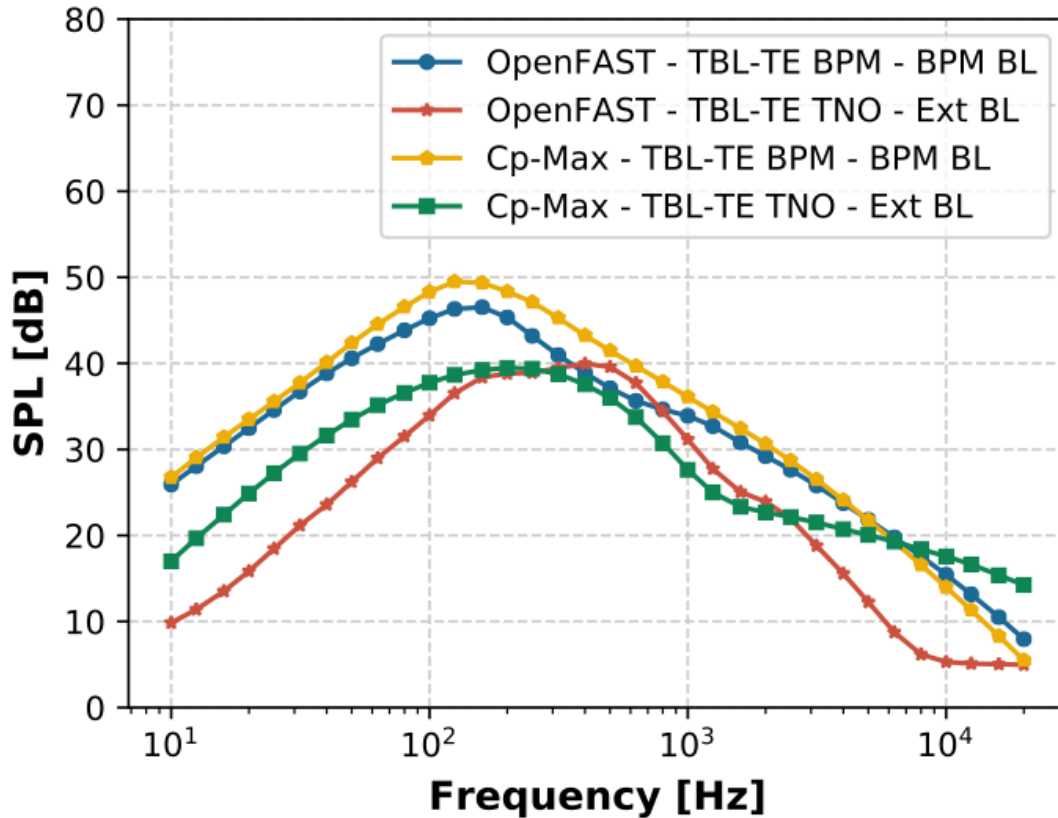


Fig. 4.15: Code-to-code comparison for the BPM and TNO TBL-TE models. The boundary layer properties are estimated from either the BPM model (BPM BL) or defined by the user (Ext BL)

The last comparison looked at the directivity models and the overall sound pressure levels at various observer locations. Simulations are run distributing 200 observers in a horizontal square of 500 meters (m) by 500 m (see Fig. 4.16). The noise is computed from the Amiet and the BPM turbulent boundary layer-trailing edge models. The code-to-code comparison returns similar predictions between OpenFAST and Cp-Max. The comparison is shown in Fig. 4.17.

The main conclusion of this code-to-code comparison is that, to the best of authors' knowledge, the models are now implemented correctly and generate similar SPL and overall SPL levels for any arbitrary observer. Nonetheless, it is clear that all of the presented models are imperfect, and improvements could be made both at the theoretical implementation levels.

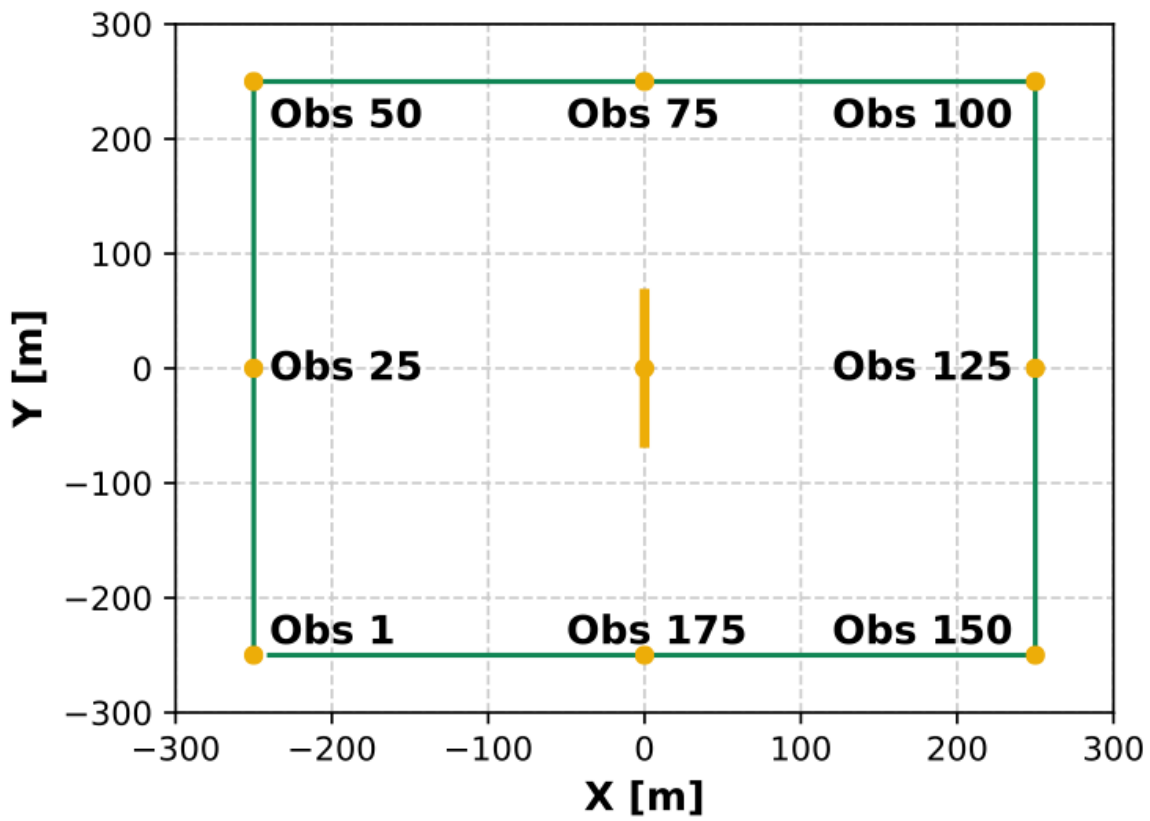


Fig. 4.16: Location and numbering of the observers

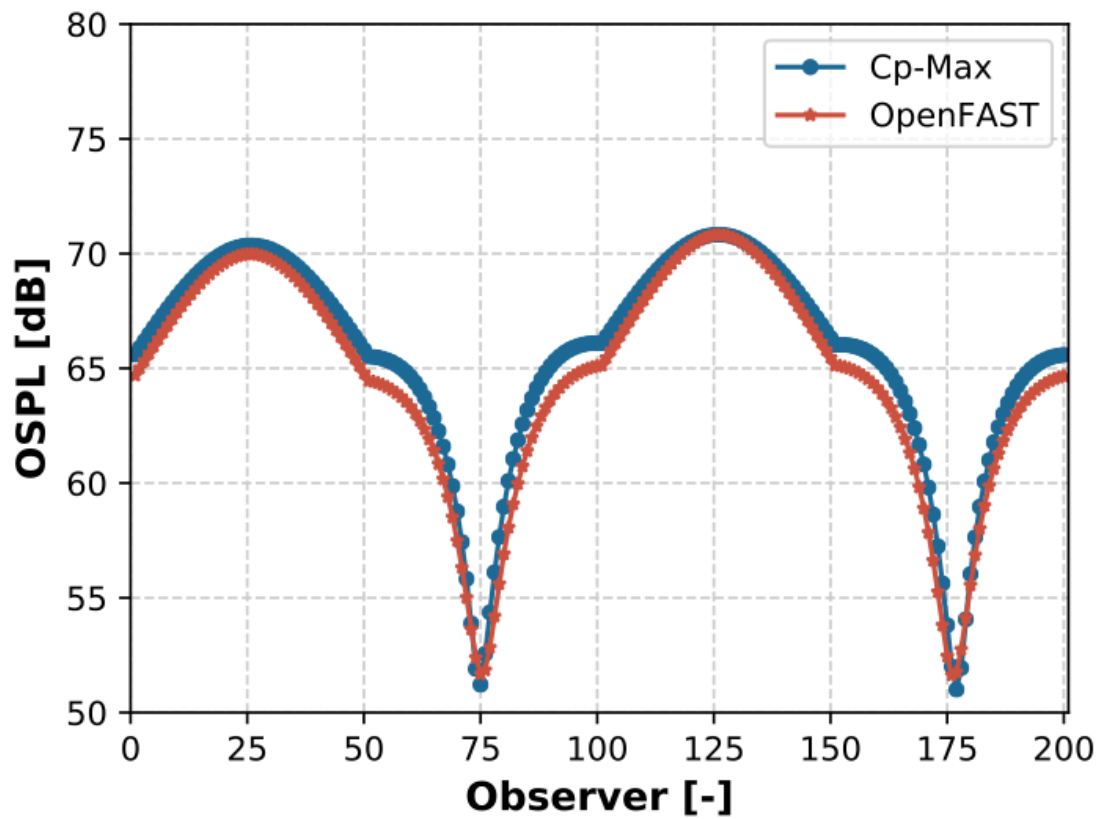


Fig. 4.17: Comparison of overall sound pressure levels for the observers distributed, as shown in the previous figure

Model Usage

The aeroacoustics model of OpenFAST has four options for the outputs:

1. Overall sound pressure level (dB/A-weighted decibels [dBA])—one value per time step per observer is generated
2. Total sound pressure level spectra (dB/dBA)—one spectrum per time step per observer is generated between 10 Hz and 20 kHz
3. Mechanism-dependent sound pressure level spectra (dB/dBA)—one spectrum per active noise mechanism per time step per observer is generated between 10 Hz and 20 kHz.
4. Overall sound pressure level (dB/A-weighted decibels [dBA])—one value per blade per node per time step per observer is generated

The overall SPL from the first option can be used to plot directivity maps of the noise. An example, which was generated using a Python script,³ is shown in Fig. 4.18. The noise map, which shows the overall SPL averaged over 1 rotor revolution, is generated for a steady wind speed of 8 m/s, a fixed rotor speed of 10.04 rpm, and a 1.17-deg pitch angle. In a horizontal circle of 500 m in diameter, 1681 observers are placed at a 2-m height. Only the Simplified Guidati and the BPM TBL-TE noise models are activated.

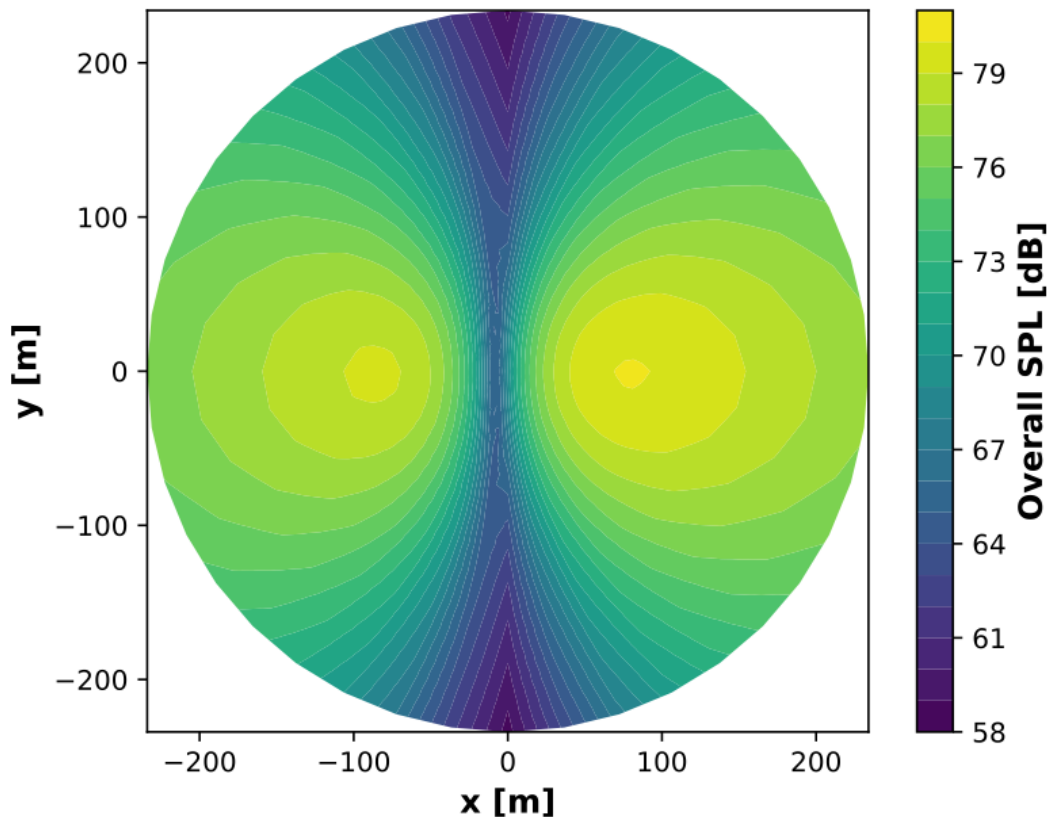


Fig. 4.18: Map of the overall SPL of the reference wind turbine at a 2-m height from Simplified Guidati and BPM TBL-TE noise models. The wind turbine is located at $x=0$, $y=0$. A steady wind of 8 m/s blows from left ($-x$) to right ($+x$).

³ <https://github.com/OpenFAST/python-toolbox>

The second output can be used to generate SPL spectra. These spectra can be computed for various observers and optionally A-weighted to account for human hearing. Fig. 4.19 shows the total SPL spectra computed for the same rotor conditions of the previous example. The A-weight greatly reduces the curve at frequency below 1,000 Hz while slightly increasing those between 1 kHz and 8 kHz.

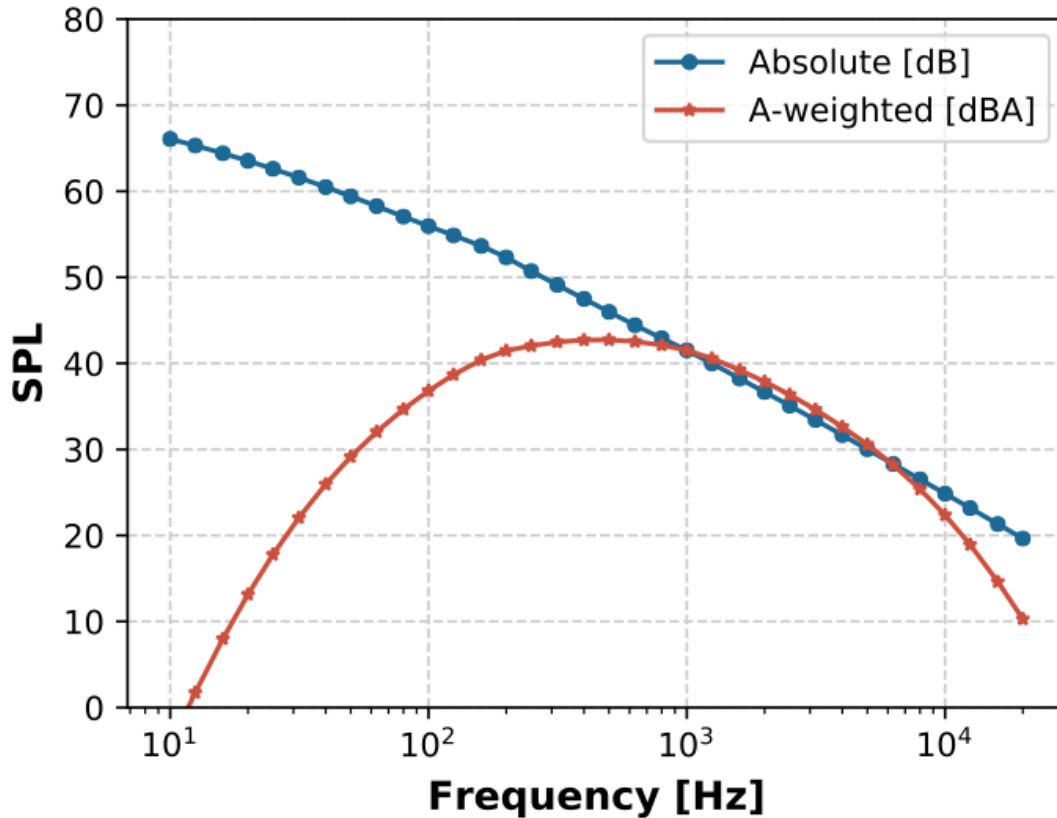


Fig. 4.19: Comparison between absolute and A-weighted SPL

The third output distinguishes the SPL spectrum per mechanism. Fig. 4.20 shows the various SPL spectra estimated by each noise model for the same rotor conditions reported earlier. The total spectrum is visibly dominated by the turbulent inflow, TBL-TE, and trailing-edge bluntiness noise mechanisms. Notably, the latter is extremely sensitive to its inputs, Ψ and h . The reference wind turbine is a purely numerical model, and these quantities have been arbitrarily set. Users should pay attention to these inputs when calling the trailing-edge bluntiness model. Consistent with literature, the laminar boundary layer-vortex shedding and tip vortex noise mechanisms have negative dB values and are, therefore, not visible. Notably, these spectra are not A-weighted, but users can activate the flag and obtain A-weighted spectra.

Finally, the fourth output can be used to visualize the noise emission across the rotor. Fig. 4.21 shows the noise generation of the rotor as seen from an observer located 175 meters downwind at a height of 2 meters. The map is generated by plotting the overall SPL generated by one blade during one rotor revolution. The plot shows that higher noise is observed when the blade is descending (the rotor from behind is seen rotating counterclockwise). This effect, which matches the results shown in [aa-MM03], is explained by the asymmetry of (4.50). Noise is indeed higher when the observer faces the leading edge of an airfoil (high Θ_e), than when it faces the trailing edge (low Θ_e).

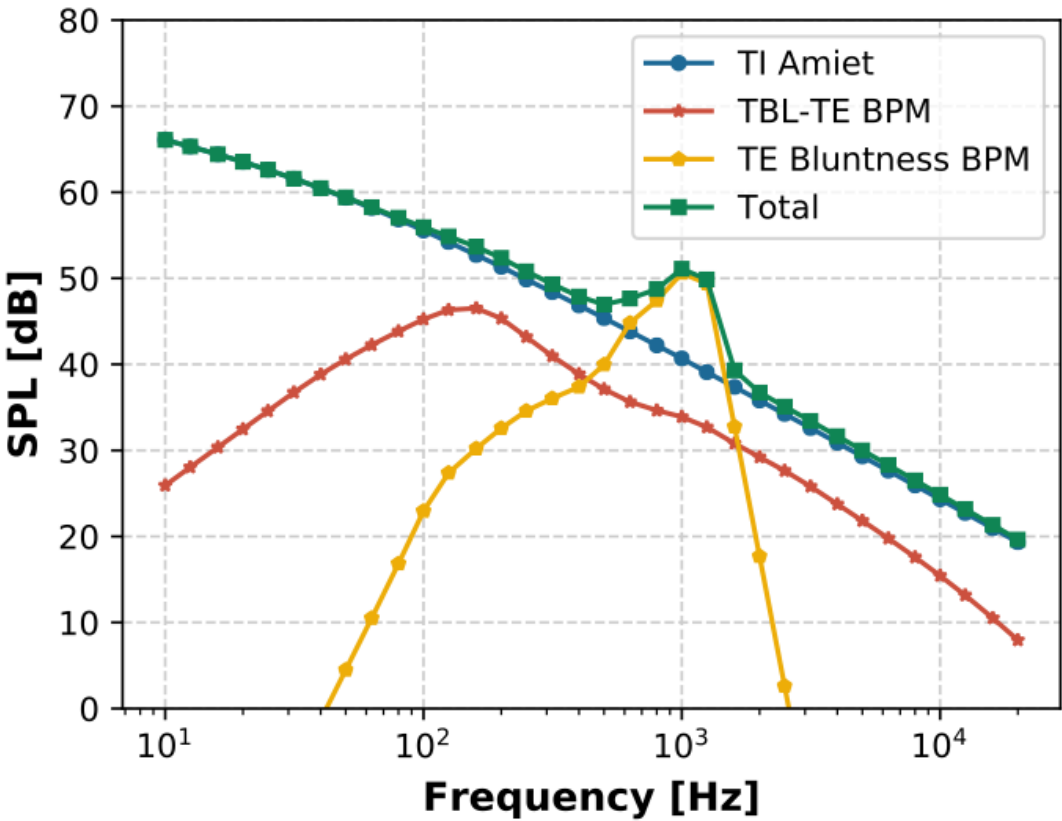


Fig. 4.20: Nonweighted SPL spectra of the various noise mechanisms

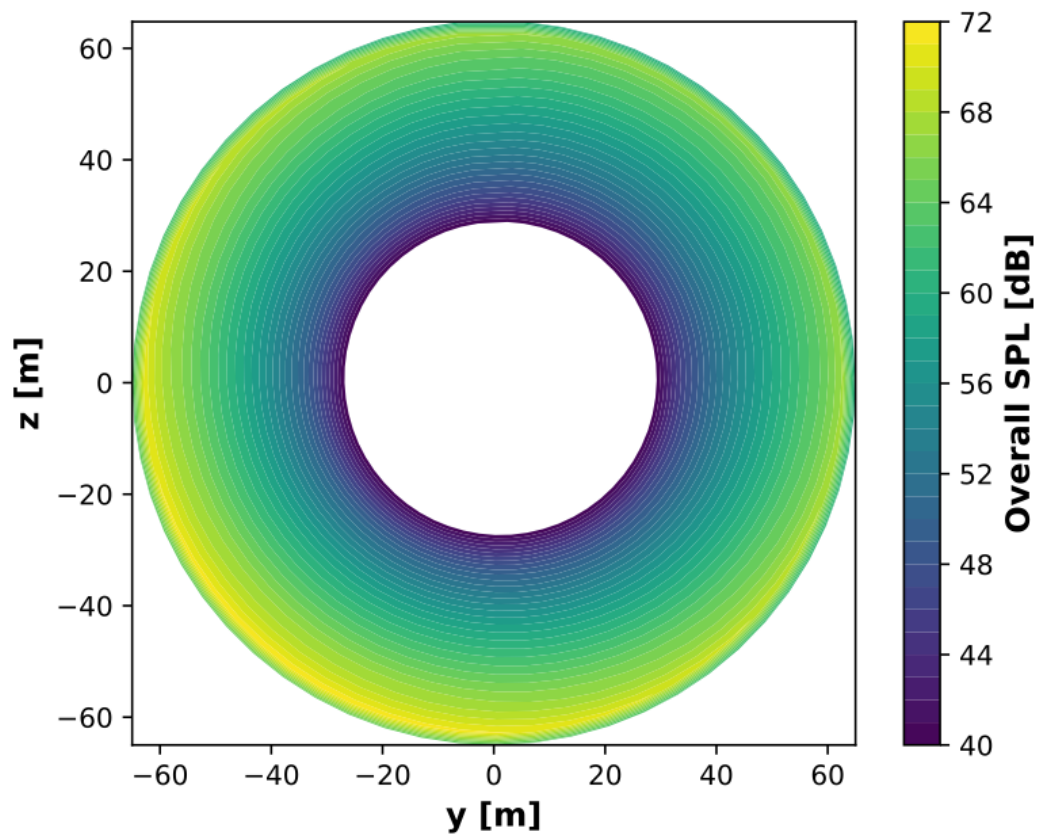


Fig. 4.21: Map of the overall SPL of the rotor of the reference wind turbine from Simplified Guidati and BPM TBL-TE noise models. The observer is located 175 meters downwind at a height of 2 meters.

Conclusions

This document describes a set of frequency-based aeroacoustics models coupled to the open-source aeroservoelastic solver OpenFAST. The goal of these models is to predict the aeroacoustics emissions of wind turbine rotors. The document shows a code-to-code comparison between the models coupled to OpenFAST and the models implemented at the Technical University of Munich and coupled to the aeroservoelastic solver Cp-Lambda. The comparison is performed simulating the aeroacoustics emissions of the IEA Wind Task 37 land-based reference wind turbine. The results show a good agreement between the two implementations. The same turbine model is later used to exercise the aeroacoustics model showcasing its capabilities. Finally, the appendices describe the entries of the input files of OpenFAST to run the aeroacoustics analysis.

Future work will focus on the validation of the aeroacoustics models. In parallel, propagation models will be investigated and implemented. Finally, attention will be dedicated to infrasound noise and to the time-domain models that can simulate it.

Using the Aeroacoustics Model in AeroDyn

A live version of this documentation is available at <https://openfast.readthedocs.io/>. To run the aeroacoustics model, the flag **CompAA** needs to be set to **True** at line 13 of the AeroDyn15 main input file in the inputs block **General Options**. When the flag is set to **True**, the following line must include the name of the file containing the inputs to the aeroacoustics model, which is discussed in [Section 4.2.3](#).

```

1  ----- AERODYN INPUT FILE -----
2  IEA Wind Task 37 land-based reference wind turbine
3  ===== General Options =====
4  False      Echo      - Echo the input to "<rootname>.AD.ech"? (flag)
5  "default"   DT_AA     - Time interval for aerodynamic calculations {or "default"}
6  ↪(s)
7  1          WakeMod    - Type of wake/induction model (switch) {0=none, 1=BEMT}
8  2          AFAeroMod  - Type of blade airfoil aerodynamics model (switch)
9  0          TwrPotent  - Type of tower influence on wind around the tower (switch)
10 0          TwrShadow  - Type of tower influence on wind based on downstream tower
11 ↪shadow (switch) {0=none, 1=Powles model, 2=Eames model}
12 False      TwrAero    - Calculate tower aerodynamic loads? (flag)
13 False      FrozenWake - Assume frozen wake during linearization? (flag)
14 False      CavitCheck - Perform cavitation check? (flag)
15 True       CompAA     - Flag to compute AeroAcoustics calculation
16 "AeroAcousticsInput.dat" AA_InputFile
17 ===== Environmental Conditions =====
18 1.225.      AirDens    - Air density (kg/m^3)
19
20 File continues...

```

Main Input File

The aeroacoustics main input file comprises a series of inputs and flags that should be set appropriately depending on the analysis that should be run. These are split into the subfields General Options, Aeroacoustics Models, Observer Input, and Outputs.

Starting from the General Options, these are:

- **Echo** – True/False: option to rewrite the input file with the correct template

- **DT_AA** – Float: time step of the aeroacoustics computations. Only multiples of the time step **DTAero** of AeroDyn can be used. If set to default, the time step DTAero is adopted.
- **AAStart** – Float: time after which the AeroAcoustics module is run.
- **BldPrent** – Float: percentage value of blade span measured from blade tip that contributes to the noise emissions; 100% corresponds to the entire blade from tip to root.

The field Aeroacoustics Models lists all the flags for the actual noise models:

- **TIMod** – Integer 0/1/2: flag to set the turbulent inflow noise model; 0 turns it off, 1 corresponds to the Amiet model discussed in [Section 4.2.3](#), and 2 corresponds to the Simplified Guidati model presented in [Section 4.2.3](#).
- **TICalcMeth** – Integer 1/2: flag to set the calculation method for the incident turbulence intensity. When set to 1, incident turbulence intensity is defined in a user-defined grid; see [Section 4.2.3](#). When set to 2, incident turbulence intensity is estimated from the time history of the incident flow.
- **TICalcTabFile** – String: name of the text file with the user-defined turbulence intensity grid; see [Section 4.2.3](#).
- **Lturb** – Float: value of L_{turb} used to estimate the turbulent lengthscale used in the Amiet model.
- **TBLTEMod** – Integer 0/1/2: flag to set the TBL-TE noise model; 0 turns off the model, 1 uses the Brooks-Pope-Marcolini (BPM) airfoil noise model (see [Section 4.2.3](#)), and 2 uses the TNO model described in [Section 4.2.3](#).
- **BLMod** – Integer 1/2: flag to set the calculation method for the boundary layer characteristics; 1 uses the simplified equations from the BPM model, 2 loads the files as described in [Section 4.2.3](#). Only used if **TBLTEMod** is different than zero.
- **TripMod** – Integer 0/1/2: if BLMod is set to 1, different semiempirical parameters are used for a nontripped boundary layer (**TRipMod=0**), heavily tripped boundary layer (**TRipMod=1**), or lightly tripped boundary layer (**TRipMod=2**); 2 is typically used for operational wind turbines, whereas 1 is often used for wind tunnel airfoil models.
- **LamMod** – Integer 0/1: flag to activate the laminar boundary layer – vortex shedding model, presented in [Section 4.2.3](#).
- **TipMod** – Integer 0/1: flag to activate the tip vortex model, presented in [Section 4.2.3](#).
- **RoundedTip** – True/False: if **TipMod=1**, this flag switches between a round tip (True) and a square tip (False), see [Section 4.2.3](#).
- **Alprat** – Float: value of the slope of the lift coefficient curve at blade tip; see [Section 4.2.3](#).
- **BluntMod** – Integer 0/1: flag to activate (**BluntMod=1**) the trailing-edge bluntness – vortex shedding model, see [Section 4.2.3](#). If the flag is set to 1, the trailing-edge geometry must be specified in the files as described in [Section 4.2.3](#).

The field Observer Locations contains the path to the file where the number of observers (NrObsLoc) and the respective locations are specified; see [Section 4.2.3](#).

Finally, the set Outputs contains a few options for the output data:

- **AWeighting** – True/False: flag to set whether the sound pressure levels are reported with (True) or without (False) the A-weighting correction; see [Section 4.2.3](#).
- **NAAOutFile** – Integer 1/2/3: flag to set the desired output file. When set to 1, a value of overall sound pressure level at every **DT_AA** time step per observer is printed to file. When set to 2, the first output is accompanied by a second file where the total sound pressure level spectrum is printed per time step per observer. When set to 3, the two first outputs are accompanied by a third file where the sound pressure level spectrum per noise mechanism is printed per time step per observer. When set to 4, a fourth file is generated with the values of overall sound pressure levels per node, per blade, per observer, and per time step.

- The following line contains the file name used to store the outputs. The file name is attached with a 1, 2, 3, and 4 flag based on the **NAAOutFile** options.

The file must be closed by an END command.

```

1  ----- AeroAcoustics Module INPUT FILE -----
   ↳ -----
2  IEA task 37 RWT turbine -- https://github.com/IEAWindTask37/IEA-3.4-130-RWT
3  ===== General Options ↳
   ↳ =====
4  False          Echo          - Echo the input to "<rootname>.AD.NN.ech"? (flag)
5  0.1            DT_AA          - Time interval for aeroacoustics calculations (s), must be a
   ↳ multiple of DT_Aero from AeroDyn15 (or "default")
6  0              AASStart       - Time after which the AeroAcoustics module is run (s)
7  70             BldPrnt        - Percentage of the blade span, starting from the tip, that
   ↳ will contribute to the overall noise levels. (float)
8  ===== Aeroacoustic Models ↳
   ↳ =====
9  2              TImod          - Turbulent Inflow noise model {0: none, 1: Amiet 2: Amiet +
   ↳ Simplified Guidati} (switch)
10 1              TICalcMeth      - Method to estimate turbulence intensity incident to the
   ↳ profile {1: given table, 2: computed on the fly} (switch) [Only used if TImod!=0]
11 "TIGrid_InVerify.txt" TICalcTabFile - Name of the file containing the table for
   ↳ incident turbulence intensity (-) [Only used if TiCalcMeth == 1]
12 0.5            SurfRoughness- Surface roughness value used to estimate the turbulent
   ↳ length scale in Amiet model (m)
13 1              TBLTEMod       - Turbulent Boundary Layer-Trailing Edge noise calculation {0:
   ↳ none, 1:BPM, 2: TNO} (switch)
14 1              BLMod          - Calculation method for boundary layer properties, {1: BPM,
   ↳ 2: Pretabulated} (switch)
15 1              TripMod        - Boundary layer trip model {0:no trip, 1: heavy trip, 2:
   ↳ light trip} (switch) [Only used if BLMod=1]
16 0              LamMod         - Laminar boundary layer noise model {0:none, 1: BPM} (switch)
17 0              TipMod         - Tip vortex noise model {0:none, 1: BPM} (switch)
18 True          RoundedTip      - Logical indicating rounded tip (flag) [Only used if TipMod=1]
19 1.0            Alprat         - Tip lift curve slope (Default = 1.0) [Only used if TipMod=1]
20 0              BluntMod       - Trailing-edge-bluntness - Vortex-shedding model {0:none, 1:
   ↳ BPM} (switch)
21 "AABlade1.dat"  AABlFile(1)    - Name of file containing distributed aerodynamic
   ↳ properties for Blade #1 (-)
22 "AABlade1.dat"  AABlFile(2)    - Name of file containing distributed aerodynamic
   ↳ properties for Blade #2 (-)
23 "AABlade1.dat"  AABlFile(3)    - Name of file containing distributed aerodynamic
   ↳ properties for Blade #3 (-)
24 ===== Observer Input ↳
   ↳ =====
25 "AA_ObserverLocations.dat" ObserverLocations - Name of file containing all
   ↳ observer locations X Y Z (-)
26 ===== Outputs ↳
   ↳ =====
27 False          AWeighting     - A-weighting Flag (flag)
28 3              NrOutFile      - Number of Output files. 1 for Time Dependent Overall SPL,
   ↳ 2 for both 1 and Frequency and Time Dependent SPL as well, or 3 for both 1 and 2 and
   ↳ Acoustics mechanism dependent, 4 for 1-3 and the overall sound pressure levels per

```

(continues on next page)

(continued from previous page)

```

↪blade per node per observer
29 "IEA_1B_RWT-AeroAcoustics_"      AAOutFile   - No Extension needed the resulting file will_
↪have .out Name of file containing
30 END of input file (the word "END" must appear in the first 3 columns of this last_
↪OutList line)
31 -----

```

Boundary Layer Inputs and Trailing Edge Geometry

When the flag **BLMod** is set equal to 2, pretabulated properties of the boundary layer must be provided and are used by the turbulent boundary layer – trailing-edge noise models. The file name is to be specified in the field **BL_file** among the inputs of the file with the airfoil polar coefficients. One airfoil file must be specified per aerodynamic station.

```

1  ! ----- AirfoilInfo Input File -----
2  ! AeroElasticSE FAST driver
3  !
4  !
5  ! -----
6  DEFAULT          InterpOrd    ! Interpolation order to use for quasi-steady table_
↪lookup {1=linear; 3=cubic spline; "default"} [default=1]
7  1                NonDimArea   ! The non-dimensional area of the airfoil (area/
↪chord^2) (set to 1.0 if unsure or unneeded)
8  @"AF20_Coords.txt" NumCoords  ! The number of coordinates in the airfoil shape_
↪file. Set to zero if coordinates not included.
9  AF20_BL.txt      BL_file      ! The file name including the boundary layer_
↪characteristics of the profile. Optional and ignored if the aeroacoustic module is not_
↪called.
10 1                NumTabs      ! Number of airfoil tables in this file. Each_
↪table must have lines for Re and Ctrl.
11 ! -----
12 ! data for table 1

```

The file, in this example named **AF20_BL.txt**, contains 8 inputs, which are tabulated for a given number of Reynolds numbers, **ReListBL**, and a given number of angles of attack, **aoaListBL**. The inputs, which are defined nondimensionally and must be provided for the suction and pressure side of the airfoil above and below the trailing edge, are:

- **Ue_Vinf** – flow velocity at the top of the boundary layer
- **Dstar** – δ^* , boundary layer displacement thickness
- **Delta** – δ , nominal boundary layer thickness
- **Cf** – friction coefficient.

In the following example, the file was generated thanks to a Python script⁴ that runs the boundary layer solver, XFOil. Notably, XFOil, by default, does not return δ , but the boundary layer momentum thickness, θ . δ can be reconstructed using the expression from [aa-DG87]:

$$\delta = \theta \bullet \left(3.15 + \frac{1.72}{H - 1} \right) + \delta^* \quad (4.56)$$

where H is the kinematic shape factor, which is also among the standard outputs of XFOil. Because it is usually impossible to obtain these values for the whole ranges of Reynolds numbers and angles of attack, the code is set to adopt the last available values and print to screen a warning.

⁴ <https://github.com/OpenFAST/python-toolbox>

When the flag **BluntMod** is set to 1, the detailed geometry of the trailing edge must also be defined along the span. Two inputs must be provided, namely the angle, Ψ between the suction and pressure sides of the profile, right before the trailing-edge point, and the height, h , of the trailing edge. Ψ must be defined in degrees, while h is in meters. Note that the BPM trailing-edge bluntness model is very sensitive to these two parameters, which, however, are often not easy to determine for real blades. Fig. 4.22 shows the two inputs.

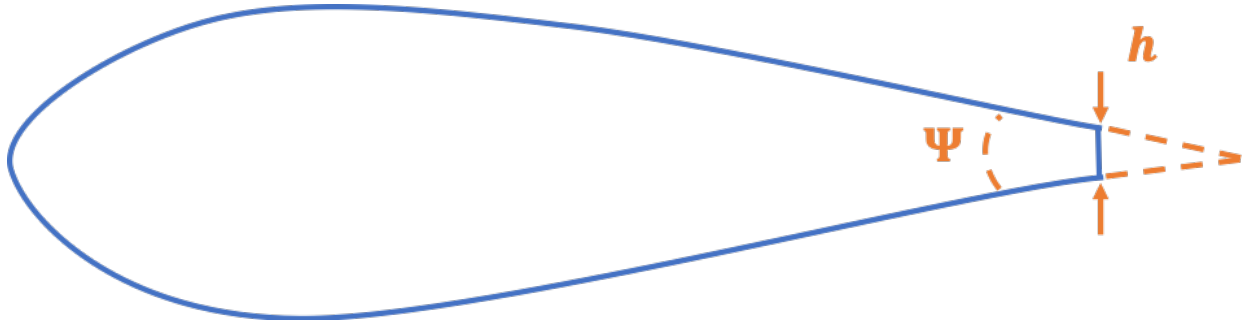


Fig. 4.22: Geometric parameters Ψ and h of the trailing-edge bluntness

One value of Ψ and one value of h per file must be defined. These values are not used if the flag **BluntMod** is set to 0.

```

1  ! Boundary layer characteristics at the trailing edge for the airfoil coordinates of /
  ↳Users/pbortolo/work/2_openfast/noise/verifyAA/OpenFAST_IEA_LB_RWT/Airfoils/AF20_Coords.
  ↳txt
2  ! Legend: aoa - angle of attack (deg), Re - Reynolds number (-, millions), PS - pressure_
  ↳side, SS - suction side, Ue_Vinf - edge velocity (-), Dstar - displacement thickness_
  ↳(-), Delta - nominal boundary layer thickness (-) Cf - friction coefficient (-)
3  4      ReListBL - Number of Reynolds numbers (it corresponds to the number of_
  ↳tables)
4  30      aoaListBL - Number of angles of attack (it corresponds to the number of_
  ↳rows in each table)
5  0.50      - Re
6  aoa      Ue_Vinf_SS      Ue_Vinf_PS      Dstar_SS      Dstar_PS      _
  ↳      Delta_SS      Delta_PS      Cf_SS      Cf_PS
7  (deg)      (-)      (-)      (-)      (-)      (-)
  ↳      (-)      (-)      (-)      (-)
8  -5.000000      8.39390e-01      -8.37360e-01      7.43700e-03      1.07730e-02      _
  ↳      2.75094e-02      5.15849e-02      1.13200e-03      1.58200e-03
9  -3.96552      8.42050e-01      -8.40230e-01      8.26600e-03      9.29500e-03      _
  ↳      2.98650e-02      4.87153e-02      1.04400e-03      1.85700e-03
10 -2.93103      8.45320e-01      -8.43690e-01      9.08800e-03      8.10000e-03      _
  ↳      3.19790e-02      4.70045e-02      9.58000e-04      2.16500e-03
11 -1.89655      8.48230e-01      -8.46710e-01      9.97400e-03      7.33700e-03      _
  ↳      3.44024e-02      4.50456e-02      8.90000e-04      2.35800e-03
12 -0.86207      8.51550e-01      -8.50140e-01      1.09130e-02      6.54100e-03      _
  ↳      3.68822e-02      4.30884e-02      8.26000e-04      2.59900e-03
13 0.17241      8.55000e-01      -8.53670e-01      1.18900e-02      5.92900e-03      _
  ↳      3.96199e-02      4.27416e-02      7.79000e-04      2.87100e-03
14 1.20690      8.63820e-01      -1.04207e+00      1.22130e-02      9.89500e-03      _
  ↳      4.18890e-02      1.68156e-02      8.18000e-04      -1.77000e-04
15 2.24138      8.61500e-01      -8.60210e-01      1.40420e-02      4.88700e-03      _
  ↳      4.51813e-02      3.93105e-02      6.78000e-04      3.28700e-03
16 3.27586      8.64430e-01      -8.63080e-01      1.52900e-02      4.57300e-03      _
  ↳      4.85938e-02      3.82233e-02      6.39000e-04      3.44000e-03

```

(continues on next page)

(continued from previous page)

17	4.31034	8.67960e-01	-8.66600e-01	1.65660e-02	4.09100e-03	↵
	↵ 5.17768e-02	3.63749e-02	5.96000e-04	3.69000e-03		
18	5.34483	8.72300e-01	-8.70850e-01	1.81000e-02	3.81700e-03	↵
	↵ 5.43379e-02	3.52278e-02	5.09000e-04	3.86300e-03		
19	6.37931	8.77930e-01	-8.76410e-01	1.98500e-02	3.39700e-03	↵
	↵ 5.69109e-02	3.31481e-02	4.18000e-04	4.13900e-03		
20	7.41379	8.86840e-01	-8.85140e-01	2.22250e-02	3.15000e-03	↵
	↵ 5.81316e-02	3.19040e-02	2.64000e-04	4.36900e-03		
21	8.44828	9.00620e-01	-8.98660e-01	2.54290e-02	2.75900e-03	↵
	↵ 5.91946e-02	2.95298e-02	1.01000e-04	4.76300e-03		
22	9.48276	9.20300e-01	-9.17700e-01	2.99830e-02	2.48300e-03	↵
	↵ 6.07767e-02	2.75551e-02	5.00000e-06	5.16000e-03		
23	10.51724	9.48080e-01	-9.44440e-01	3.80160e-02	2.13200e-03	↵
	↵ 6.65531e-02	2.48447e-02	-1.60000e-05	5.76800e-03		
24	11.55172	9.89560e-01	-9.84930e-01	5.83630e-02	1.85700e-03	↵
	↵ 8.76076e-02	2.18890e-02	-1.50000e-05	6.49000e-03		
25	12.58621	1.02883e+00	-1.02353e+00	8.80990e-02	1.66700e-03	↵
	↵ 1.21588e-01	2.00072e-02	-1.30000e-05	7.20200e-03		
26	13.62069	1.05789e+00	-1.05226e+00	1.18914e-01	1.51000e-03	↵
	↵ 1.57264e-01	1.78004e-02	-1.10000e-05	7.74800e-03		
27	14.65517	1.07975e+00	-1.07394e+00	1.48726e-01	1.41900e-03	↵
	↵ 1.91423e-01	1.65710e-02	-1.00000e-05	8.15600e-03		
28	15.68966	1.09657e+00	-1.09067e+00	1.76430e-01	1.34400e-03	↵
	↵ 2.22657e-01	1.56180e-02	-9.00000e-06	8.50600e-03		
29	16.72414	1.11040e+00	-1.10441e+00	2.02883e-01	1.26100e-03	↵
	↵ 2.52158e-01	1.43276e-02	-9.00000e-06	8.80900e-03		
30	17.75862	1.12290e+00	-1.11682e+00	2.29606e-01	1.20600e-03	↵
	↵ 2.81695e-01	1.35432e-02	-8.00000e-06	9.07600e-03		
31	18.79310	1.13461e+00	-1.12844e+00	2.55478e-01	1.15500e-03	↵
	↵ 3.10143e-01	1.28744e-02	-8.00000e-06	9.34700e-03		
32	19.82759	1.14605e+00	-1.13974e+00	2.80923e-01	1.08200e-03	↵
	↵ 3.37970e-01	1.16844e-02	-8.00000e-06	9.61200e-03		
33	20.86207	1.15722e+00	-1.15073e+00	3.05117e-01	1.03800e-03	↵
	↵ 3.64240e-01	1.10866e-02	-7.00000e-06	9.87000e-03		
34	21.89655	1.16808e+00	-1.16138e+00	3.27770e-01	9.81000e-04	↵
	↵ 3.88826e-01	1.02373e-02	-7.00000e-06	1.01370e-02		
35	22.93103	1.17845e+00	-1.17148e+00	3.48909e-01	9.33000e-04	↵
	↵ 4.11299e-01	9.52780e-03	-7.00000e-06	1.03870e-02		
36	23.96552	1.18930e+00	-1.18205e+00	3.70277e-01	8.93000e-04	↵
	↵ 4.34300e-01	9.01762e-03	-7.00000e-06	1.06550e-02		
37	25.00000	1.19987e+00	-1.19227e+00	3.90503e-01	8.36000e-04	↵
	↵ 4.55921e-01	8.12755e-03	-7.00000e-06	1.09080e-02		
38	1.00	- Re				
39	aoa	Ue_Vinf_SS	Ue_Vinf_PS	Dstar_SS	Dstar_PS	↵
	↵ Delta_SS	Delta_PS	Cf_SS	Cf_PS		
40	(deg)	(-)	(-)	(-)	(-)	↵
	↵ (-)	(-)	(-)	(-)		
41	-5.00000	8.34300e-01	-8.32480e-01	6.49600e-03	7.74600e-03	↵
	↵ 2.28566e-02	3.97467e-02	8.39000e-04	1.54900e-03		
42	-3.96552	8.37330e-01	-8.35790e-01	7.10100e-03	6.55800e-03	↵
	↵ 2.45059e-02	3.67266e-02	7.84000e-04	1.80000e-03		
43	-2.93103	8.40670e-01	-8.39370e-01	7.75600e-03	5.65600e-03	↵

(continues on next page)

(continued from previous page)

→	2.62162e-02	3.42658e-02	7.27000e-04	2.03700e-03	
44	-1.89655	8.44170e-01	-8.43070e-01	8.45300e-03	4.96000e-03
→	2.79616e-02	3.22259e-02	6.72000e-04	2.25700e-03	
45	-0.86207	8.47840e-01	-8.46890e-01	9.21600e-03	4.45100e-03
→	2.98142e-02	3.07238e-02	6.18000e-04	2.45400e-03	
46	0.17241	8.51730e-01	-8.50900e-01	1.00790e-02	3.95100e-03
→	3.18738e-02	2.89503e-02	5.65000e-04	2.66300e-03	
47	1.20690	8.55470e-01	-8.54730e-01	1.09340e-02	3.54400e-03
→	3.37289e-02	2.74209e-02	5.12000e-04	2.86100e-03	
48	2.24138	8.59040e-01	-8.58320e-01	1.18130e-02	3.25200e-03
→	3.55603e-02	2.64490e-02	4.62000e-04	3.03800e-03	
49	3.27586	8.63480e-01	-8.62770e-01	1.29500e-02	2.91700e-03
→	3.78947e-02	2.47691e-02	4.08000e-04	3.23200e-03	
50	4.31034	8.67590e-01	-8.66830e-01	1.40320e-02	2.69800e-03
→	3.97441e-02	2.39342e-02	3.50000e-04	3.40400e-03	
51	5.34483	8.72380e-01	-8.71540e-01	1.53110e-02	2.43000e-03
→	4.18407e-02	2.22446e-02	2.92000e-04	3.59200e-03	
52	6.37931	8.78360e-01	-8.77360e-01	1.68420e-02	2.23600e-03
→	4.38267e-02	2.12352e-02	2.20000e-04	3.78300e-03	
53	7.41379	8.86030e-01	-8.84810e-01	1.87390e-02	2.00100e-03
→	4.60113e-02	1.94428e-02	1.44000e-04	4.00100e-03	
54	8.44828	8.96310e-01	-8.94850e-01	2.13480e-02	1.83100e-03
→	4.88127e-02	1.83696e-02	5.90000e-05	4.24200e-03	
55	9.48276	9.25990e-01	-9.23230e-01	2.81520e-02	1.56900e-03
→	5.51012e-02	1.62260e-02	-1.00000e-06	4.73700e-03	
56	10.51724	9.66170e-01	-9.62320e-01	4.28900e-02	1.36700e-03
→	7.03103e-02	1.45187e-02	-9.00000e-06	5.34800e-03	
57	11.55172	1.00255e+00	-9.97860e-01	6.33540e-02	1.21700e-03
→	9.26255e-02	1.29836e-02	-7.00000e-06	5.90200e-03	
58	12.58621	1.03100e+00	-1.02578e+00	8.62500e-02	1.10600e-03
→	1.18923e-01	1.16999e-02	-6.00000e-06	6.34900e-03	
59	13.62069	1.05406e+00	-1.04857e+00	1.10634e-01	1.04100e-03
→	1.47132e-01	1.09721e-02	-6.00000e-06	6.70700e-03	
60	14.65517	1.07334e+00	-1.06769e+00	1.35720e-01	9.66000e-04
→	1.76016e-01	9.96935e-03	-5.00000e-06	7.01900e-03	
61	15.68966	1.08881e+00	-1.08308e+00	1.60129e-01	9.17000e-04
→	2.03832e-01	9.33244e-03	-5.00000e-06	7.27400e-03	
62	16.72414	1.10158e+00	-1.09579e+00	1.83765e-01	8.82000e-04
→	2.30423e-01	8.89329e-03	-5.00000e-06	7.49000e-03	
63	17.75862	1.11342e+00	-1.10758e+00	2.08205e-01	8.32000e-04
→	2.57695e-01	8.20477e-03	-4.00000e-06	7.69800e-03	
64	18.79310	1.12407e+00	-1.11817e+00	2.32504e-01	8.01000e-04
→	2.84583e-01	7.81234e-03	-4.00000e-06	7.88600e-03	
65	19.82759	1.13501e+00	-1.12904e+00	2.57953e-01	7.76000e-04
→	3.12682e-01	7.52201e-03	-4.00000e-06	8.07500e-03	
66	20.86207	1.14614e+00	-1.14008e+00	2.83630e-01	7.33000e-04
→	3.41005e-01	6.90325e-03	-4.00000e-06	8.27100e-03	
67	21.89655	1.15868e+00	-1.15248e+00	3.10888e-01	7.07000e-04
→	3.71055e-01	6.60979e-03	-4.00000e-06	8.48600e-03	
68	22.93103	1.17050e+00	-1.16410e+00	3.35623e-01	6.81000e-04
→	3.98279e-01	6.28286e-03	-3.00000e-06	8.69100e-03	
69	23.96552	1.18348e+00	-1.17683e+00	3.61314e-01	6.45000e-04

(continues on next page)

(continued from previous page)

→	4.26528e-01	5.81057e-03	-3.00000e-06	8.91700e-03	
70	25.000000	1.19753e+00	-1.19058e+00	3.87323e-01	6.21000e-04
→	4.54991e-01	5.52432e-03	-3.00000e-06	9.15800e-03	
71	5.00	-	Re		
72	aoa	Ue_Vinf_SS	Ue_Vinf_PS	Dstar_SS	Dstar_PS
→	Delta_SS	Delta_PS	Cf_SS	Cf_PS	
73	(deg)	(-)	(-)	(-)	(-)
→	(-)	(-)	(-)	(-)	(-)
74	-5.000000	8.23420e-01	-8.21880e-01	4.67200e-03	4.76700e-03
→	1.77334e-02	2.96859e-02	6.92000e-04	1.41000e-03	
75	-3.96552	8.25550e-01	-8.24400e-01	5.04400e-03	4.14000e-03
→	1.88321e-02	2.75480e-02	6.57000e-04	1.55000e-03	
76	-2.93103	8.27930e-01	-8.27220e-01	5.46200e-03	3.53900e-03
→	2.00407e-02	2.52464e-02	6.21000e-04	1.70500e-03	
77	-1.89655	8.30490e-01	-8.30120e-01	5.91700e-03	3.10400e-03
→	2.13254e-02	2.34284e-02	5.86000e-04	1.84000e-03	
78	-0.86207	8.33100e-01	-8.33000e-01	6.40000e-03	2.77600e-03
→	2.26264e-02	2.19701e-02	5.50000e-04	1.95800e-03	
79	0.17241	8.35520e-01	-8.35690e-01	6.86100e-03	2.45300e-03
→	2.37731e-02	2.03359e-02	5.15000e-04	2.08300e-03	
80	1.20690	8.38270e-01	-8.38660e-01	7.40600e-03	2.17500e-03
→	2.51176e-02	1.87906e-02	4.79000e-04	2.20700e-03	
81	2.24138	8.41350e-01	-8.41880e-01	8.04900e-03	1.95800e-03
→	2.66635e-02	1.75032e-02	4.40000e-04	2.31900e-03	
82	3.27586	8.43950e-01	-8.44520e-01	8.65200e-03	1.80300e-03
→	2.79650e-02	1.65339e-02	4.03000e-04	2.40900e-03	
83	4.31034	8.48180e-01	-8.48810e-01	9.58300e-03	1.61000e-03
→	3.00737e-02	1.51804e-02	3.59000e-04	2.53200e-03	
84	5.34483	8.53570e-01	-8.54090e-01	1.08300e-02	1.48600e-03
→	3.27612e-02	1.43249e-02	3.08000e-04	2.63700e-03	
85	6.37931	8.72880e-01	-8.73060e-01	1.51570e-02	1.28200e-03
→	4.16833e-02	1.28096e-02	1.92000e-04	2.88700e-03	
86	7.41379	8.92130e-01	-8.91760e-01	1.98220e-02	1.14700e-03
→	4.87740e-02	1.17767e-02	8.30000e-05	3.11600e-03	
87	8.44828	9.17360e-01	-9.16020e-01	2.50640e-02	9.92000e-04
→	5.31945e-02	1.04181e-02	2.00000e-06	3.41900e-03	
88	9.48276	9.42910e-01	-9.40410e-01	3.17040e-02	8.85000e-04
→	5.85499e-02	9.42477e-03	-1.00000e-06	3.70700e-03	
89	10.51724	9.64800e-01	-9.61630e-01	4.02300e-02	7.96000e-04
→	6.64893e-02	8.47323e-03	-2.00000e-06	3.96100e-03	
90	11.55172	9.86420e-01	-9.82570e-01	5.11880e-02	7.23000e-04
→	7.76623e-02	7.65452e-03	-2.00000e-06	4.20700e-03	
91	12.58621	1.00657e+00	-1.00210e+00	6.43270e-02	6.71000e-04
→	9.20001e-02	7.06023e-03	-2.00000e-06	4.43100e-03	
92	13.62069	1.02475e+00	-1.01984e+00	7.93340e-02	6.16000e-04
→	1.09051e-01	6.35528e-03	-1.00000e-06	4.64000e-03	
93	14.65517	1.04370e+00	-1.03850e+00	9.84840e-02	5.79000e-04
→	1.31195e-01	5.91001e-03	-1.00000e-06	4.84500e-03	
94	15.68966	1.06004e+00	-1.05467e+00	1.18503e-01	5.43000e-04
→	1.54410e-01	5.44594e-03	-1.00000e-06	5.02500e-03	
95	16.72414	1.07448e+00	-1.06905e+00	1.39604e-01	5.14000e-04
→	1.78759e-01	5.05912e-03	-1.00000e-06	5.18500e-03	

(continues on next page)

(continued from previous page)

96	17.75862	1.08720e+00	-1.08175e+00	1.61656e-01	4.93000e-04	↵
	↵ 2.03997e-01	4.79726e-03	-1.00000e-06	5.32500e-03		
97	18.79310	1.09867e+00	-1.09324e+00	1.84226e-01	4.68000e-04	↵
	↵ 2.29525e-01	4.45243e-03	-1.00000e-06	5.45500e-03		
98	19.82759	1.10970e+00	-1.10430e+00	2.08500e-01	4.51000e-04	↵
	↵ 2.56774e-01	4.24858e-03	-1.00000e-06	5.57800e-03		
99	20.86207	1.11936e+00	-1.11397e+00	2.32097e-01	4.34000e-04	↵
	↵ 2.83065e-01	4.03443e-03	-1.00000e-06	5.69000e-03		
100	21.89655	1.12815e+00	-1.12274e+00	2.54679e-01	4.14000e-04	↵
	↵ 3.07965e-01	3.77358e-03	-1.00000e-06	5.79400e-03		
101	22.93103	1.13774e+00	-1.13227e+00	2.78750e-01	4.00000e-04	↵
	↵ 3.34530e-01	3.60784e-03	-1.00000e-06	5.90600e-03		
102	23.96552	1.14721e+00	-1.14164e+00	3.02299e-01	3.84000e-04	↵
	↵ 3.60352e-01	3.41109e-03	-1.00000e-06	6.01800e-03		
103	25.00000	1.15816e+00	-1.15244e+00	3.27151e-01	3.68000e-04	↵
	↵ 3.87710e-01	3.21949e-03	-1.00000e-06	6.14600e-03		
104	10.00	- Re				
105	aoa	Ue_Vinf_SS	Ue_Vinf_PS	Dstar_SS	Dstar_PS	↵
	↵ Delta_SS	Delta_PS	Cf_SS	Cf_PS		
106	(deg)	(-)	(-)	(-)	(-)	↵
	↵ (-)	(-)	(-)	(-)	(-)	
107	-5.00000	8.19760e-01	-8.18060e-01	4.17800e-03	4.54900e-03	↵
	↵ 1.65706e-02	2.88150e-02	6.56000e-04	1.23100e-03		
108	-3.96552	8.21540e-01	-8.20450e-01	4.52500e-03	3.74000e-03	↵
	↵ 1.76308e-02	2.59028e-02	6.23000e-04	1.39100e-03		
109	-2.93103	8.23580e-01	-8.22970e-01	4.89400e-03	3.21700e-03	↵
	↵ 1.87333e-02	2.38284e-02	5.91000e-04	1.51700e-03		
110	-1.89655	8.25560e-01	-8.25320e-01	5.25400e-03	2.85300e-03	↵
	↵ 1.97567e-02	2.22669e-02	5.60000e-04	1.62100e-03		
111	-0.86207	8.27870e-01	-8.28060e-01	5.67900e-03	2.46600e-03	↵
	↵ 2.09522e-02	2.03860e-02	5.28000e-04	1.74400e-03		
112	0.17241	8.30330e-01	-8.30840e-01	6.14400e-03	2.18100e-03	↵
	↵ 2.22219e-02	1.88758e-02	4.96000e-04	1.84900e-03		
113	1.20690	8.32880e-01	-8.33650e-01	6.64800e-03	1.94100e-03	↵
	↵ 2.35312e-02	1.74735e-02	4.63000e-04	1.94900e-03		
114	2.24138	8.35130e-01	-8.36090e-01	7.13000e-03	1.75100e-03	↵
	↵ 2.46910e-02	1.62700e-02	4.31000e-04	2.03800e-03		
115	3.27586	8.39970e-01	-8.41060e-01	8.09900e-03	1.56800e-03	↵
	↵ 2.72181e-02	1.50508e-02	3.88000e-04	2.14300e-03		
116	4.31034	8.50470e-01	-8.51560e-01	1.01990e-02	1.37300e-03	↵
	↵ 3.25448e-02	1.36378e-02	3.18000e-04	2.29400e-03		
117	5.34483	8.64450e-01	-8.65280e-01	1.32660e-02	1.23700e-03	↵
	↵ 3.92329e-02	1.26866e-02	2.31000e-04	2.45100e-03		
118	6.37931	8.78610e-01	-8.79110e-01	1.65810e-02	1.08900e-03	↵
	↵ 4.49765e-02	1.14397e-02	1.47000e-04	2.62200e-03		
119	7.41379	8.91030e-01	-8.91080e-01	1.96290e-02	9.93000e-04	↵
	↵ 4.89936e-02	1.06282e-02	7.60000e-05	2.76500e-03		
120	8.44828	9.08900e-01	-9.08620e-01	2.35230e-02	8.71000e-04	↵
	↵ 5.22284e-02	9.45732e-03	2.00000e-06	2.96800e-03		
121	9.48276	9.32700e-01	-9.30700e-01	2.84210e-02	7.79000e-04	↵
	↵ 5.52443e-02	8.61055e-03	-0.00000e+00	3.20000e-03		
122	10.51724	9.51380e-01	-9.48770e-01	3.46600e-02	6.96000e-04	↵

(continues on next page)

(continued from previous page)

↪	6.05165e-02	7.64709e-03	-1.00000e-06	3.39700e-03	
123	11.55172	9.71740e-01	-9.68450e-01	4.35850e-02	6.37000e-04 ↪
↪	6.90670e-02	6.98615e-03	-1.00000e-06	3.59500e-03	
124	12.58621	9.91260e-01	-9.87290e-01	5.44080e-02	5.84000e-04 ↪
↪	8.03205e-02	6.33577e-03	-1.00000e-06	3.78700e-03	
125	13.62069	1.00996e+00	-1.00542e+00	6.74960e-02	5.36000e-04 ↪
↪	9.47613e-02	5.73102e-03	-1.00000e-06	3.97000e-03	
126	14.65517	1.02771e+00	-1.02275e+00	8.31660e-02	5.06000e-04 ↪
↪	1.12645e-01	5.35979e-03	-1.00000e-06	4.13700e-03	
127	15.68966	1.04427e+00	-1.03905e+00	1.00836e-01	4.71000e-04 ↪
↪	1.33082e-01	4.88548e-03	-1.00000e-06	4.29600e-03	
128	16.72414	1.06019e+00	-1.05485e+00	1.21136e-01	4.45000e-04 ↪
↪	1.56673e-01	4.55077e-03	-1.00000e-06	4.44600e-03	
129	17.75862	1.07407e+00	-1.06868e+00	1.42220e-01	4.22000e-04 ↪
↪	1.81035e-01	4.24533e-03	-1.00000e-06	4.57900e-03	
130	18.79310	1.08623e+00	-1.08087e+00	1.64037e-01	4.01000e-04 ↪
↪	2.06006e-01	3.94306e-03	-0.00000e+00	4.69600e-03	
131	19.82759	1.09748e+00	-1.09215e+00	1.87080e-01	3.86000e-04 ↪
↪	2.32142e-01	3.76503e-03	-0.00000e+00	4.80500e-03	
132	20.86207	1.10794e+00	-1.10267e+00	2.10804e-01	3.67000e-04 ↪
↪	2.58816e-01	3.50553e-03	-0.00000e+00	4.90800e-03	
133	21.89655	1.11776e+00	-1.11253e+00	2.35256e-01	3.54000e-04 ↪
↪	2.86067e-01	3.34709e-03	-0.00000e+00	5.00500e-03	
134	22.93103	1.12664e+00	-1.12138e+00	2.58366e-01	3.43000e-04 ↪
↪	3.11568e-01	3.20986e-03	-0.00000e+00	5.09600e-03	
135	23.96552	1.13635e+00	-1.13106e+00	2.83067e-01	3.28000e-04 ↪
↪	3.38816e-01	3.02058e-03	-0.00000e+00	5.19400e-03	
136	25.00000	1.14573e+00	-1.14034e+00	3.06604e-01	3.16000e-04 ↪
↪	3.64612e-01	2.86692e-03	-0.00000e+00	5.29100e-03	

Observer Positions

The number and position of observers is set in the file `ObserverLocations`, which is explained in [Section 4.2.3](#). The positions must be specified in the OpenFAST global inertial frame coordinate system, which is located at the tower base and has the x-axis pointing downwind, the y-axis pointing laterally, and the z-axis pointing vertically upward. A scheme of the coordinate system for the observers is shown in [Fig. 4.23](#).

The International Energy Agency Wind Task 37 land-based reference wind turbine, which is shown in [Table 4.5](#), has a hub height of 110 meters and a rotor radius of 65 meters, and has the International Electrotechnical Commission 61400-11 standards compliant observer located at:

$x = 175$ [m]

$y = 0$ [m]

$z = 0$ [m].

An example of a file listing four observers located at a 2-meter height is shown here:

```

1 4 NrObsLoc - Total Number of observer locations
2 X Observer location in tower-base coordinate X horizontal (m), Y Observer location in
↪ tower-base coordinate Y Lateral (m), Z Observer location in tower-base coordinate Z
↪ Vertical (m)
```

(continues on next page)

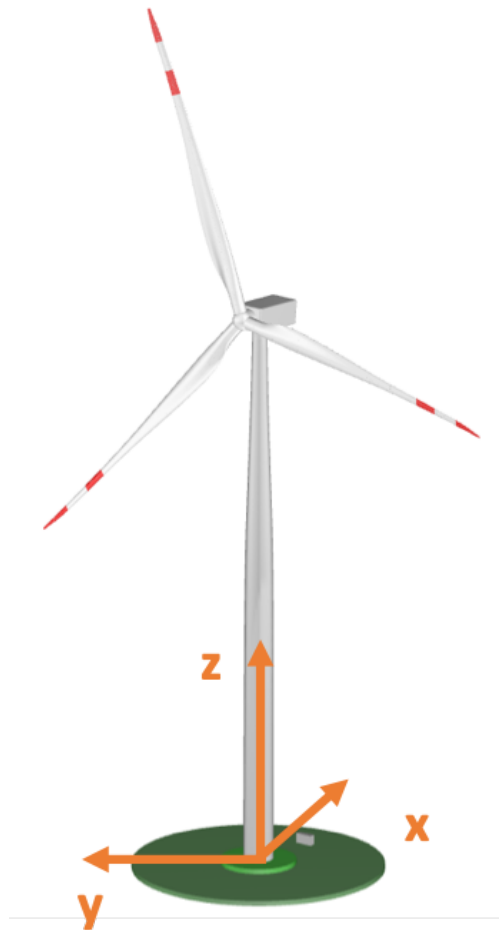


Fig. 4.23: Reference system for the observers

(continued from previous page)

```

3 -200 -200 2
4 -200 +200 2
5 +200 -200 2
6 +200 +200 2

```

Turbulence Grid

When the flag **TICalcMeth** is set equal to 1, the grid of turbulence intensity of the wind TI must be defined by the user. This is done by creating a file called **TIGrid_In.txt**, which mimics a TurbSim output file and contains a grid of turbulence intensity, which is defined as a fraction value. The file defines a grid centered at hub height and oriented with the OpenFAST global inertial frame coordinate system; see Fig. 4.23. A user-defined number of lateral and vertical points equally spaced by a user-defined number of meters must be specified. Note that an average wind speed must be defined to convert the turbulence intensity of the wind to the incident turbulent intensity I_1 . An example file for a 160 (lateral) by 180 (vertical) meters grid looks like the following:

```

1 Average Inflow Wind Speed
2 8.0
3 Total Grid points In Y (lateral), Starts from - radius goes to + radius+
4 4
5 Total Grid points In Z (vertical), Starts from bottom tip (hub-radius)
6 3
7 Grid spacing In Y (lateral)
8 40
9 Grid spacing In Z (vertical)
10 60
11 0.1200 0.1200 0.1200 0.1200
12 0.1100 0.1100 0.1100 0.1100
13 0.1000 0.1000 0.1000 0.1000

```

4.2.4 BeamDyn User Guide and Theory Manual

This document offers a quick reference guide for the BeamDyn software program. It is intended to be used by the general user in combination with other OpenFAST manuals. The manual will be updated as new releases are issued and as needed to provide further information on advancements or modifications to the software. For reference, additional materials such as presentation slides, development plans, and publications can be downloaded from the list below.

- BeamDyn inputs from sectional beam properties

The authors are grateful to the U.S. Department of Energy Wind and Water Power Program and the NREL Laboratory Directed Research and Development (LDRD) program through the grant “High-Fidelity Computational Modeling of Wind-Turbine Structural Dynamics” for supporting the development of this software.

Introduction

BeamDyn is a time-domain structural-dynamics module for slender structures created by the National Renewable Energy Laboratory (NREL) through support from the U.S. Department of Energy Wind and Water Power Program and the NREL Laboratory Directed Research and Development (LDRD) program through the grant “High-Fidelity Computational Modeling of Wind-Turbine Structural Dynamics”, see References [WJSJ15, WS13, WSJJ14, WYS13]. The module has been coupled into the FAST aero-hydro-servo-elastic wind turbine multi-physics engineering tool where it is used to model blade structural dynamics. The BeamDyn module follows the requirements of the FAST modularization framework, see References [Jon13]; [GSJ13, JJ13, SJJ14], couples to FAST version 8, and provides new capabilities for modeling initially curved and twisted composite wind turbine blades undergoing large deformation. BeamDyn can also be driven as a stand-alone code to compute the static and dynamic responses of slender structures (blades or otherwise) under prescribed boundary and applied loading conditions uncoupled from FAST.

The model underlying BeamDyn is the geometrically exact beam theory (GEBT) [Hod06]. GEBT supports full geometric nonlinearity and large deflection, with bending, torsion, shear, and extensional degree-of-freedom (DOFs); anisotropic composite material couplings (using full 6×6 mass and stiffness matrices, including bend-twist coupling); and a reference axis that permits blades that are not straight (supporting built-in curve, sweep, and sectional offsets). The GEBT beam equations are discretized in space with Legendre spectral finite elements (LSFEs). LSFEs are p -type elements that combine the accuracy of global spectral methods with the geometric modeling flexibility of the h -type finite elements (FEs) [Pat84]. For smooth solutions, LSFEs have exponential convergence rates compared to low-order elements that have algebraic convergence [SG03, WS13]. Two spatial numerical integration schemes are implemented for the finite element inner products: reduced Gauss quadrature and trapezoidal-rule integration. Trapezoidal-rule integration is appropriate when a large number of sectional properties are specified along the beam axis, for example, in a long wind turbine blade with material properties that vary dramatically over the length. Time integration of the BeamDyn equations of motion is achieved through the implicit generalized- α solver, with user-specified numerical damping. The combined GEBT-LSFE approach permits users to model a long, flexible, composite wind turbine blade with a single high-order element. Given the theoretical foundation and powerful numerical tools introduced above, BeamDyn can solve the complicated nonlinear composite beam problem in an efficient manner. For example, it was recently shown that a grid-independent dynamic solution of a 50-m composite wind turbine blade and with dozens of cross-section stations could be achieved with a single 7th-order LSFE [WSJJ16].

When coupled with FAST, loads and responses are transferred between BeamDyn, ElastoDyn, ServoDyn, and AeroDyn via the FAST driver program (glue code) to enable aero-elasto-servo interaction at each coupling time step. There is a separate instance of BeamDyn for each blade. At the root node, the inputs to BeamDyn are the six displacements (three translations and three rotations), six velocities, and six accelerations; the root node outputs from BeamDyn are the six reaction loads (three translational forces and three moments). BeamDyn also outputs the blade displacements, velocities, and accelerations along the beam length, which are used by AeroDyn to calculate the local aerodynamic loads (distributed along the length) that are used as inputs for BeamDyn. In addition, BeamDyn can calculate member internal reaction loads, as requested by the user. Please refer to Figure [fig:FlowChart] for the coupled interactions between BeamDyn and other modules in FAST. When coupled to FAST, BeamDyn replaces the more simplified blade structural model of ElastoDyn that is still available as an option, but is only applicable to straight isotropic blades dominated by bending. When uncoupled from FAST, the root motion (boundary condition) and applied loads are specified via a stand-alone BeamDyn driver code.

The BeamDyn input file defines the blade geometry; cross-sectional material mass, stiffness, and damping properties; FE resolution; and other simulation- and output-control parameters. The blade geometry is defined through a curvilinear blade reference axis by a series of key points in three-dimensional (3D) space along with the initial twist angles at these points. Each *member* contains at least three key points for the cubic spline fit implemented in BeamDyn; each member is discretized with a single LSFE with a parameter defining the order of the element. Note that the number of key points defining the member and the order (N) of the LSFE are independent. LSFE nodes, which are located at the $N + 1$ Gauss-Legendre-Lobatto points, are not evenly spaced along the element; node locations are generated by the module based on the mesh information. Blade properties are specified in a non-dimensional coordinate ranging from 0.0 to 1.0 along the blade reference axis and are linearly interpolated between two stations if needed by the spatial integration method. The BeamDyn applied loads can be either distributed loads specified at quadrature points, concentrated loads specified at FE nodes, or a combination of the two. When BeamDyn is coupled to FAST, the blade



Fig. 4.24: Coupled interaction between BeamDyn and FAST

analysis node discretization may be independent between BeamDyn and AeroDyn.

This document is organized as follows. Section *Running BeamDyn* details how to obtain the BeamDyn and FAST software archives and run either the stand-alone version of BeamDyn or BeamDyn coupled to FAST. Section *Input Files* describes the BeamDyn input files. Section *Output Files* discusses the output files generated by BeamDyn. Section *BeamDyn Theory* summarizes the BeamDyn theory. Section *Future Work* outlines potential future work. Example input files are shown in Appendix Section 4.2.4. A summary of available output channels is found in Appendix *BeamDyn List of Output Channels*.

Running BeamDyn

This section discusses how to obtain and execute BeamDyn from a personal computer. Both the stand-alone version and the FAST-coupled version of the software are considered.

Downloading the BeamDyn Software

There are two forms of the BeamDyn software to choose from: stand-alone and coupled to the FAST simulator. Although the user may not necessarily need both forms, he/she would likely need to be familiar with and run the stand-alone model if building a model of the blade from scratch. The stand-alone version is also helpful for model troubleshooting, even if the goal is to conduct aero-hydro-servo-elastic simulations of onshore/offshore wind turbines within FAST.

Stand-Alone BeamDyn Archive

Users can download the stand-alone BeamDyn archive from our Web server at <https://nwtc.nrel.gov/BeamDyn>. The file has a name similar to `BD_v1.00.00a.exe`, but may have a different version number. The user can then download the self-extracting archive (`.exe`) to expand the archive into a folder he/she specifies.

The archive contains the `bin`, `CertTest`, `Compiling`, `Docs`, and `Source` folders. The `bin` folder includes the main executable file, `BeamDyn_Driver.exe`, which is used to execute the stand-alone BeamDyn program. The `CertTest` folder contains a collection of sample BeamDyn input files and driver input files that can be used as templates for the user's own models. This document may be found in the `Docs` folder. The `Compiling` folder contains files for compiling the stand-alone `BeamDyn_v1.00.00.exe` file with either Visual Studio or gFortran. The Fortran source code is located in the `Source` folder.

FAST Archive

Download the FAST archive, which includes BeamDyn, from our Web server at <https://nwtc.nrel.gov/FAST8>. The file has a name similar to `FAST_v8.12.00.exe`, but may have a different version number. Run the downloaded self-extracting archive (`.exe`) to expand the archive into a user-specified folder. The FAST executable file is located in the archive's `bin` folder. An example model using the NREL 5-MW reference turbine is located in the `CertTest` folder.

Running BeamDyn

Running the Stand-Alone BeamDyn Program

The stand-alone BeamDyn program, `BeamDyn_Driver.exe`, simulates static and dynamic responses of the user's input model, without coupling to FAST. Unlike the coupled version, the stand-alone software requires the use of a driver file in addition to the primary and blade BeamDyn input files. This driver file specifies inputs normally provided to BeamDyn by FAST, including motions of the blade root and externally applied loads. Both the BeamDyn summary file and the results output file are available when using the stand-alone BeamDyn (see Section [Output Files](#) for more information regarding the BeamDyn output files).

Run the stand-alone BeamDyn software from a DOS command prompt by typing, for example:

```
>BeamDyn_Driver.exe Dvr_5MW_Dynamic.inp
```

where, `Dvr_5MW_Dynamic.inp` is the name of the BeamDyn driver input file, as described in Section [BeamDyn Driver Input File](#).

Running BeamDyn Coupled to FAST

Run the coupled FAST software from a DOS command prompt by typing, for example:

```
>FAST_Win32.exe Test26.fst
```

where `Test26.fst` is the name of the primary FAST input file. This input file has a feature switch to enable or disable the BeamDyn capabilities within FAST, and a corresponding reference to the BeamDyn input file. See the documentation supplied with FAST for further information.

Input Files

Users specify the blade model parameters; including its geometry, cross-sectional properties, and FE and output control parameters; via a primary BeamDyn input file and a blade property input file. When used in stand-alone mode, an additional driver input file is required. This driver file specifies inputs normally provided to BeamDyn by FAST, including simulation range, root motions, and externally applied loads.

No lines should be added or removed from the input files, except in tables where the number of rows is specified.

Units

BeamDyn uses the SI system (kg, m, s, N). Angles are assumed to be in radians unless otherwise specified.

BeamDyn Driver Input File

The driver input file is needed only for the stand-alone version of BeamDyn. It contains inputs that are normally set by FAST and that are necessary to control the simulation for uncoupled models.

The driver input file begins with two lines of header information, which is for the user but is not used by the software. If BeamDyn is run in the stand-alone mode, the results output file will be prefixed with the same name of this driver input file.

A sample BeamDyn driver input file is given in [Section 4.2.4](#).

Simulation Control Parameters

`DynamicSolve` is a logical variable that specifies if BeamDyn should use dynamic analysis (`DynamicSolve = true`) or static analysis (`DynamicSolve = false`). `t_initial` and `t_final` specify the starting time of the simulation and ending time of the simulation, respectively. `dt` specifies the time step size.

Gravity Parameters

`Gx`, `Gy`, and `Gz` specify the components of gravity vector along X , Y , and Z directions in the global coordinate system, respectively. In FAST, this is normally 0, 0, and -9.80665.

Inertial Frame Parameters

This section defines the relation between two inertial frames, the global coordinate system and initial blade reference coordinate system. `GlbPos(1)`, `GlbPos(2)`, and `GlbPos(3)` specify three components of the initial global position vector along X , Y , and Z directions resolved in the global coordinate system, see Figure Fig. 4.25. And the following 3×3 direction cosine matrix (`GlbDCM`) relates the rotations from the global coordinate system to the initial blade reference coordinate system.

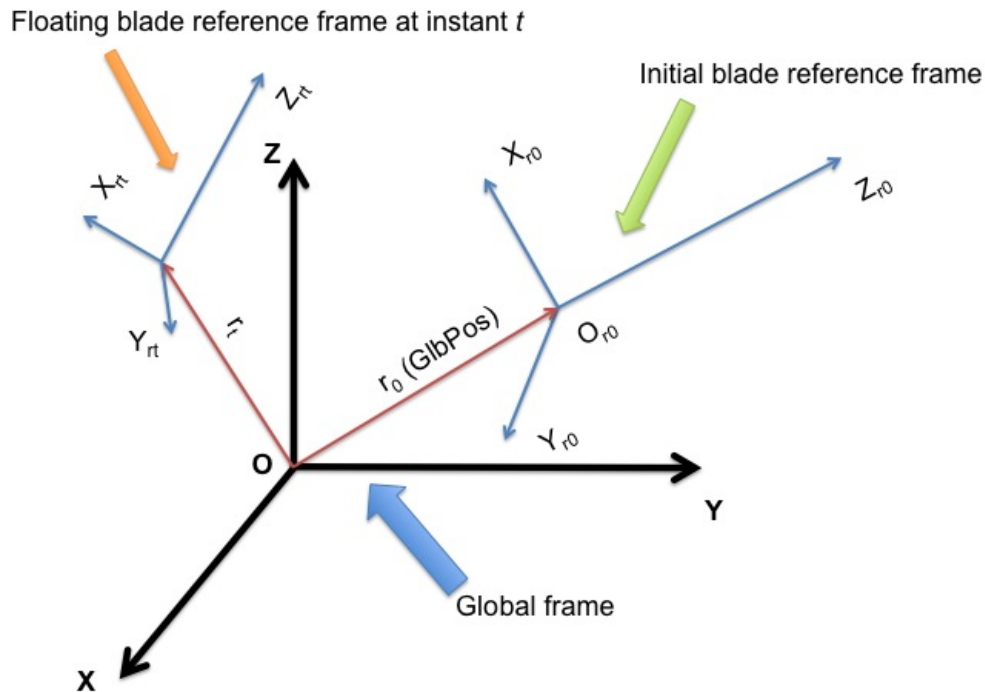


Fig. 4.25: Global and blade coordinate systems in BeamDyn.

Blade Floating Reference Frame Parameters

This section specifies the parameters that define the blade floating reference frame, which is a body-attached floating frame; the blade root is cantilevered at the origin of this frame. Based on the driver input file, the floating blade reference frame is assumed to be in a constant rigid-body rotation mode about the origin of the global coordinate system, that is,

$$v_{rt} = \omega_r \times r_t \quad (4.57)$$

where v_{rt} is the root (origin of the floating blade reference frame) translational velocity vector; ω_r is the constant root (origin of the floating blade reference frame) angular velocity vector; and r_t is the global position vector introduced in the previous section at instant t , see Fig. 4.25. The floating blade reference frame coincides with the initial floating blade reference frame at the beginning $t = 0$. `RootVel(4)`, `RootVel(5)`, and `RootVel(6)` specify the three components of the constant root angular velocity vector about X , Y , and Z axes in global coordinate system, respectively. `RootVel(1)`, `RootVel(2)`, and `RootVel(3)`, which are the three components of the root translational velocity vector along X , Y , and Z directions in global coordinate system, respectively, are calculated based on Eq. (4.57).

BeamDyn can handle more complicated root motions by changing, for example, the `BD_InputSolve` subroutine in the `Driver_Beam.f90` (requiring a recompile of stand-alone BeamDyn).

The blade is initialized in the rigid-body motion mode, i.e., based on the root velocity information defined in this section and the position information defined in the previous section, the motion of other points along the blade are initialized as

$$\begin{aligned} a_0 &= \omega_r \times (\omega_r \times (r_0 + P)) \\ v_0 &= v_{r0} + \omega_r \times P \\ \omega_0 &= \omega_r \end{aligned} \quad (4.58)$$

where a_0 is the initial translational acceleration vector along the blade; v_0 and ω_0 the initial translational and angular velocity vectors along the blade, respectively; and P is the position vector along the blade relative to the root. Note that these equations are actually implemented with a call to the NWTC Library's mesh mapping routines.

Applied Load

This section defines the applied loads, including distributed, point (lumped), and tip-concentrated loads, for the stand-alone analysis.

The first six entries `DistrLoad(i)`, $i \in [1, 6]$, specify three components of uniformly distributed force vector and three components of uniformly distributed moment vector in the global coordinate systems, respectively.

The following six entries `TipLoad(i)`, $i \in [1, 6]$, specify three components of concentrated tip force vector and three components of concentrated tip moment vector in the global coordinate system, respectively.

`NumPointLoads` defines how many point loads along the blade will be applied. The table following this input contains two header lines with seven columns and `NumPointLoads` rows. The first column is the non-dimensional distance along the local blade reference axis, ranging from $[0.0, 1.0]$. The next three columns, `Fx`, `Fy`, and `Fz` specify three components of point-force vector. The remaining three columns, `Mx`, `My`, and `Mz` specify three components of a moment vector.

The distributed load defined in this section is assumed to be uniform along the blade and constant throughout the simulation. The tip load is a constant concentrated load applied at the tip of a blade.

It is noted that all the loads defined in this section are dead loads, i.e., they are not rotating with the blade following the rigid-body rotation defined in the previous section.

BeamDyn is capable of handling more complex loading cases, e.g., time-dependent loads, through customization of the source code (requiring a recompile of stand-alone BeamDyn). The user can define such loads in the `BD_InputSolve` subroutine in the `Driver_Beam.f90` file, which is called every time step. The following section can be modified to define the concentrated load at each FE node:

```

u%PointLoad%Force(1:3,u%PointLoad%NNodes) = u%PointLoad%Force(1:3,u%PointLoad%NNodes)
↪+ DvrData%TipLoad(1:3)
u%PointLoad%Moment(1:3,u%PointLoad%NNodes) = u%PointLoad%Moment(1:3,u%PointLoad%NNodes)
↪+ DvrData%TipLoad(4:6)

```

where the first index in each array ranges from 1 to 3 for load vector components along three global directions and the second index of each array ranges from 1 to `u%PointLoad%NNodes`, where the latter is the total number of FE nodes. Note that `u%PointLoad%Force(1:3,:)` and `u%PointLoad%Moment(1:3,:)` have been populated with the point-load loads read from the BeamDyn driver input file using the call to `Transfer_Point_to_Point` earlier in the subroutine.

For example, a time-dependent sinusoidal force acting along the X direction applied at the 2^{nd} FE node can be defined as

```

u%PointLoad%Force(:, :) = 0.0D0
u%PointLoad%Force(1,2) = 1.0D+03*SIN((2.0*pi)*t/6.0 )
u%PointLoad%Moment(:, :) = 0.0D0

```

with `1.0D+03` being the amplitude and `6.0` being the period. Note that this particular implementation overrides the tip-load and point-loads defined in the driver input file.

Similar to the concentrated load, the distributed loads can be defined in the same subroutine

```

DO i=1,u%DistrLoad%NNodes
  u%DistrLoad%Force(1:3,i) = DvrData%DistrLoad(1:3)
  u%DistrLoad%Moment(1:3,i)= DvrData%DistrLoad(4:6)
ENDDO

```

where `u%DistrLoad%NNodes` is the number of nodes input to BeamDyn (on the quadrature points), and `DvrData%DistrLoad(:)` is the constant uniformly distributed load BeamDyn reads from the driver input file. The user can modify `DvrData%DistrLoad(:)` to define the loads based on need.

We note that the distributed loads are defined at the quadrature points for numerical integrations. For example, if Gauss quadrature is chosen, then the distributed loads are defined at Gauss points plus the two end points of the beam (root and tip). For trapezoidal quadrature, `p%ngp` stores the number of trapezoidal quadrature points.

Primary Input File

`InputFile` is the file name of the primary BeamDyn input file. This name should be in quotations and can contain an absolute path or a relative path.

BeamDyn Primary Input File

The BeamDyn primary input file defines the blade geometry, LSFE-discretization and simulation options, output channels, and name of the blade input file. The geometry of the blade is defined by key-point coordinates and initial twist angles (in units of degree) in the blade local coordinate system (IEC standard blade system where Z_r is along blade axis from root to tip, X_r directs normally toward the suction side, and Y_r directs normally toward the trailing edge).

The file is organized into several functional sections. Each section corresponds to an aspect of the BeamDyn model.

A sample BeamDyn primary input file is given in [Section 4.2.4](#).

The primary input file begins with two lines of header information, which are for the user but are not used by the software.

Simulation Controls

The user can set the `Echo` flag to `TRUE` to have BeamDyn echo the contents of the BeamDyn input file (useful for debugging errors in the input file).

The `QuasiStaticInit` flag indicates if BeamDyn should perform a quasi-static solution at initialization to better initialize its states. In general, this should be set to true for better numerical performance (it reduces startup transients).

`rhoInf` specifies the numerical damping parameter (spectral radius of the amplification matrix) in the range of $[0.0, 1.0]$ used in the generalized- α time integrator implemented in BeamDyn for dynamic analysis. For `rhoInf` = 1.0, no numerical damping is introduced and the generalized- α scheme is identical to the Newmark scheme; for `rhoInf` = 0.0, maximum numerical damping is introduced. Numerical damping may help produce numerically stable solutions.

`Quadrature` specifies the spatial numerical integration scheme. There are two options: 1) Gauss quadrature; and 2) Trapezoidal quadrature. We note that in the current version, Gauss quadrature is implemented in reduced form to improve efficiency and avoid shear locking. In the trapezoidal quadrature, only one member (FE element) can be defined in the following `GEOMETRY` section of the primary input file. Trapezoidal quadrature is appropriate when the number of “blade input stations” (described below) is significantly greater than the order of the LSFE.

`Refine` specifies a refinement parameter used in trapezoidal quadrature. An integer value greater than unity will split the space between two input stations into “Refine factor” of segments. The keyword “DEFAULT” may be used to set it to 1, i.e., no refinement is needed. This entry is not used in Gauss quadrature.

`N_Fact` specifies a parameter used in the modified Newton-Raphson scheme. If `N_Fact` = 1 a full Newton iteration scheme is used, i.e., the global tangent stiffness matrix is computed and factorized at each iteration; if `N_Fact` > 1 a modified Newton iteration scheme is used, i.e., the global stiffness matrix is computed and factorized every `N_Fact` iterations within each time step. The keyword “DEFAULT” sets `N_Fact` = 5.

`DTBeam` specifies the constant time increment of the time-integration in seconds. The keyword “DEFAULT” may be used to indicate that the module should employ the time increment prescribed by the driver code (FAST/stand-alone driver program).

`load_retries` specifies the maximum number of load retries allowed. This option currently works only for static analysis. For every load retry, the applied load is halved to promote convergence of the Newton-Raphson scheme in iteration of smaller load steps as opposed to one single large load step which may cause divergence of the Newton-Raphson scheme. The keyword “DEFAULT” sets `load_retries` = 20.

`NRMax` specifies the maximum number of iterations per time step in the Newton-Raphson scheme. If convergence is not reached within this number of iterations, BeamDyn returns an error message and terminates the simulation. The keyword “DEFAULT” sets `NRMax` = 10.

`Stop_Tol` specifies a tolerance parameter used in convergence criteria of a nonlinear solution that is used for the termination of the iteration. The keyword “DEFAULT” sets `Stop_Tol` = 1.0E-05. Please refer to [Section 4.2.4](#) for more details.

`tngt_stf_fd` is a boolean that sets the flag to compute the tangent stiffness matrix using finite differencing instead of analytical differentiation. The finite differencing is performed using a central scheme. The keyword “DEFAULT” sets `tngt_stf_fd` = FALSE.

`tngt_stf_comp` is a boolean that sets the flag to compare the analytical tangent stiffness matrix against the finite differenced tangent stiffness matrix. Information is written to the terminal regarding the dof where the maximum difference is observed. If `tngt_stf_fd` = FALSE and `tngt_stf_comp` = TRUE, the analytical tangent stiffness matrix is used to solve the system of equations while the finite difference tangent stiffness matrix is used only to perform the comparison of the two matrices. The keyword “DEFAULT” sets `tngt_stf_comp` = FALSE.

`tngt_stf_pert` sets the perturbation size for finite differencing. The “DEFAULT” value based on experience is set to 1e-06.

`tngt_stf_diff_tol` is the maximum allowable relative difference between the analytical and finite differenced tangent stiffness matrices. If for any entry in the matrices, the relative difference exceeds this value the simulation will terminate.

The “DEFAULT” value is currently set to 1e-01.

RotStates is a flag that indicates if BeamDyn’s continuous states should be oriented in the rotating frame during linearization analysis when coupled to OpenFAST. If multi-blade coordinate (MBC3) analysis is performed, RotStates must be true.

Geometry Parameter

The blade geometry is defined by a curvilinear local blade reference axis. The blade reference axis locates the origin and orientation of each a local coordinate system where the cross-sectional 6x6 stiffness and mass matrices are defined in the BeamDyn blade input file. It should not really matter where in the cross section the 6x6 stiffness and mass matrices are defined relative to, as long as the reference axis is consistently defined and closely follows the natural geometry of the blade.

The blade beam model is composed of several *members* in contiguous series and each member is defined by at least three key points in BeamDyn. A cubic-spline-fit pre-processor implemented in BeamDyn automatically generates the member based on the key points and then interconnects the members into a blade. There is always a shared key point at adjacent members; therefore the total number of key points is related to number of members and key points in each member.

member_total specifies the total number of beam members used in the structure. With the LSFEE discretization, a single member and a sufficiently high element order, order_elem below, may well be sufficient.

kp_total specifies the total number of key points used to define the beam members.

The following section contains member_total lines. Each line has two integers providing the member number (must be 1, 2, 3, etc., sequentially) and the number of key points in this member, respectively. It is noted that the number of key points in each member is not independent of the total number of key points and they should satisfy the following equality:

$$kp_total = \sum_{i=1}^{member_total} n_i - member_total + 1 \quad (4.59)$$

where n_i is the number of key points in the i^{th} member. Because cubic splines are implemented in BeamDyn, n_i must be greater than or equal to three. Figures Fig. 4.26 and Fig. 4.27 show two cases for member and key-point definition.

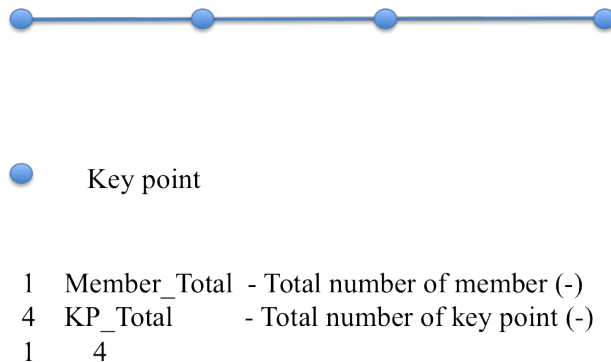


Fig. 4.26: Member and key point definition: one member defined by four key points;

The next section defines the key-point information, preceded by two header lines. Each key point is defined by three physical coordinates (kp_xr, kp_yr, kp_zr) in the IEC standard blade coordinate system (the blade reference coordinate system) along with a structural twist angle (initial_twist) in the unit of degrees. The structural twist angle is also following the IEC standard which is defined as the twist about the negative Z_l axis. The key points are entered sequentially (from the root to tip) and there should be a total of kp_total lines for BeamDyn to read in the information, after two header lines. Please refer to Fig. 4.28 for more details on the blade geometry definition.



● Key point

2	Member_Total	- Total number of member (-)
6	KP_Total	- Total number of key point (-)
1	4	
2	3	

Fig. 4.27: Member and key point definition: two members defined by six key points.

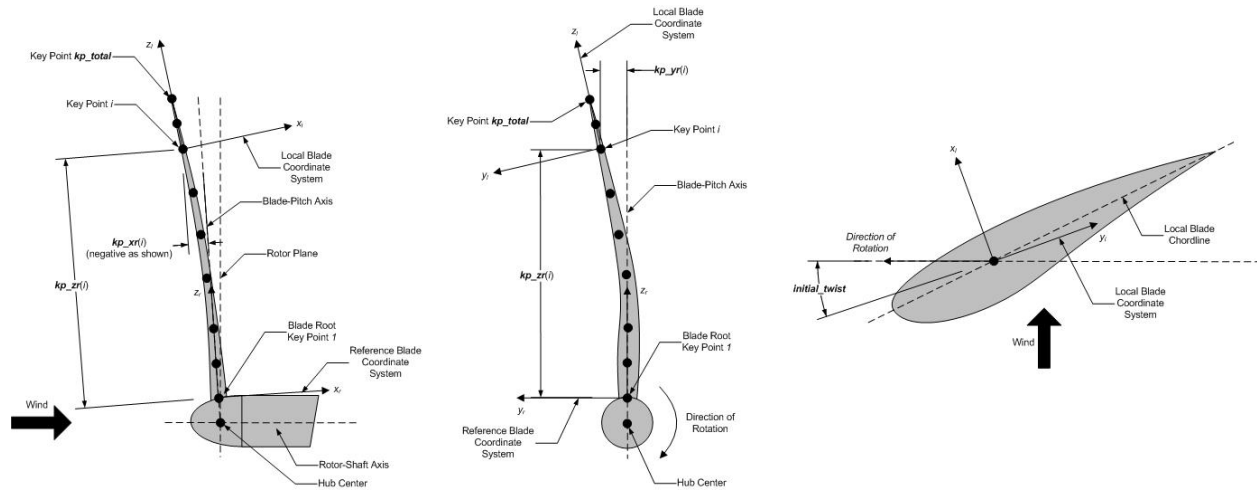


Fig. 4.28: BeamDyn Blade Geometry - Top: Side View; Middle: Front View (Looking Downwind); Bottom: Cross Section View (Looking Toward the Tip, from the Root)

Mesh Parameter

`Order_Elem` specifies the order of shape functions for each finite element. Each LSFE will have `Order_Elem+1` nodes located at the GLL quadrature points. All LSFEs will have the same order. With the LSFE discretization, an increase in accuracy will, in general, be better achieved by increasing `Order_Elem` (i.e., p -refinement) rather than increasing the number of members (i.e., h -refinement). For Gauss quadrature, `Order_Elem` should be greater than one.

Material Parameter

`BldFile` is the file name of the blade input file. This name should be in quotations and can contain an absolute path or a relative path.

Pitch Actuator Parameter

In this release, the pitch actuator implemented in BeamDyn is not available. The `UsePitchAct` should be set to “FALSE” in this version, whereby the input blade-pitch angle prescribed by the driver code is used to orient the blade directly. `PitchJ`, `PitchK`, and `PitchC` specify the pitch actuator inertial, stiffness, and damping coefficient, respectively. In future releases, specifying `UsePitchAct = TRUE` will enable a second-order pitch actuator, whereby the pitch angular orientation, velocity, and acceleration are determined by the actuator based on the input blade-pitch angle prescribed by the driver code.

Outputs

In this section of the primary input file, the user sets flags and switches for the desired output behavior.

Specifying `SumPrint = TRUE` causes BeamDyn to generate a summary file with name `InputFile.sum`. See [Section 4.2.4](#) for summary file details.

`OutFmt` parameter controls the formatting of the results within the stand-alone BeamDyn output file. It needs to be a valid Fortran format string, but BeamDyn currently does not check the validity. This input is unused when BeamDyn is used coupled to FAST.

`NNodeOuts` specifies the number of nodes where output can be written to a file. Currently, BeamDyn can output quantities at a maximum of nine nodes.

`OutNd` is a list `NNodeOuts` long of node numbers between 1 and the number of nodes on the output mesh, separated by any combination of commas, semicolons, spaces, and/or tabs. The nodal positions are given in the summary file, if output. For Gaussian quadrature, the number of nodes on the output mesh is the total number of FE nodes; for trapezoidal quadrature, this is the number of quadrature nodes.

The `OutList` block contains a list of output parameters. Enter one or more lines containing quoted strings that in turn contain one or more output parameter names. Separate output parameter names by any combination of commas, semicolons, spaces, and/or tabs. If you prefix a parameter name with a minus sign, “-”, underscore, “_”, or the characters “m” or “M”, BeamDyn will multiply the value for that channel by -1 before writing the data. The parameters are written in the order they are listed in the input file. BeamDyn allows you to use multiple lines so that you can break your list into meaningful groups and so the lines can be shorter. You may enter comments after the closing quote on any of the lines. Entering a line with the string “END” at the beginning of the line or at the beginning of a quoted string found at the beginning of the line will cause BeamDyn to quit scanning for more lines of channel names. Node-related quantities are generated for the requested nodes identified through the `OutNd` list above. If BeamDyn encounters an unknown/invalid channel name, it warns the users but will remove the suspect channel from the output file. Please refer to [Appendix Section 4.2.4](#) for a complete list of possible output parameters and their names.

Nodal Outputs

In addition to the named outputs in [Section 4.2.4](#) above, BeamDyn allows for outputting the full set blade node motions and loads (tower nodes unavailable at present). Please refer to the BeamDyn_Nodes tab in the Excel file `OutListParameters.xlsx` for a complete list of possible output parameters.

This section follows the *END* statement from normal Outputs section described above, and includes a separator description line followed by the following options.

BldNd_BlOutNd specifies which nodes to output. This is currently unused.

The **OutList** section controls the nodal output quantities generated by BeamDyn. In this section, the user specifies the name of the channel family to output. The output name for each channel is then created internally by BeamDyn by combining the blade number, node number, and channel family name. For example, if the user specifies **TDxr** as the channel family name, the output channels will be named with the convention of **B β N###TDxr** where β is the blade number, and ### is the three digit node number.

Sample Nodal Outputs section

This sample includes the *END* statement from the regular outputs section.

```

1  END of input file (the word "END" must appear in the first 3 columns of this last
   ↳OutList line)
2  ----- NODE OUTPUTS -----
3      99  BldNd_BlOutNd  - Blade nodes on each blade (currently unused)
4      OutList  - The next line(s) contains a list of output parameters.  See
   ↳OutListParameters.xlsx, BeamDyn_Nodes tab for a listing of available output channels,
   ↳(-)
5  "FxL"      - Sectional force resultants at each node expressed in l      l: a floating
   ↳coordinate system local to the deflected beam      (N)
6  "FyL"      - Sectional force resultants at each node expressed in l      l: a floating
   ↳coordinate system local to the deflected beam      (N)
7  "FzL"      - Sectional force resultants at each node expressed in l      l: a floating
   ↳coordinate system local to the deflected beam      (N)
8  "MxL"      - Sectional moment resultants at each node expressed in l      l: a floating
   ↳coordinate system local to the deflected beam      (N-m)
9  "MyL"      - Sectional moment resultants at each node expressed in l      l: a floating
   ↳coordinate system local to the deflected beam      (N-m)
10 "MzL"      - Sectional moment resultants at each node expressed in l      l: a floating
   ↳coordinate system local to the deflected beam      (N-m)
11 "Fxr"      - Sectional force resultants at each node expressed in r      r: a floating
   ↳reference coordinate system fixed to the root of the moving beam; when coupled to FAST
   ↳for blades, this is equivalent to the IEC blade (b) coordinate system      (N)
12 "Fyr"      - Sectional force resultants at each node expressed in r      r: a floating
   ↳reference coordinate system fixed to the root of the moving beam; when coupled to FAST
   ↳for blades, this is equivalent to the IEC blade (b) coordinate system      (N)
13 "Fzr"      - Sectional force resultants at each node expressed in r      r: a floating
   ↳reference coordinate system fixed to the root of the moving beam; when coupled to FAST
   ↳for blades, this is equivalent to the IEC blade (b) coordinate system      (N)
14 "Mxr"      - Sectional moment resultants at each node expressed in r      r: a floating
   ↳reference coordinate system fixed to the root of the moving beam; when coupled to FAST
   ↳for blades, this is equivalent to the IEC blade (b) coordinate system      (N-m)
15 "Myr"      - Sectional moment resultants at each node expressed in r      r: a floating
   ↳reference coordinate system fixed to the root of the moving beam; when coupled to FAST

```

(continues on next page)

(continued from previous page)

↳for blades, this is equivalent to the IEC blade (b) coordinate system (N-m)
 16 "Mzr" - Sectional moment resultants at each node expressed in r r: a floating_
 ↳reference coordinate system fixed to the root of the moving beam; when coupled to FAST_
 ↳for blades, this is equivalent to the IEC blade (b) coordinate system (N-m)
 17 "TDxr" - Sectional translational deflection (relative to the undeflected position)_
 ↳at each node expressed in r r: a floating reference coordinate system fixed to the_
 ↳root of the moving beam; when coupled to FAST for blades, this is equivalent to the_
 ↳IEC blade (b) coordinate system (m)
 18 "TDyr" - Sectional translational deflection (relative to the undeflected position)_
 ↳at each node expressed in r r: a floating reference coordinate system fixed to the_
 ↳root of the moving beam; when coupled to FAST for blades, this is equivalent to the_
 ↳IEC blade (b) coordinate system (m)
 19 "TDzr" - Sectional translational deflection (relative to the undeflected position)_
 ↳at each node expressed in r r: a floating reference coordinate system fixed to the_
 ↳root of the moving beam; when coupled to FAST for blades, this is equivalent to the_
 ↳IEC blade (b) coordinate system (m)
 20 "RDxr" - Sectional angular/rotational deflection Wiener-Milenkovic parameter_
 ↳(relative to the undeflected orientation) at each node expressed in r r: a floating_
 ↳reference coordinate system fixed to the root of the moving beam; when coupled to FAST_
 ↳for blades, this is equivalent to the IEC blade (b) coordinate system (-)
 21 "RDyr" - Sectional angular/rotational deflection Wiener-Milenkovic parameter_
 ↳(relative to the undeflected orientation) at each node expressed in r r: a floating_
 ↳reference coordinate system fixed to the root of the moving beam; when coupled to FAST_
 ↳for blades, this is equivalent to the IEC blade (b) coordinate system (-)
 22 "RDzr" - Sectional angular/rotational deflection Wiener-Milenkovic parameter_
 ↳(relative to the undeflected orientation) at each node expressed in r r: a floating_
 ↳reference coordinate system fixed to the root of the moving beam; when coupled to FAST_
 ↳for blades, this is equivalent to the IEC blade (b) coordinate system (-)
 23 "AbsXg" - Node position in X (global coordinate) g: the global inertial frame_
 ↳coordinate system; when coupled to FAST, this is equivalent to FAST's global inertial_
 ↳frame (i) coordinate system (m)
 24 "AbsYg" - Node position in Y (global coordinate) g: the global inertial frame_
 ↳coordinate system; when coupled to FAST, this is equivalent to FAST's global inertial_
 ↳frame (i) coordinate system (m)
 25 "AbsZg" - Node position in Z (global coordinate) g: the global inertial frame_
 ↳coordinate system; when coupled to FAST, this is equivalent to FAST's global inertial_
 ↳frame (i) coordinate system (m)
 26 "AbsXr" - Node position in X (relative to root) r: a floating reference_
 ↳coordinate system fixed to the root of the moving beam; when coupled to FAST for_
 ↳blades, this is equivalent to the IEC blade (b) coordinate system (m)
 27 "AbsYr" - Node position in Y (relative to root) r: a floating reference_
 ↳coordinate system fixed to the root of the moving beam; when coupled to FAST for_
 ↳blades, this is equivalent to the IEC blade (b) coordinate system (m)
 28 "AbsZr" - Node position in Z (relative to root) r: a floating reference_
 ↳coordinate system fixed to the root of the moving beam; when coupled to FAST for_
 ↳blades, this is equivalent to the IEC blade (b) coordinate system (m)
 29 "TVxg" - Sectional translational velocities (absolute) g: the global inertial_
 ↳frame coordinate system; when coupled to FAST, this is equivalent to FAST's global_
 ↳inertial frame (i) coordinate system (m/s)
 30 "TVyg" - Sectional translational velocities (absolute) g: the global inertial_
 ↳frame coordinate system; when coupled to FAST, this is equivalent to FAST's global_
 ↳inertial frame (i) coordinate system (m/s)

(continues on next page)

(continued from previous page)

```

31 "TVzg"      - Sectional translational velocities (absolute)    g: the global inertial_
    ↳ frame coordinate system; when coupled to FAST, this is equivalent to FAST's global_
    ↳ inertial frame (i) coordinate system      (m/s)
32 "TVxl"      - Sectional translational velocities (absolute)    l: a floating coordinate_
    ↳ system local to the deflected beam      (m/s)
33 "TVyl"      - Sectional translational velocities (absolute)    l: a floating coordinate_
    ↳ system local to the deflected beam      (m/s)
34 "TVzl"      - Sectional translational velocities (absolute)    l: a floating coordinate_
    ↳ system local to the deflected beam      (m/s)
35 "TVxr"      - Sectional translational velocities (absolute)    r: a floating reference_
    ↳ coordinate system fixed to the root of the moving beam; when coupled to FAST for_
    ↳ blades, this is equivalent to the IEC blade (b) coordinate system      (m/s)
36 "TVyr"      - Sectional translational velocities (absolute)    r: a floating reference_
    ↳ coordinate system fixed to the root of the moving beam; when coupled to FAST for_
    ↳ blades, this is equivalent to the IEC blade (b) coordinate system      (m/s)
37 "TVzr"      - Sectional translational velocities (absolute)    r: a floating reference_
    ↳ coordinate system fixed to the root of the moving beam; when coupled to FAST for_
    ↳ blades, this is equivalent to the IEC blade (b) coordinate system      (m/s)
38 "RVxg"      - Sectional angular/rotational velocities (absolute) g: the global_
    ↳ inertial frame coordinate system; when coupled to FAST, this is equivalent to FAST's_
    ↳ global inertial frame (i) coordinate system      (deg/s)
39 "RVyg"      - Sectional angular/rotational velocities (absolute) g: the global_
    ↳ inertial frame coordinate system; when coupled to FAST, this is equivalent to FAST's_
    ↳ global inertial frame (i) coordinate system      (deg/s)
40 "RVzg"      - Sectional angular/rotational velocities (absolute) g: the global_
    ↳ inertial frame coordinate system; when coupled to FAST, this is equivalent to FAST's_
    ↳ global inertial frame (i) coordinate system      (deg/s)
41 "RVxl"      - Sectional angular/rotational velocities (absolute) l: a floating_
    ↳ coordinate system local to the deflected beam      (deg/s)
42 "RVyl"      - Sectional angular/rotational velocities (absolute) l: a floating_
    ↳ coordinate system local to the deflected beam      (deg/s)
43 "RVzl"      - Sectional angular/rotational velocities (absolute) l: a floating_
    ↳ coordinate system local to the deflected beam      (deg/s)
44 "RVxr"      - Sectional angular/rotational velocities (absolute) r: a floating_
    ↳ reference coordinate system fixed to the root of the moving beam; when coupled to FAST_
    ↳ for blades, this is equivalent to the IEC blade (b) coordinate system      (deg/s)
45 "RVyr"      - Sectional angular/rotational velocities (absolute) r: a floating_
    ↳ reference coordinate system fixed to the root of the moving beam; when coupled to FAST_
    ↳ for blades, this is equivalent to the IEC blade (b) coordinate system      (deg/s)
46 "RVzr"      - Sectional angular/rotational velocities (absolute) r: a floating_
    ↳ reference coordinate system fixed to the root of the moving beam; when coupled to FAST_
    ↳ for blades, this is equivalent to the IEC blade (b) coordinate system      (deg/s)
47 "TAxl"      - Sectional angular/rotational velocities (absolute) l: a floating_
    ↳ coordinate system local to the deflected beam      (m/s^2)
48 "TAyl"      - Sectional angular/rotational velocities (absolute) l: a floating_
    ↳ coordinate system local to the deflected beam      (m/s^2)
49 "TAzl"      - Sectional angular/rotational velocities (absolute) l: a floating_
    ↳ coordinate system local to the deflected beam      (m/s^2)
50 "TAxr"      - Sectional angular/rotational velocities (absolute) r: a floating_
    ↳ reference coordinate system fixed to the root of the moving beam; when coupled to FAST_
    ↳ for blades, this is equivalent to the IEC blade (b) coordinate system      (m/s^2)
51 "TAyr"      - Sectional angular/rotational velocities (absolute) r: a floating_

```

(continues on next page)

(continued from previous page)

```

↪reference coordinate system fixed to the root of the moving beam; when coupled to FAST_
↪for blades, this is equivalent to the IEC blade (b) coordinate system      (m/s^2)
52 "TAzr"      - Sectional angular/rotational velocities (absolute)    r: a floating_
↪reference coordinate system fixed to the root of the moving beam; when coupled to FAST_
↪for blades, this is equivalent to the IEC blade (b) coordinate system      (m/s^2)
53 "RAx1"      - Sectional angular/rotational velocities (absolute)    l: a floating_
↪coordinate system local to the deflected beam      (deg/s^2)
54 "RAy1"      - Sectional angular/rotational velocities (absolute)    l: a floating_
↪coordinate system local to the deflected beam      (deg/s^2)
55 "RAz1"      - Sectional angular/rotational velocities (absolute)    l: a floating_
↪coordinate system local to the deflected beam      (deg/s^2)
56 "RAxr"      - Sectional angular/rotational velocities (absolute)    r: a floating_
↪reference coordinate system fixed to the root of the moving beam; when coupled to FAST_
↪for blades, this is equivalent to the IEC blade (b) coordinate system      (deg/s^2)
57 "RAyr"      - Sectional angular/rotational velocities (absolute)    r: a floating_
↪reference coordinate system fixed to the root of the moving beam; when coupled to FAST_
↪for blades, this is equivalent to the IEC blade (b) coordinate system      (deg/s^2)
58 "RAzr"      - Sectional angular/rotational velocities (absolute)    r: a floating_
↪reference coordinate system fixed to the root of the moving beam; when coupled to FAST_
↪for blades, this is equivalent to the IEC blade (b) coordinate system      (deg/s^2)
59 "PFxL"      - Applied point forces at each node expressed in l      l: a floating_
↪coordinate system local to the deflected beam      (N)
60 "PFyL"      - Applied point forces at each node expressed in l      l: a floating_
↪coordinate system local to the deflected beam      (N)
61 "PFzL"      - Applied point forces at each node expressed in l      l: a floating_
↪coordinate system local to the deflected beam      (N)
62 "PMxL"      - Applied point moments at each node expressed in l      l: a floating_
↪coordinate system local to the deflected beam      (N-m)
63 "PMyL"      - Applied point moments at each node expressed in l      l: a floating_
↪coordinate system local to the deflected beam      (N-m)
64 "PMzL"      - Applied point moments at each node expressed in l      l: a floating_
↪coordinate system local to the deflected beam      (N-m)
65 "DFxL"      - Applied distributed forces at each node expressed in l  l: a floating_
↪coordinate system local to the deflected beam      (N/m)
66 "DFyL"      - Applied distributed forces at each node expressed in l  l: a floating_
↪coordinate system local to the deflected beam      (N/m)
67 "DFzL"      - Applied distributed forces at each node expressed in l  l: a floating_
↪coordinate system local to the deflected beam      (N/m)
68 "DMxL"      - Applied distributed moments at each node expressed in l  l: a floating_
↪coordinate system local to the deflected beam      (N-m/m)
69 "DMyL"      - Applied distributed moments at each node expressed in l  l: a floating_
↪coordinate system local to the deflected beam      (N-m/m)
70 "DMzL"      - Applied distributed moments at each node expressed in l  l: a floating_
↪coordinate system local to the deflected beam      (N-m/m)
71 "DFxR"      - Applied distributed forces at each node expressed in r  r: a floating_
↪reference coordinate system fixed to the root of the moving beam; when coupled to FAST_
↪for blades, this is equivalent to the IEC blade (b) coordinate system      (N/m)
72 "DFyR"      - Applied distributed forces at each node expressed in r  r: a floating_
↪reference coordinate system fixed to the root of the moving beam; when coupled to FAST_
↪for blades, this is equivalent to the IEC blade (b) coordinate system      (N/m)
73 "DFzR"      - Applied distributed forces at each node expressed in r  r: a floating_
↪reference coordinate system fixed to the root of the moving beam; when coupled to FAST_

```

(continues on next page)

(continued from previous page)

↪for blades, this is equivalent to the IEC blade (b) coordinate system (N/m)
 74 "DMxR" - Applied distributed forces at each node expressed in r r: a floating_
 ↪reference coordinate system fixed to the root of the moving beam; when coupled to FAST_
 ↪for blades, this is equivalent to the IEC blade (b) coordinate system (N-m/m)
 75 "DMyR" - Applied distributed forces at each node expressed in r r: a floating_
 ↪reference coordinate system fixed to the root of the moving beam; when coupled to FAST_
 ↪for blades, this is equivalent to the IEC blade (b) coordinate system (N-m/m)
 76 "DMzR" - Applied distributed forces at each node expressed in r r: a floating_
 ↪reference coordinate system fixed to the root of the moving beam; when coupled to FAST_
 ↪for blades, this is equivalent to the IEC blade (b) coordinate system (N-m/m)
 77 "FFbx1" - Gyroscopic force x l: a floating coordinate system local to the_
 ↪deflected beam (N)
 78 "FFby1" - Gyroscopic force y l: a floating coordinate system local to the_
 ↪deflected beam (N)
 79 "FFbz1" - Gyroscopic force z l: a floating coordinate system local to the_
 ↪deflected beam (N)
 80 "FFbxr" - Gyroscopic force x r: a floating reference coordinate system fixed to_
 ↪the root of the moving beam; when coupled to FAST for blades, this is equivalent to_
 ↪the IEC blade (b) coordinate system (N)
 81 "FFbyr" - Gyroscopic force y r: a floating reference coordinate system fixed to_
 ↪the root of the moving beam; when coupled to FAST for blades, this is equivalent to_
 ↪the IEC blade (b) coordinate system (N)
 82 "FFb zr" - Gyroscopic force z r: a floating reference coordinate system fixed to_
 ↪the root of the moving beam; when coupled to FAST for blades, this is equivalent to_
 ↪the IEC blade (b) coordinate system (N)
 83 "MFbx1" - Gyroscopic moment about x l: a floating coordinate system local to the_
 ↪deflected beam (N-m)
 84 "MFby1" - Gyroscopic moment about y l: a floating coordinate system local to the_
 ↪deflected beam (N-m)
 85 "MFbz1" - Gyroscopic moment about z l: a floating coordinate system local to the_
 ↪deflected beam (N-m)
 86 "MFbxr" - Gyroscopic moment about x r: a floating reference coordinate system_
 ↪fixed to the root of the moving beam; when coupled to FAST for blades, this is_
 ↪equivalent to the IEC blade (b) coordinate system (N-m)
 87 "MFbyr" - Gyroscopic moment about y r: a floating reference coordinate system_
 ↪fixed to the root of the moving beam; when coupled to FAST for blades, this is_
 ↪equivalent to the IEC blade (b) coordinate system (N-m)
 88 "MFb zr" - Gyroscopic moment about z r: a floating reference coordinate system_
 ↪fixed to the root of the moving beam; when coupled to FAST for blades, this is_
 ↪equivalent to the IEC blade (b) coordinate system (N-m)
 89 "FFcx1" - Elastic restoring force Fc x l: a floating coordinate system local to_
 ↪the deflected beam (N)
 90 "FFcy1" - Elastic restoring force Fc y l: a floating coordinate system local to_
 ↪the deflected beam (N)
 91 "FFcz1" - Elastic restoring force Fc z l: a floating coordinate system local to_
 ↪the deflected beam (N)
 92 "FFcxr" - Elastic restoring force Fc x r: a floating reference coordinate system_
 ↪fixed to the root of the moving beam; when coupled to FAST for blades, this is_
 ↪equivalent to the IEC blade (b) coordinate system (N)
 93 "FFcyr" - Elastic restoring force Fc y r: a floating reference coordinate system_
 ↪fixed to the root of the moving beam; when coupled to FAST for blades, this is_
 ↪equivalent to the IEC blade (b) coordinate system (N)

(continues on next page)

(continued from previous page)

94 "FFczzr" - Elastic restoring force F_c z r: a floating reference coordinate system
 ↳ fixed to the root of the moving beam; when coupled to FAST for blades, this is
 ↳ equivalent to the IEC blade (b) coordinate system (N)

95 "MFcx1" - Elastic restoring moment F_c about x l: a floating coordinate system
 ↳ local to the deflected beam (N-m)

96 "MFcy1" - Elastic restoring moment F_c about y l: a floating coordinate system
 ↳ local to the deflected beam (N-m)

97 "MFcz1" - Elastic restoring moment F_c about z l: a floating coordinate system
 ↳ local to the deflected beam (N-m)

98 "MFcxr" - Elastic restoring moment F_c about x r: a floating reference coordinate
 ↳ system fixed to the root of the moving beam; when coupled to FAST for blades, this is
 ↳ equivalent to the IEC blade (b) coordinate system (N-m)

99 "MFcyr" - Elastic restoring moment F_c about y r: a floating reference coordinate
 ↳ system fixed to the root of the moving beam; when coupled to FAST for blades, this is
 ↳ equivalent to the IEC blade (b) coordinate system (N-m)

100 "MFczzr" - Elastic restoring moment F_c about z r: a floating reference coordinate
 ↳ system fixed to the root of the moving beam; when coupled to FAST for blades, this is
 ↳ equivalent to the IEC blade (b) coordinate system (N-m)

101 "FFdx1" - Elastic restoring force F_d x l: a floating coordinate system local to
 ↳ the deflected beam (N)

102 "FFdy1" - Elastic restoring force F_d y l: a floating coordinate system local to
 ↳ the deflected beam (N)

103 "FFdz1" - Elastic restoring force F_d z l: a floating coordinate system local to
 ↳ the deflected beam (N)

104 "FFdxr" - Elastic restoring force F_d x r: a floating reference coordinate system
 ↳ fixed to the root of the moving beam; when coupled to FAST for blades, this is
 ↳ equivalent to the IEC blade (b) coordinate system (N)

105 "FFdyr" - Elastic restoring force F_d y r: a floating reference coordinate system
 ↳ fixed to the root of the moving beam; when coupled to FAST for blades, this is
 ↳ equivalent to the IEC blade (b) coordinate system (N)

106 "FFdzzr" - Elastic restoring force F_d z r: a floating reference coordinate system
 ↳ fixed to the root of the moving beam; when coupled to FAST for blades, this is
 ↳ equivalent to the IEC blade (b) coordinate system (N)

107 "MFdx1" - Elastic restoring moment F_d about x l: a floating coordinate system
 ↳ local to the deflected beam (N-m)

108 "MFdy1" - Elastic restoring moment F_d about y l: a floating coordinate system
 ↳ local to the deflected beam (N-m)

109 "MFdz1" - Elastic restoring moment F_d about z l: a floating coordinate system
 ↳ local to the deflected beam (N-m)

110 "MFdxr" - Elastic restoring moment F_d about x r: a floating reference coordinate
 ↳ system fixed to the root of the moving beam; when coupled to FAST for blades, this is
 ↳ equivalent to the IEC blade (b) coordinate system (N-m)

111 "MFdyr" - Elastic restoring moment F_d about y r: a floating reference coordinate
 ↳ system fixed to the root of the moving beam; when coupled to FAST for blades, this is
 ↳ equivalent to the IEC blade (b) coordinate system (N-m)

112 "MFdzzr" - Elastic restoring moment F_d about z r: a floating reference coordinate
 ↳ system fixed to the root of the moving beam; when coupled to FAST for blades, this is
 ↳ equivalent to the IEC blade (b) coordinate system (N-m)

113 "FFgx1" - Gravity force x l: a floating coordinate system local to the deflected
 ↳ beam (N)

114 "FFgy1" - Gravity force y l: a floating coordinate system local to the deflected
 ↳ beam (N)

(continues on next page)

(continued from previous page)

```

115 "FFgzl"    - Gravity force z      l: a floating coordinate system local to the deflected_
    ↳ beam      (N)
116 "FFgxr"    - Gravity force x      r: a floating reference coordinate system fixed to the_
    ↳ root of the moving beam; when coupled to FAST for blades, this is equivalent to the_
    ↳ IEC blade (b) coordinate system      (N)
117 "FFgyr"    - Gravity force y      r: a floating reference coordinate system fixed to the_
    ↳ root of the moving beam; when coupled to FAST for blades, this is equivalent to the_
    ↳ IEC blade (b) coordinate system      (N)
118 "FFg zr"    - Gravity force z      r: a floating reference coordinate system fixed to the_
    ↳ root of the moving beam; when coupled to FAST for blades, this is equivalent to the_
    ↳ IEC blade (b) coordinate system      (N)
119 "MFgxl"    - Gravity moment about x    l: a floating coordinate system local to the_
    ↳ deflected beam      (N-m)
120 "MFgyl"    - Gravity moment about y    l: a floating coordinate system local to the_
    ↳ deflected beam      (N-m)
121 "MFgzl"    - Gravity moment about z    l: a floating coordinate system local to the_
    ↳ deflected beam      (N-m)
122 "MFgxr"    - Gravity moment about x    r: a floating reference coordinate system fixed_
    ↳ to the root of the moving beam; when coupled to FAST for blades, this is equivalent to_
    ↳ the IEC blade (b) coordinate system      (N-m)
123 "MFgyr"    - Gravity moment about y    r: a floating reference coordinate system fixed_
    ↳ to the root of the moving beam; when coupled to FAST for blades, this is equivalent to_
    ↳ the IEC blade (b) coordinate system      (N-m)
124 "MFg zr"    - Gravity moment about z    r: a floating reference coordinate system fixed_
    ↳ to the root of the moving beam; when coupled to FAST for blades, this is equivalent to_
    ↳ the IEC blade (b) coordinate system      (N-m)
125 "FFixl"    - Inertial force x      l: a floating coordinate system local to the_
    ↳ deflected beam      (N)
126 "FFiyl"    - Inertial force y      l: a floating coordinate system local to the_
    ↳ deflected beam      (N)
127 "FFizl"    - Inertial force z      l: a floating coordinate system local to the_
    ↳ deflected beam      (N)
128 "FFixr"    - Inertial force x      r: a floating reference coordinate system fixed to_
    ↳ the root of the moving beam; when coupled to FAST for blades, this is equivalent to_
    ↳ the IEC blade (b) coordinate system      (N)
129 "FFiyr"    - Inertial force y      r: a floating reference coordinate system fixed to_
    ↳ the root of the moving beam; when coupled to FAST for blades, this is equivalent to_
    ↳ the IEC blade (b) coordinate system      (N)
130 "FFizr"    - Inertial force z      r: a floating reference coordinate system fixed to_
    ↳ the root of the moving beam; when coupled to FAST for blades, this is equivalent to_
    ↳ the IEC blade (b) coordinate system      (N)
131 "MFixl"    - Inertial moment about x    l: a floating coordinate system local to the_
    ↳ deflected beam      (N-m)
132 "MFiyl"    - Inertial moment about y    l: a floating coordinate system local to the_
    ↳ deflected beam      (N-m)
133 "MFizl"    - Inertial moment about z    l: a floating coordinate system local to the_
    ↳ deflected beam      (N-m)
134 "MFixr"    - Inertial moment about x    r: a floating reference coordinate system_
    ↳ fixed to the root of the moving beam; when coupled to FAST for blades, this is_
    ↳ equivalent to the IEC blade (b) coordinate system      (N-m)
135 "MFiyr"    - Inertial moment about y    r: a floating reference coordinate system_
    ↳ fixed to the root of the moving beam; when coupled to FAST for blades, this is_

```

(continues on next page)

(continued from previous page)

```

136 ↪equivalent to the IEC blade (b) coordinate system      (N-m)
"MFizr"      - Inertial moment about z      r: a floating reference coordinate system ↪
137 ↪fixed to the root of the moving beam; when coupled to FAST for blades, this is ↪
↪equivalent to the IEC blade (b) coordinate system      (N-m)
137 END of input file (the word "END" must appear in the first 3 columns of this last ↪
↪OutList line)
138 -----

```

Blade Input File

The blade input file defines the cross-sectional properties at various stations along a blade and six damping coefficient for the whole blade. A sample BeamDyn blade input file is given in [Section 4.2.4](#). The blade input file begins with two lines of header information, which is for the user but is not used by the software.

Blade Parameters

`Station_Total` specifies the number cross-sectional stations along the blade axis used in the analysis.

`Damp_Type` specifies if structural damping is considered in the analysis. If `Damp_Type = 0`, then no damping is considered in the analysis and the six damping coefficient in the next section will be ignored. If `Damp_Type = 1`, structural damping will be included in the analysis.

Damping Coefficient

This section specifies six damping coefficients, μ_{ii} with $i \in [1, 6]$, for six DOFs (three translations and three rotations). Viscous damping is implemented in BeamDyn where the damping forces are proportional to the strain rate. These are stiffness-proportional damping coefficients, whereby the 6×6 damping matrix at each cross section is scaled from the 6×6 stiffness matrix by these diagonal entries of a 6×6 scaling matrix:

$$\underline{\mathcal{F}}^{Damp} = \underline{\mu} \underline{S} \dot{\underline{\epsilon}} \quad (4.60)$$

where $\underline{\mathcal{F}}^{Damp}$ is the damping force, \underline{S} is the 6×6 cross-sectional stiffness matrix, $\dot{\underline{\epsilon}}$ is the strain rate, and $\underline{\mu}$ is the damping coefficient matrix defined as

$$\underline{\mu} = \begin{bmatrix} \mu_{11} & 0 & 0 & 0 & 0 & 0 \\ 0 & \mu_{22} & 0 & 0 & 0 & 0 \\ 0 & 0 & \mu_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu_{66} \end{bmatrix} \quad (4.61)$$

Distributed Properties

This section specifies the cross-sectional properties at each of the `Station_Total` stations. For each station, a non-dimensional parameter η specifies the station location along the local blade reference axis ranging from $[0.0, 1.0]$. The first and last station parameters must be set to 0.0 (for the blade root) and 1.0 (for the blade tip), respectively.

Following the station location parameter η , there are two 6×6 matrices providing the structural and inertial properties for this cross-section. First is the stiffness matrix and then the mass matrix. We note that these matrices are defined in a local coordinate system along the blade axis with Z_l directing toward the unit tangent vector of the blade reference axis. For a cross-section without coupling effects, for example, the stiffness matrix is given as follows:

$$\begin{bmatrix} K_{ShrFlp} & 0 & 0 & 0 & 0 & 0 \\ 0 & K_{ShrEdg} & 0 & 0 & 0 & 0 \\ 0 & 0 & EA & 0 & 0 & 0 \\ 0 & 0 & 0 & EI_{Edg} & 0 & 0 \\ 0 & 0 & 0 & 0 & EI_{Flp} & 0 \\ 0 & 0 & 0 & 0 & 0 & GJ \end{bmatrix} \quad (4.62)$$

where K_{ShrEdg} and K_{ShrFlp} are the edge and flap shear stiffnesses, respectively; EA is the extension stiffness; EI_{Edg} and EI_{Flp} are the edge and flap stiffnesses, respectively; and GJ is the torsional stiffness. It is pointed out that for a generic cross-section, the sectional property matrices can be derived from a sectional analysis tool, e.g. VABS, BECAS, or NuMAD/BPE.

A generalized sectional mass matrix is given by:

$$\begin{bmatrix} m & 0 & 0 & 0 & 0 & -mY_{cm} \\ 0 & m & 0 & 0 & 0 & mX_{cm} \\ 0 & 0 & m & mY_{cm} & -mX_{cm} & 0 \\ 0 & 0 & mY_{cm} & i_{Edg} & -i_{cp} & 0 \\ 0 & 0 & -mX_{cm} & -i_{cp} & i_{Flp} & 0 \\ -mY_{cm} & mX_{cm} & 0 & 0 & 0 & i_{plr} \end{bmatrix} \quad (4.63)$$

where m is the mass density per unit span; X_{cm} and Y_{cm} are the local coordinates of the sectional center of mass, respectively; i_{Edg} and i_{Flp} are the edge and flap mass moments of inertia per unit span, respectively; i_{plr} is the polar moment of inertia per unit span; and i_{cp} is the sectional cross-product of inertia per unit span. We note that for beam structure, the i_{plr} is given as (although this relationship is not checked by BeamDyn)

$$i_{plr} = i_{Edg} + i_{Flp} \quad (4.64)$$

Output Files

BeamDyn produces three types of output files, depending on the options selected: an echo file, a summary file, and a time-series results file. The following sections detail the purpose and contents of these files.

Echo File

If the user sets the Echo flag to TRUE in the BeamDyn primary input file, the contents of this file will be echoed to a file with the naming convention `InputFile.ech`. The echo file is helpful for debugging the input files. The contents of an echo file will be truncated if BeamDyn encounters an error while parsing an input file. The error usually corresponds to the line after the last successfully echoed line.

Summary File

In stand-alone mode, BeamDyn generates a summary file with the naming convention, `InputFile.sum` if the `SumPrint` parameter is set to `TRUE`. When coupled to FAST, the summary file is named `InputFile.BD.sum`. This file summarizes key information about the simulation, including:

- Blade mass.
- Blade length.
- Blade center of mass.
- Initial global position vector in BD coordinate system.
- Initial global rotation tensor in BD coordinate system.
- Analysis type.
- Numerical damping coefficients.
- Time step size.
- Maximum number of iterations in the Newton-Raphson solution.
- Convergence parameter in the stopping criterion.
- Factorization frequency in the Newton-Raphson solution.
- Numerical integration (quadrature) method.
- FE mesh refinement factor used in trapezoidal quadrature.
- Number of elements.
- Number of FE nodes.
- Initial position vectors of FE nodes in BD coordinate system.
- Initial rotation vectors of FE nodes in BD coordinate system.
- Quadrature point position vectors in BD coordinate system. For Gauss quadrature, the physical coordinates of Gauss points are listed. For trapezoidal quadrature, the physical coordinates of the quadrature points are listed.
- Sectional stiffness and mass matrices at quadrature points in local blade reference coordinate system. These are the data being used in calculations at quadrature points and they can be different from the section in Blade Input File since BeamDyn linearly interpolates the sectional properties into quadrature points based on need.
- Initial displacement vectors of FE nodes in BD coordinate system.
- Initial rotational displacement vectors of FE nodes in BD coordinate system.
- Initial translational velocity vectors of FE nodes in BD coordinate system.
- Initial angular velocity vectors of FE nodes in BD coordinate system.
- Requested output information.

All of these quantities are output in this file in the BD coordinate system, the one being used internally in BeamDyn calculations. The initial blade reference coordinate system, denoted by a subscript $r0$ that follows the IEC standard, is related to the internal BD coordinate system by [Table 4.6](#) in [Section 4.2.4](#).

Results File

The BeamDyn time-series results are written to a text-based file with the naming convention `DriverInputFile.out` where `DriverInputFile` is the name of the driver input file when BeamDyn is run in the stand-alone mode. If BeamDyn is coupled to FAST, then FAST will generate a master results file that includes the BeamDyn results. The results in `DriverInputFile.out` are in table format, where each column is a data channel (the first column always being the simulation time), and each row corresponds to a simulation time step. The data channel are specified in the OUTPUT section of the primary input file. The column format of the BeamDyn-generated file is specified using the `OutFmt` parameters of the primary input file.

BeamDyn Theory

This section focuses on the theory behind the BeamDyn module. The theoretical foundation, numerical tools, and some special handling in the implementation will be introduced. References will be provided in each section detailing the theories and numerical tools.

In this chapter, matrix notation is used to denote vectorial or vectorial-like quantities. For example, an underline denotes a vector \underline{u} , an over bar denotes unit vector \bar{n} , and a double underline denotes a tensor $\underline{\underline{A}}$. Note that sometimes the underlines only denote the dimension of the corresponding matrix.

Coordinate Systems

Fig. 4.28 (in Section 4.2.4) and Fig. 4.29 show the coordinate system used in BeamDyn.

Global Coordinate System

The global coordinate system is denoted as X , Y , and Z in Fig. 4.29. This is an inertial frame and in FAST its origin is usually placed at the bottom of the tower as shown.

BD Coordinate System

The BD coordinate system is denoted as x_1 , x_2 , and x_3 respectively in Fig. 4.29. This is an inertial frame used internally in BeamDyn (i.e., doesn't rotate with the rotor) and its origin is placed at the initial position of the blade root point.

Blade Reference Coordinate System

The blade reference coordinate system is denoted as X_{rt} , Y_{rt} , and Z_{rt} in Fig. 4.29 at initialization ($t = 0$). The blade reference coordinate system is a floating frame that attaches at the blade root and is rotating with the blade. Its origin is at the blade root and the directions of axes following the IEC standard, i.e., Z_r is pointing along the blade axis from root to tip; Y_r pointing nominally towards the trailing edge of the blade and parallel with the chord line at the zero-twist blade station; and X_r is orthogonal with the Y_r and Z_r axes, such that they form a right-handed coordinate system (pointing nominally downwind). We note that the initial blade reference coordinate system, denoted by subscript $r0$, coincides with the BD coordinate system, which is used internally in BeamDyn and introduced in the previous section. The axis convention relations between the initial blade reference coordinate system and the BD coordinate system can be found in Table 4.6.

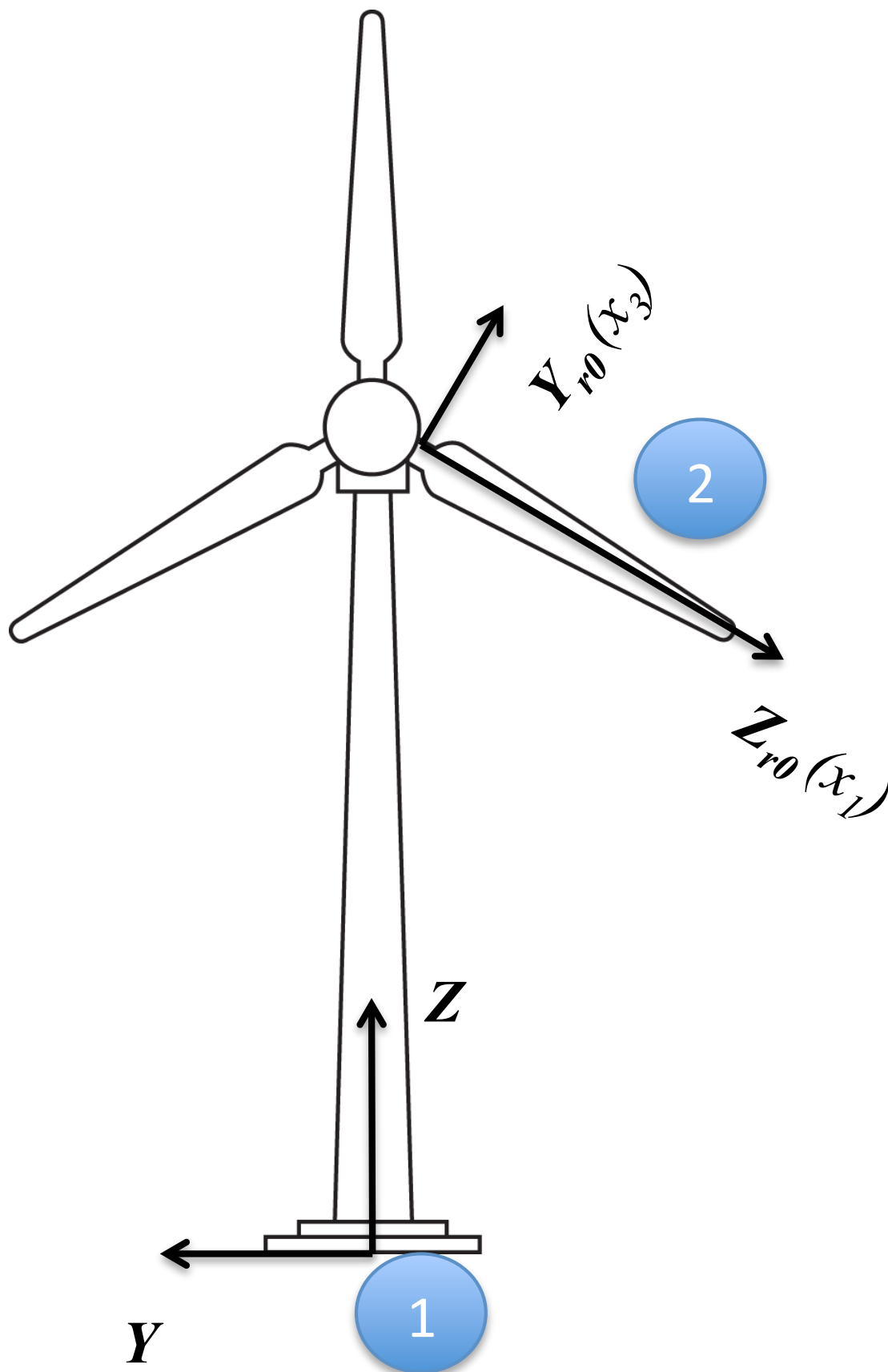


Fig. 4.29: Global, blade reference, and internal coordinate systems in BeamDyn. Illustration by Al Hicks, NREL.

Table 4.6: Transformation between blade coordinate system and BD coordinate system.

Blade Frame	X_{r0}	Y_{r0}	Z_{r0}
BD Frame	x_2	x_3	x_1

Local blade coordinate system

The local blade coordinate system is used for some input and output quantities, for example, the cross-sectional mass and stiffness matrices and the sectional force and moment resultants. This coordinate system is different from the blade reference coordinate system in that its Z_l axis is always tangent to the blade axis as the blade deflects. Note that a subscript l denotes the local blade coordinate system.

Geometrically Exact Beam Theory

The theoretical foundation of BeamDyn is the geometrically exact beam theory. This theory features the capability of beams that are initially curved and twisted and subjected to large displacement and rotations. Along with a proper two-dimensional (2D) cross-sectional analysis, the coupling effects between all six DOFs, including extension, bending, shear, and torsion, can be captured by GEBT as well. The term, “geometrically exact” refer to the fact that there is no approximation made on the geometries, including both initial and deformed geometries, in formulating the equations [Hod06].

The governing equations of motion for geometrically exact beam theory can be written as [Bau10]

$$\begin{aligned} \dot{\underline{h}} - \underline{F}' &= \underline{f} \\ \dot{\underline{g}} + \dot{\underline{u}}\underline{h} - \underline{M}' + (\tilde{x}'_0 + \tilde{u}')^T \underline{F} &= \underline{m} \end{aligned} \quad (4.65)$$

where \underline{h} and \underline{g} are the linear and angular momenta resolved in the inertial coordinate system, respectively; \underline{F} and \underline{M} are the beam’s sectional force and moment resultants, respectively; \underline{u} is the one-dimensional (1D) displacement of a point on the reference line; \underline{x}_0 is the position vector of a point along the beam’s reference line; and \underline{f} and \underline{m} are the distributed force and moment applied to the beam structure. The notation $(\bullet)'$ indicates a derivative with respect to beam axis x_1 and $(\dot{\bullet})$ indicates a derivative with respect to time. The tilde operator $(\tilde{\bullet})$ defines a skew-symmetric tensor corresponding to the given vector. In the literature, it is also termed as “cross-product matrix”. For example,

$$\tilde{n} = \begin{bmatrix} 0 & -n_3 & n_2 \\ n_3 & 0 & -n_1 \\ -n_2 & n_1 & 0 \end{bmatrix}$$

The constitutive equations relate the velocities to the momenta and the 1D strain measures to the sectional resultants as

$$\begin{Bmatrix} \underline{h} \\ \underline{g} \end{Bmatrix} = \underline{\underline{M}} \begin{Bmatrix} \dot{\underline{u}} \\ \underline{\omega} \end{Bmatrix} \quad (4.66)$$

$$\begin{Bmatrix} \underline{F} \\ \underline{M} \end{Bmatrix} = \underline{\underline{C}} \begin{Bmatrix} \underline{\epsilon} \\ \underline{\kappa} \end{Bmatrix}$$

where $\underline{\underline{M}}$ and $\underline{\underline{C}}$ are the 6×6 sectional mass and stiffness matrices, respectively (note that they are not really tensors); $\underline{\epsilon}$ and $\underline{\kappa}$ are the 1D strains and curvatures, respectively; and, $\underline{\omega}$ is the angular velocity vector that is defined by the rotation tensor \underline{R} as $\underline{\omega} = \text{axial}(\dot{\underline{R}} \underline{R}^T)$. The axial vector \underline{a} associated with a second-order tensor \underline{A} is denoted $\underline{a} = \text{axial}(\underline{A})$ and its components are defined as

$$\underline{a} = \text{axial}(\underline{A}) = \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \end{Bmatrix} = \frac{1}{2} \begin{Bmatrix} A_{32} - A_{23} \\ A_{13} - A_{31} \\ A_{21} - A_{12} \end{Bmatrix} \quad (4.67)$$

The 1D strain measures are defined as

$$\begin{Bmatrix} \underline{\epsilon} \\ \underline{\kappa} \end{Bmatrix} = \begin{Bmatrix} \underline{x}'_0 + \underline{u}' - (\underline{R} \underline{R}_0) \bar{\mathbf{i}}_1 \\ \underline{k} \end{Bmatrix} \quad (4.68)$$

where $\underline{k} = \text{axial}[(\underline{R} \underline{R}_0)'(\underline{R} \underline{R}_0)^T]$ is the sectional curvature vector resolved in the inertial basis; \underline{R}_0 is the initial rotation tensor; and $\bar{\mathbf{i}}_1$ is the unit vector along x_1 direction in the inertial basis. These three sets of equations, including equations of motion Eq. (4.65), constitutive equations Eq. (4.66), and kinematical equations Eq. (4.68), provide a full mathematical description of the beam elasticity problems.

Numerical Implementation with Legendre Spectral Finite Elements

For a displacement-based finite element implementation, there are six degree-of-freedom at each node: three displacement components and three rotation components. Here we use \underline{q} to denote the elemental displacement array as $\underline{q} = [\underline{u}^T \ \underline{c}^T]^T$ where \underline{u} is the displacement and \underline{c} is the rotation-parameter vector. The acceleration array can thus be defined as $\underline{a} = [\underline{\ddot{u}}^T \ \underline{\dot{\omega}}^T]^T$. For nonlinear finite-element analysis, the discretized and incremental forms of displacement, velocity, and acceleration are written as

$$\begin{aligned} \underline{q}(x_1) &= \underline{N} \hat{\underline{q}} & \Delta \underline{q}^T &= [\Delta \underline{u}^T \ \Delta \underline{c}^T] \\ \underline{v}(x_1) &= \underline{N} \hat{\underline{v}} & \Delta \underline{v}^T &= [\Delta \underline{\dot{u}}^T \ \Delta \underline{\dot{\omega}}^T] \\ \underline{a}(x_1) &= \underline{N} \hat{\underline{a}} & \Delta \underline{a}^T &= [\Delta \underline{\ddot{u}}^T \ \Delta \underline{\dot{\omega}}^T] \end{aligned} \quad (4.69)$$

where \underline{N} is the shape function matrix and $(\hat{\cdot})$ denotes a column matrix of nodal values.

The displacement fields in an element are approximated as

$$\begin{aligned} \underline{u}(\xi) &= h^k(\xi) \hat{\underline{u}}^k \\ \underline{u}'(\xi) &= h^{k'}(\xi) \hat{\underline{u}}^k \end{aligned} \quad (4.70)$$

where $h^k(\xi)$, the component of shape function matrix \underline{N} , is the p^{th} -order polynomial Lagrangian-interpolant shape function of node k , $k = \{1, 2, \dots, p+1\}$, $\hat{\underline{u}}^k$ is the k^{th} nodal value, and $\xi \in [-1, 1]$ is the element natural coordinate. However, as discussed in [BEH08], the 3D rotation field cannot simply be interpolated as the displacement field in the form of

$$\begin{aligned} \underline{c}(\xi) &= h^k(\xi) \hat{\underline{c}}^k \\ \underline{c}'(\xi) &= h^{k'}(\xi) \hat{\underline{c}}^k \end{aligned} \quad (4.71)$$

where \underline{c} is the rotation field in an element and $\hat{\underline{c}}^k$ is the nodal value at the k^{th} node, for three reasons:

- 1) rotations do not form a linear space so that they must be “composed” rather than added;
- 2) a rescaling operation is needed to eliminate the singularity existing in the vectorial rotation parameters;
- 3) the rotation field lacks objectivity, which, as defined by [JelenicC99], refers to the invariance of strain measures computed through interpolation to the addition of a rigid-body motion.

Therefore, we adopt the more robust interpolation approach proposed by [JelenicC99] to deal with the finite rotations. Our approach is described as follows

Step 1:

Compute the nodal relative rotations, $\hat{\underline{r}}^k$, by removing the reference rotation, $\hat{\underline{c}}^1$, from the finite rotation at each node, $\hat{\underline{r}}^k = (\hat{\underline{c}}^{1-}) \oplus \hat{\underline{c}}^k$. It is noted that the minus sign on $\hat{\underline{c}}^1$ denotes that the relative rotation is calculated by removing the reference rotation from each node. The composition in that equation is an equivalent of $\underline{R}(\hat{\underline{r}}^k) = \underline{R}^T(\hat{\underline{c}}^1) \underline{R}(\hat{\underline{c}}^k)$.

Step 2:

Interpolate the relative-rotation field: $\underline{r}(\xi) = h^k(\xi)\hat{\underline{r}}^k$ and $\underline{r}'(\xi) = h^{k'}(\xi)\hat{\underline{r}}^{k'}$. Find the curvature field $\underline{\kappa}(\xi) = \underline{\underline{R}}(\hat{\underline{c}}^1)\underline{\underline{H}}(\underline{r})\underline{r}'$, where $\underline{\underline{H}}$ is the tangent tensor that relates the curvature vector $\underline{\kappa}$ and rotation vector \underline{c} as

$$\underline{\kappa} = \underline{\underline{H}} \underline{c}' \quad (4.72)$$

Step 3:

Restore the rigid-body rotation removed in Step 1: $\underline{c}(\xi) = \hat{\underline{c}}^1 \oplus \underline{r}(\xi)$.

Note that the relative-rotation field can be computed with respect to any of the nodes of the element; we choose node 1 as the reference node for convenience. In the LSFE approach, shape functions (i.e., those composing \underline{N}) are p^{th} -order Lagrangian interpolants, where nodes are located at the $p + 1$ Gauss-Lobatto-Legendre (GLL) points in the $[-1, 1]$ element natural-coordinate domain. Fig. 4.30 shows representative LSFE basis functions for fourth- and eighth-order elements. Note that nodes are clustered near element endpoints. More details on the LSFE and its applications can be found in References [Pat84, RP87, SG03, SG04].

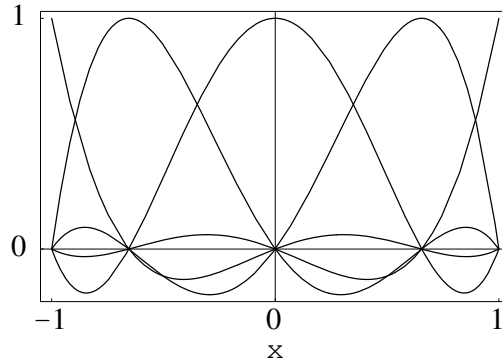


Fig. 4.30: Representative $p + 1$ Lagrangian-interpolant shape functions in the element natural coordinates for a fourth-order LSFEs, where nodes are located at the Gauss-Lobatto-Legendre points.

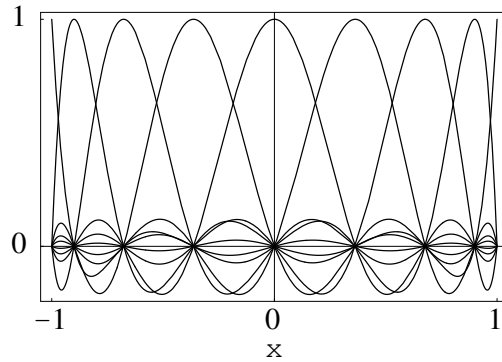


Fig. 4.31: Representative $p + 1$ Lagrangian-interpolant shape functions in the element natural coordinates for an eighth-order LSFEs, where nodes are located at the Gauss-Lobatto-Legendre points.

Wiener-Milenković Rotation Parameter

In BeamDyn, the 3D rotations are represented as Wiener-Milenković parameters defined in the following equation:

$$\underline{c} = 4 \tan\left(\frac{\phi}{4}\right) \bar{n} \quad (4.73)$$

where ϕ is the rotation angle and \bar{n} is the unit vector of the rotation axis. It can be observed that the valid range for this parameter is $|\phi| < 2\pi$. The singularities existing at integer multiples of $\pm 2\pi$ can be removed by a rescaling operation at π as:

$$\underline{r} = \begin{cases} 4(q_0 \underline{p} + p_0 \underline{q} + \tilde{p} \underline{q})/(\Delta_1 + \Delta_2), & \text{if } \Delta_2 \geq 0 \\ -4(q_0 \underline{p} + p_0 \underline{q} + \tilde{p} \underline{q})/(\Delta_1 - \Delta_2), & \text{if } \Delta_2 < 0 \end{cases} \quad (4.74)$$

where \underline{p} , \underline{q} , and \underline{r} are the vectorial parameterization of three finite rotations such that $\underline{R}(\underline{r}) = \underline{R}(\underline{p})\underline{R}(\underline{q})$, $p_0 = 2 - \underline{p}^T \underline{p}/8$, $q_0 = 2 - \underline{q}^T \underline{q}/8$, $\Delta_1 = (4 - p_0)(4 - q_0)$, and $\Delta_2 = p_0 q_0 - \underline{p}^T \underline{q}$. It is noted that the rescaling operation could cause a discontinuity of the interpolated rotation field; therefore a more robust interpolation algorithm has been introduced in Section *Numerical Implementation with Legendre Spectral Finite Elements* where the rescaling-independent relative-rotation field is interpolated.

The rotation tensor expressed in terms of Wiener-Milenković parameters is

$$\underline{R}(\underline{c}) = \frac{1}{(4 - c_0)^2} \begin{bmatrix} c_0^2 + c_1^2 - c_2^2 - c_3^2 & 2(c_1 c_2 - c_0 c_3) & 2(c_1 c_3 + c_0 c_2) \\ 2(c_1 c_2 + c_0 c_3) & c_0^2 - c_1^2 + c_2^2 - c_3^2 & 2(c_2 c_3 - c_0 c_1) \\ 2(c_1 c_3 - c_0 c_2) & 2(c_2 c_3 + c_0 c_1) & c_0^2 - c_1^2 - c_2^2 + c_3^2 \end{bmatrix} \quad (4.75)$$

where $\underline{c} = [c_1 \ c_2 \ c_3]^T$ is the Wiener-Milenković parameter and $c_0 = 2 - \frac{1}{8} \underline{c}^T \underline{c}$. The relation between rotation tensor and direction cosine matrix (DCM) is

$$\underline{R} = (\underline{DCM})^T \quad (4.76)$$

Interested users are referred to [BEH08] and [WYS13] for more details on the rotation parameter and its implementation with GEBT.

Linearization Process

The nonlinear governing equations introduced in the previous section are solved by Newton-Raphson method, where a linearization process is needed. The linearization of each term in the governing equations are presented in this section.

According to [Bau10], the linearized governing equations in Eq. (4.65) are in the form of

$$\underline{\hat{M}} \Delta \hat{u} + \underline{\hat{G}} \Delta \hat{v} + \underline{\hat{K}} \Delta \hat{q} = \underline{\hat{F}}^{ext} - \underline{\hat{F}} \quad (4.77)$$

where the $\underline{\hat{M}}$, $\underline{\hat{G}}$, and $\underline{\hat{K}}$ are the elemental mass, gyroscopic, and stiffness matrices, respectively; $\underline{\hat{F}}$ and $\underline{\hat{F}}^{ext}$ are the elemental forces and externally applied loads, respectively. They are defined for an element of length l along x_1 as follows

$$\begin{aligned} \underline{\hat{M}} &= \int_0^l \underline{N}^T \underline{\mathcal{M}} \underline{N} dx_1 \\ \underline{\hat{G}} &= \int_0^l \underline{N}^T \underline{\mathcal{G}}^I \underline{N} dx_1 \\ \underline{\hat{K}} &= \int_0^l \left[\underline{N}^T (\underline{\mathcal{K}}^I + \underline{\mathcal{Q}}) \underline{N} + \underline{N}^T \underline{\mathcal{P}} \underline{N}' + \underline{N}'^T \underline{\mathcal{C}} \underline{N}' + \underline{N}'^T \underline{\mathcal{Q}} \underline{N} \right] dx_1 \\ \underline{\hat{F}} &= \int_0^l (\underline{N}^T \underline{\mathcal{F}}^I + \underline{N}^T \underline{\mathcal{F}}^D + \underline{N}'^T \underline{\mathcal{F}}^C) dx_1 \\ \underline{\hat{F}}^{ext} &= \int_0^l \underline{N}^T \underline{\mathcal{F}}^{ext} dx_1 \end{aligned} \quad (4.78)$$

where $\underline{\mathcal{F}}^{ext}$ is the applied load vector. The new matrix notations in Eqs. (4.78) to are briefly introduced here. $\underline{\mathcal{F}}^C$ and $\underline{\mathcal{F}}^D$ are elastic forces obtained from Eq. (4.65) as

$$\begin{aligned}\underline{\mathcal{F}}^C &= \begin{Bmatrix} \underline{F} \\ \underline{M} \end{Bmatrix} = \underline{\mathcal{C}} \begin{Bmatrix} \underline{\epsilon} \\ \underline{\kappa} \end{Bmatrix} \\ \underline{\mathcal{F}}^D &= \begin{bmatrix} \underline{0} \\ (\tilde{x}'_0 + \tilde{u}')^T \end{bmatrix} \underline{\mathcal{F}}^C \equiv \underline{\Upsilon} \underline{\mathcal{F}}^C\end{aligned}\quad (4.79)$$

where $\underline{0}$ denotes a 3×3 null matrix. The $\underline{\mathcal{G}}^I$, $\underline{\mathcal{K}}^I$, $\underline{\mathcal{Q}}$, $\underline{\mathcal{P}}$, $\underline{\mathcal{Q}}$, and $\underline{\mathcal{F}}^I$ in Eqs. (4.78) are defined as

$$\begin{aligned}\underline{\mathcal{G}}^I &= \begin{bmatrix} \underline{0} & (\widetilde{\omega m \eta})^T + \widetilde{\omega m \tilde{\eta}}^T \\ \underline{0} & \widetilde{\omega \underline{\rho}} - \underline{\underline{\rho}} \widetilde{\omega} \end{bmatrix} \\ \underline{\mathcal{K}}^I &= \begin{bmatrix} \underline{0} & \dot{\omega} m \tilde{\eta}^T + \widetilde{\omega \omega} m \tilde{\eta}^T \\ \underline{0} & \ddot{u} m \tilde{\eta} + \underline{\underline{\rho}} \dot{\omega} - \underline{\underline{\rho}} \dot{\omega} + \widetilde{\omega \rho \omega} - \widetilde{\omega \underline{\rho} \omega} \end{bmatrix} \\ \underline{\mathcal{Q}} &= \begin{bmatrix} \underline{0} & \underline{\underline{C}}_{11} \tilde{E}_1 - \tilde{F} \\ \underline{0} & \underline{\underline{C}}_{21} \tilde{E}_1 - \tilde{M} \end{bmatrix} \\ \underline{\mathcal{P}} &= \begin{bmatrix} \underline{0} & \underline{0} \\ \tilde{F} + (\underline{\underline{C}}_{11} \tilde{E}_1)^T & (\underline{\underline{C}}_{21} \tilde{E}_1)^T \end{bmatrix} \\ \underline{\mathcal{Q}} &= \underline{\Upsilon} \underline{\mathcal{Q}} \\ \underline{\mathcal{F}}^I &= \begin{Bmatrix} m \ddot{u} + (\dot{\omega} + \widetilde{\omega \omega}) m \underline{\eta} \\ m \tilde{\eta} \ddot{u} + \underline{\underline{\rho}} \dot{\omega} + \widetilde{\omega \underline{\rho} \omega} \end{Bmatrix}\end{aligned}\quad (4.80)$$

where m is the mass density per unit length, $\underline{\eta}$ is the location of the sectional center of mass, $\underline{\rho}$ is the moment of inertia tensor, and the following notations were introduced to simplify the above expressions

$$\begin{aligned}\underline{E}_1 &= \underline{x}'_0 + \underline{u}' \\ \underline{\mathcal{C}} &= \begin{bmatrix} \underline{\underline{C}}_{11} & \underline{\underline{C}}_{12} \\ \underline{\underline{C}}_{21} & \underline{\underline{C}}_{22} \end{bmatrix}\end{aligned}\quad (4.81)$$

Damping Forces and Linearization

A viscous damping model has been implemented into BeamDyn to account for the structural damping effect. The damping force is defined as

$$\underline{f}_d = \underline{\mu} \underline{\mathcal{C}} \begin{Bmatrix} \dot{\underline{\epsilon}} \\ \dot{\underline{\kappa}} \end{Bmatrix}\quad (4.82)$$

where $\underline{\mu}$ is a user-defined damping-coefficient diagonal matrix. The damping force can be recast in two separate parts, like $\underline{\mathcal{F}}^C$ and $\underline{\mathcal{F}}^D$ in the elastic force, as

$$\begin{aligned}\underline{\mathcal{F}}_d^C &= \begin{Bmatrix} \underline{F}_d \\ \underline{M}_d \end{Bmatrix} \\ \underline{\mathcal{F}}_d^D &= \begin{Bmatrix} \underline{0} \\ (\tilde{x}'_0 + \tilde{u}')^T \underline{F}_d \end{Bmatrix}\end{aligned}\quad (4.83)$$

The linearization of the structural damping forces are as follows:

$$\begin{aligned}\Delta \underline{\mathcal{F}}_d^C &= \underline{\mathcal{S}}_d \begin{Bmatrix} \Delta \underline{u}' \\ \Delta \underline{c}' \end{Bmatrix} + \underline{\mathcal{Q}}_d \begin{Bmatrix} \Delta \underline{u} \\ \Delta \underline{c} \end{Bmatrix} + \underline{\mathcal{G}}_d \begin{Bmatrix} \Delta \underline{\dot{u}} \\ \Delta \underline{\dot{\omega}} \end{Bmatrix} + \underline{\mu} \underline{\mathcal{C}} \begin{Bmatrix} \Delta \underline{\dot{u}}' \\ \Delta \underline{\dot{\omega}}' \end{Bmatrix} \\ \Delta \underline{\mathcal{F}}_d^D &= \underline{\mathcal{P}}_d \begin{Bmatrix} \Delta \underline{u}' \\ \Delta \underline{c}' \end{Bmatrix} + \underline{\mathcal{Q}}_d \begin{Bmatrix} \Delta \underline{u} \\ \Delta \underline{c} \end{Bmatrix} + \underline{\mathcal{X}}_d \begin{Bmatrix} \Delta \underline{\dot{u}} \\ \Delta \underline{\dot{\omega}} \end{Bmatrix} + \underline{\mathcal{Y}}_d \begin{Bmatrix} \Delta \underline{\dot{u}}' \\ \Delta \underline{\dot{\omega}}' \end{Bmatrix}\end{aligned}\quad (4.84)$$

where the newly introduced matrices are defined as

$$\begin{aligned}
 \underline{\underline{S}}_d &= \underline{\underline{\mu C}} \begin{bmatrix} \underline{\underline{\omega}}^T & \underline{\underline{0}} \\ \underline{\underline{0}} & \underline{\underline{\omega}}^T \end{bmatrix} \\
 \underline{\underline{Q}}_d &= \begin{bmatrix} \underline{\underline{0}} & \underline{\underline{\mu C}}_{11} (\dot{\underline{\underline{u}}} - \underline{\underline{\omega}} \tilde{E}_1) - \tilde{F}_d \\ \underline{\underline{0}} & \underline{\underline{\mu C}}_{21} (\dot{\underline{\underline{u}}} - \underline{\underline{\omega}} \tilde{E}_1) - \tilde{M}_d \end{bmatrix} \\
 \underline{\underline{G}}_d &= \begin{bmatrix} \underline{\underline{0}} & \underline{\underline{C}}_{11}^T \underline{\underline{\mu}}^T \tilde{E}_1 \\ \underline{\underline{0}} & \underline{\underline{C}}_{12}^T \underline{\underline{\mu}}^T \tilde{E}_1 \end{bmatrix} \\
 \underline{\underline{P}}_d &= \begin{bmatrix} \underline{\underline{0}} & \underline{\underline{0}} \\ \tilde{F}_d + \tilde{E}_1^T \underline{\underline{\mu C}}_{11} \underline{\underline{\omega}}^T & \tilde{E}_1^T \underline{\underline{\mu C}}_{12} \underline{\underline{\omega}}^T \end{bmatrix} \\
 \underline{\underline{Q}}_d &= \begin{bmatrix} \underline{\underline{0}} & \underline{\underline{0}} \\ \underline{\underline{0}} & \tilde{E}_1^T \underline{\underline{Q}}_{12} \end{bmatrix} \\
 \underline{\underline{X}}_d &= \begin{bmatrix} \underline{\underline{0}} & \underline{\underline{0}} \\ \underline{\underline{0}} & \tilde{E}_1^T \underline{\underline{G}}_{12} \end{bmatrix} \\
 \underline{\underline{Y}}_d &= \begin{bmatrix} \underline{\underline{0}} & \underline{\underline{0}} \\ \tilde{E}_1^T \underline{\underline{\mu C}}_{11} & \tilde{E}_1^T \underline{\underline{\mu C}}_{12} \end{bmatrix}
 \end{aligned} \tag{4.85}$$

where $\underline{\underline{Q}}_{12}$ and $\underline{\underline{G}}_{12}$ are the 3×3 sub matrices of $\underline{\underline{Q}}$ and $\underline{\underline{G}}$ as $\underline{\underline{C}}_{12}$ in Eq. (4.81).

Convergence Criterion and Generalized- α Time Integrator

The system of nonlinear equations in Eqs. (4.65) are solved using the Newton-Raphson method with the linearized form in Eq. (4.77). In the present implementation, an energy-like stopping criterion has been chosen, which is calculated as

$$|\Delta \mathbf{U}^{(i)T} ({}^{t+\Delta t} \mathbf{R} - {}^{t+\Delta t} \mathbf{F}^{(i-1)})| \leq |\epsilon_E (\Delta \mathbf{U}^{(1)T} ({}^{t+\Delta t} \mathbf{R} - {}^t \mathbf{F}))| \tag{4.86}$$

where $|\cdot|$ denotes the absolute value, $\Delta \mathbf{U}$ is the incremental displacement vector, \mathbf{R} is the vector of externally applied nodal point loads, \mathbf{F} is the vector of nodal point forces corresponding to the internal element stresses, and ϵ_E is the user-defined energy tolerance. The superscript on the left side of a variable denotes the time-step number (in a dynamic analysis), while the one on the right side denotes the Newton-Raphson iteration number. As pointed out by [BC80], this criterion provides a measure of when both the displacements and the forces are near their equilibrium values.

Time integration is performed using the generalized- α scheme in BeamDyn, which is an unconditionally stable (for linear systems), second-order accurate algorithm. The scheme allows for users to choose integration parameters that introduce high-frequency numerical dissipation. More details regarding the generalized- α method can be found in [Bau10, CH93].

Calculation of Reaction Loads

Since the root motion of the wind turbine blade, including displacements and rotations, translational and angular velocities, and translational and angular accelerates, are prescribed as inputs to BeamDyn either by the driver (in stand-alone mode) or by FAST glue code (in FAST-coupled mode), the reaction loads at the root are needed to satisfy equality of the governing equations. The reaction loads at the root are also the loads passing from blade to hub in a full turbine analysis.

The governing equations in Eq. (4.65) can be recast in a compact form

$$\underline{\underline{F}}^I - \underline{\underline{F}}^{C'} + \underline{\underline{F}}^D = \underline{\underline{F}}^{ext} \tag{4.87}$$

with all the vectors defined in Section [sec:LinearProcess]. At the blade root, the governing equation is revised as

$$\underline{\mathcal{F}}^I - \underline{\mathcal{F}}^{C'} + \underline{\mathcal{F}}^D = \underline{\mathcal{F}}^{ext} + \underline{\mathcal{F}}^R \quad (4.88)$$

where $\underline{\mathcal{F}}^R = [\underline{F}^R \quad \underline{M}^R]^T$ is the reaction force vector and it can be solved from Eq. (4.88) given that the motion fields are known at this point.

Calculation of Blade Loads

BeamDyn can also calculate the blade loads at each finite element node along the blade axis. The governing equation in Eq. (4.87) are recast as

$$\underline{\mathcal{F}}^A + \underline{\mathcal{F}}^V - \underline{\mathcal{F}}^{C'} + \underline{\mathcal{F}}^D = \underline{\mathcal{F}}^{ext} \quad (4.89)$$

where the inertial force vector $\underline{\mathcal{F}}^I$ is split into $\underline{\mathcal{F}}^A$ and $\underline{\mathcal{F}}^V$:

$$\begin{aligned} \underline{\mathcal{F}}^A &= \begin{Bmatrix} m\ddot{\underline{u}} + \dot{\omega}m\underline{\eta} \\ m\tilde{\eta}\ddot{\underline{u}} + \underline{\rho}\dot{\underline{\omega}} \end{Bmatrix} \\ \underline{\mathcal{F}}^V &= \begin{Bmatrix} \tilde{\omega}\tilde{\omega}m\underline{\eta} \\ \tilde{\omega}\underline{\rho}\underline{\omega} \end{Bmatrix} \end{aligned} \quad (4.90)$$

The blade loads are thus defined as

$$\underline{\mathcal{F}}^{BF} \equiv \underline{\mathcal{F}}^V - \underline{\mathcal{F}}^{C'} + \underline{\mathcal{F}}^D \quad (4.91)$$

We note that if structural damping is considered in the analysis, the $\underline{\mathcal{F}}_d^C$ and $\underline{\mathcal{F}}_d^D$ are incorporated into the internal elastic forces, $\underline{\mathcal{F}}^C$ and $\underline{\mathcal{F}}^D$, for calculation.

Future Work

The following list contains future work on BeamDyn software:

- Eliminating numerical problems in single precision.
- Implementing eigenvalue analysis.
- Improving input options for stand-alone version to make it more user-friendly.
- Implementing GEBT based on modal method for computational efficiency.
- Adding more options for blade cross-sectional properties inputs. For example, for general isotropic beams, engineering parameters including sectional offsets, material properties, etc will be used to generate the 6×6 matrices needed by BeamDyn.
- Writing a general guidance on modeling composite beam structures using BeamDyn, , for example, how to select a time step, how to select the model discretization, how to define the blade reference axis, where to get 6x6 mass/stiffness matrices, etc.
- Extending applications in FAST to other slender structures in the wind turbine system, for example, tower, mooring lines, and shaft.
- Developing a simplified form of GEBT with only rotational DOFs (bending, torsion) for computational efficiency.

Appendix

BeamDyn Input Files

In this appendix we describe the BeamDyn input-file structure and provide examples for the NREL 5MW Reference Wind Turbine.

OpenFAST+BeamDyn and stand-alone BeamDyn (static and dynamic) simulations all require two files:

1) BeamDyn primary input file (NREL 5MW static example): This file includes information on the numerical-solution parameters (e.g., numerical damping, quadrature rules), and the geometric definition of the beam reference line via “members” and “key points”. This file also specifies the “blade input file.”

2) BeamDyn blade input file (NREL 5MW example):

Stand-alone BeamDyn simulation also require a driver input file; we list here examples for static and dynamic simulations:

3a) BeamDyn driver for dynamic simulations (NREL 5MW example): This file specifies the inputs for a single blade (e.g., forces, orientations, root velocity) and specifies the BeamDyn primary input file.

3b) BeamDyn driver for static simulations (NREL 5MW example): Same as above but for static analysis.

BeamDyn List of Output Channels

This is a list of all possible output parameters for the BeamDyn module. The names are grouped by meaning, but can be ordered in the OUTPUTS section of the BeamDyn primary input file as the user sees fit. $N\beta$, refers to output node β , where β is a number in the range [1,9], corresponding to entry β in the OutNd list. When coupled to FAST, “ $B\alpha$ ” is prefixed to each output name, where α is a number in the range [1,3], corresponding to the blade number. The outputs are expressed in one of the following three coordinate systems:

- **r**: a floating reference coordinate system fixed to the root of the moving beam; when coupled to FAST for blades, this is equivalent to the IEC blade (b) coordinate system.
- **l**: a floating coordinate system local to the deflected beam.
- **g**: the global inertial frame coordinate system; when coupled to FAST, this is equivalent to FAST’s global inertial frame (i) coordinate system.

4.2.5 SubDyn User Guide and Theory Manual

Introduction

SubDyn is a time-domain structural-dynamics module for multimember fixed-bottom substructures created by the National Renewable Energy Laboratory (NREL) through U.S. Department of Energy Wind and Water Power Program support. The module has been coupled into the FAST aero-hydro-servo-elastic computer-aided engineering (CAE) tool. Substructure types supported by SubDyn include monopiles, tripods, jackets, and other non-floating lattice-type substructures common for offshore wind installations in shallow and transitional water depths. SubDyn can also be used to model lattice support structures for land-based wind turbines.

The new SubDyn module follows the requirements of the FAST modularization framework, couples to OpenFAST, and provides new capabilities (relative to prior released versions of the software) for modeling the dynamic loading on multimember substructures. (Refer to Appendix E and the *changelog.txt* file that is provided in the archives for more details about changes among different versions.) SubDyn can also be driven as a standalone code to compute the mode shapes, natural frequencies, and time-domain responses of substructures under prescribed motion at the interface to the tower, uncoupled from FAST and in the absence of external loading other than gravity.

Channel Name(s)	Units	Description
RootFxr, RootFyr, RootFzr	(N), (N), (N)	Root reaction forces expressed in r
RootMxr, RootMyr, RootMzr	(N m), (N m), (N m)	Root reaction moments expressed in r
TipTDxr, TipTDyr, TipTDzr	(m), (m), (m)	Tip translational deflection (relative to the undeflected position) expressed in r
TipRDxr, TipRDyr, TipRDzr	(-), (-), (-)	Tip angular/rotational deflection Wiener-Milenković parameter (relative to the undeflected orientation) expressed in r
TipTVXg, TipTVYg, TipTVZg	(m/s), (m/s), (m/s)	Tip translational velocities (absolute) expressed in g
TipRVXg, TipRVYg, TipRVZg	(deg/s), (deg/s), (deg/s)	Tip angular/rotational velocities (absolute) expressed in g
TipTAXg, TipTAYg, TipTAZg	(m/s ²), (m/s ²), (m/s ²)	Tip translational accelerations (absolute) expressed in g
TipRAXg, TipRAYg, TipRAZg	(deg/s ²), (deg/s ²), (deg/s ²)	Tip angular/rotational accelerations (absolute) expressed in g
NβFxl, NβFyl, NβFzl	(N), (N), (N)	Sectional force resultants at Nβ expressed in l
NβMxl, NβMy1, NβMzl	(N m), (N m), (N m)	Sectional moment resultants at Nβ expressed in l
NβTDxr, NβTDyr, NβTDzr	(m), (m), (m)	Sectional translational deflection (relative to the undeflected position) at Nβ expressed in r
NβRDxr, NβRDyr, NβRDzr	(-), (-), (-)	Sectional angular/rotational deflection Wiener-Milenković parameter (relative to the undeflected orientation) at Nβ expressed in r
NβTVXg, NβTVYg, NβTVZg	(m/s), (m/s), (m/s)	Sectional translational velocities (absolute) at Nβ expressed in g
NβRVXg, NβRVYg, NβRVZg	(deg/s), (deg/s), (deg/s)	Sectional angular/rotational velocities (absolute) at Nβ expressed in g
NβTAXg, NβTAYg, NβTAZg	(m/s ²), (m/s ²), (m/s ²)	Sectional translational accelerations (absolute) at Nβ expressed in g
NβRAXg, NβRAYg, NβRAZg	(deg/s ²), (deg/s ²), (deg/s ²)	Sectional angular/rotational accelerations (absolute) at Nβ expressed in g
NβPFxl, NβPFyl, NβPFzl	(N), (N), (N)	Applied point forces at Nβ expressed in l
NβPMxl, NβPMyl, NβPMzl	(N m), (N m), (N m)	Applied point moments at Nβ expressed in l
NβDFxl, NβDFyl, NβDFzl	(N/m), (N/m), (N/m)	Applied distributed forces at Nβ expressed in l
NβDMxl, NβDMyl, NβDMzl	(N m/m), (N m/m), (N m/m)	Applied distributed moments at Nβ expressed in l

Fig. 4.32: BeamDyn Output Channel List

SubDyn relies on two main engineering schematizations: (1) a linear frame finite-element beam model (LFEB), and (2) a dynamics system reduction via the Craig-Bampton(C-B) method, together with a static-improvement method (SIM), greatly reducing the number of modes needed to obtain an accurate solution. More details can be found in Section 6, and in [SDRJ13], [DS13], [DSRJ13], [JBH+20].

In SubDyn, the substructure is considered to be either clamped or supported by springs at the seabed, and rigidly connected to the transition piece (TP) at the substructure top nodes (interface nodes). The spring constants are provided by the user to simulate soil-structure-interaction (SSI). Other restraint formulations may be implemented in the future. Only the substructure structural dynamics are intended to be modeled within SubDyn. When integrated with FAST, the structural dynamics of the TP, tower, and rotor-nacelle assembly (RNA) are modeled within FAST's ElastoDyn module and hydrodynamics are modeled within FAST's HydroDyn module. For full lattice support structures or other structures with no transition piece, however, the entire support structure up to the yaw bearing may be modeled within SubDyn. Modeling the tower in SubDyn as opposed to ElastoDyn, for example, allows for the possibility of including more than the first two fore-aft and side-to-side bending modes, thus accounting for more general flexibility of the tower and its segments. However, for tubular towers, the structural model in ElastoDyn tends to be more accurate because ElastoDyn considers geometric nonlinearities not treated in SubDyn.

Loads and responses are transferred between SubDyn, HydroDyn, and ElastoDyn via the FAST driver program (glue code) to enable hydro-elastic interaction at each coupling time step. At the interface nodes, the TP six degree-of-freedom (DOF) displacements (three translations and three rotations), velocities, and accelerations are inputs to SubDyn from ElastoDyn; and the six reaction loads at the TP (three forces and three moments) are outputs from SubDyn to ElastoDyn. SubDyn also outputs the local substructure displacements, velocities, and accelerations to HydroDyn in order to calculate the local hydrodynamic loads that become inputs for SubDyn. In addition, SubDyn can calculate member internal reaction loads, as requested by the user (see Figure 1).

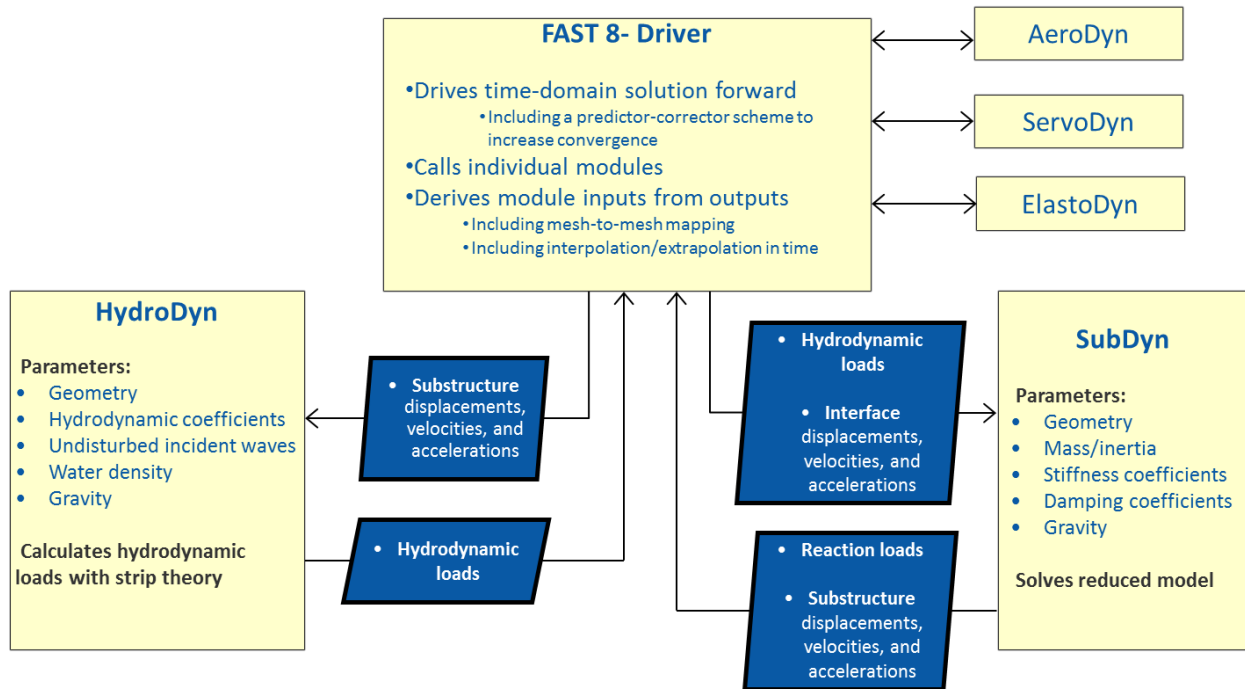


Fig. 4.33: SubDyn, HydroDyn, and FAST 8 coupled interaction

The input file defines the substructure geometry, material properties, restraints and SSI data files, finite-element resolution, number of retained modes in the dynamics system reduction, modal damping coefficients, and auxiliary parameters. The geometry is defined by joint coordinates in the global reference system (inertial-frame coordinate system shown in), with the origin at the intersection of the undeflected tower centerline with mean sea level (MSL) or ground level for land-based structures. A member connects two joints; multiple members may use a common joint. Nodes are

the result of the member refinement into multiple (***NDiv*** input parameter) elements (nodes are located at the ends of each element, as shown in), and they are calculated by the module.

In the current release, the geometry of a member is defined by its outer diameter and wall thickness (assuming a tubular geometry), and the material properties are defined by its Young's modulus, shear modulus, and mass density. Member properties are specified at the joints; if properties change from one joint to the other, they will be linearly interpolated for the inner elements. Thus, a tapered member will be treated as a cylindrical member with step-wise variation of its properties. In a future release, a tapered finite-element formulation will be implemented, and a more accurate representation of a tapered member will become available.

The hydrodynamic loads (including buoyancy) are computed by HydroDyn and transferred by the glue code at those nodes that are underwater (submerged nodes). Additionally, the self-weight distributed load components (from gravity) are calculated by SubDyn and applied at all the nodes. Note that other load and inertial properties may be input via the HydroDyn module input file, where marine growth and flooding/ballasting of the members can be specified.

This document is organized as follows. Section [Running SubDyn](#) details how to obtain the SubDyn and FAST software archives and run either the stand-alone version of SubDyn or SubDyn coupled to FAST. Section [Input Files](#) describes the SubDyn input files. Section 4 discusses the [Output Files](#) generated by SubDyn; these include echo files, a summary file, and the results file. Section 5 provides modeling guidance when using SubDyn. The SubDyn theory is covered in Section [SubDyn Theory](#). Section [Known Limitations and Future Work](#) outlines future work, and Section 8 contains a list of references. Example input files are shown in Appendices [Section 4.2.5](#) and B. A summary of available output channels are found in Appendix [Appendix D. List of Output Channels](#). Instructions for compiling the stand-alone SubDyn program are detailed in Appendix D. Appendix E tracks the major changes that have been made to SubDyn for each public release.

Running SubDyn

This section discusses how to obtain and execute SubDyn from a personal computer. Both the stand-alone version and the FAST-coupled version of the software are considered.

Downloading the SubDyn Software

There are two forms of the SubDyn software to choose from: stand alone and coupled to the FAST simulator. Although the user may not necessarily need both forms, he/she would likely need to be familiar with and run the stand-alone model if building a model of the substructure from scratch. The stand-alone version is also helpful for model troubleshooting and may benefit users who are interested in conducting aero-hydro-servo-elastic simulations of an offshore wind turbine.

Users can refer to the OpenFAST installation to download and compile SubDyn.

Running SubDyn

Running the Stand-Alone SubDyn Program

The stand-alone SubDyn program, *SubDyn_win32.exe*, simulates substructure dynamic responses of the user's input model, without coupling to FAST. Unlike the coupled version, the stand-alone software requires the use of a driver file in addition to the primary SubDyn input file. This driver file specifies inputs normally provided to SubDyn by FAST, including motions of the TP reference point. Both the SubDyn summary file and the results output file are available when using the stand-alone SubDyn (see Section 4 for more information regarding the SubDyn output files).

Run the standalone SubDyn software from a DOS command prompt by typing, for example:

```
>SubDyn_win32.exe MyDriverFile.dvr
```

where, *MyDriverFile.dvr* is the name of the SubDyn driver file, as described in [Section 4.2.5](#). The SubDyn primary input file is described in [Section 4.2.5](#).

Running SubDyn Coupled to FAST

Run the coupled FAST software from a DOS command prompt by typing, for example:

```
>FAST_Win32.exe Test21.fst
```

where, *Test21.fst* is the name of the primary FAST input file. This input file has a feature switch to enable or disable the SubDyn capabilities within FAST, and a corresponding reference to the SubDyn input file. See the documentation supplied with FAST for further information.

Input Files

The user specifies the substructure model parameters, including its geometry and properties, via a primary SubDyn input file. When used in stand-alone mode, an additional driver input file is required. This driver file specifies inputs normally provided to SubDyn by FAST, including motions of the TP reference point.

No lines should be added or removed from the input files, except in tables where the number of rows is specified.

Additional input files containing soil-structure information (*SSIfile*) can be provided by the user specifying their paths in the main SubDyn input file under the section titled *BASE REACTION JOINTS*.

Units

SubDyn uses the SI system (kg, m, s, N). Angles are assumed to be in radians unless otherwise specified.

SubDyn Driver Input File

The driver input file is only needed for the stand-alone version of SubDyn and contains inputs that are normally set by FAST, and that are necessary to control the simulation for uncoupled models. It is possible to provide per-time-step inputs to SubDyn, even in stand-alone mode, by tying the driver file to an additional input file containing time-histories of the TP motion (displacements, velocities, and accelerations). A sample SubDyn driver input file is given in [Section 4.2.5](#).

Users can set the **Echo** flag in this file to TRUE so that *SubDyn_win32.exe* echoes the contents of the driver input file (useful for debugging errors in the driver file). The echo file has the naming convention of **OutRootName.dvr.ech**. **OutRootName** is specified in the SUBDYN section of the driver input file (see below).

Environmental conditions

Set the gravity constant using the **Gravity** parameter. SubDyn expects a magnitude, so in SI units this would be set to $9.80665 \frac{m}{s^2}$ for standard gravity. **WtrDpth** specifies the water depth (depth of the seabed), based on the reference MSL, and must be a value greater than zero.

SubDyn module inputs

SDInputFile is the file name of the primary SubDyn input file. This name should be in quotations and can contain an absolute path or a relative path. All SubDyn-generated output files will be prefixed with **OutRootName**. If this parameter includes a file path, the output will be generated in that folder. If this output is left empty, the driver filename is used (without the extension) is used. **NSteps** specifies the number of simulation time steps, and **TimeStep** specifies the time between steps. Next, the user must specify the location of the TP reference point **TP_RefPoint** (in the global reference system). This is normally set by FAST through the ElastoDyn input file, and it is the so-called *platform* reference point location. When coupled to FAST, the *platform* reference point location is identified by only one (Z) coordinate. The interface joints, defined in SubDyn's main input file, are rigidly connected to this reference point. To utilize the same geometry definition within SubDyn's main input file, while still allowing for different substructure orientations about the vertical, the user can set **SubRotateZ** to a prescribed angle in degrees with respect to the global Z-axis. The entire substructure will be rotated by that angle. (This feature is only available in stand-alone mode.)

Input motion

Setting **InputsMod** = 0 sets all TP reference-point input motions to zero for all time steps. Setting **InputsMod** = 1 allows the user to provide steady (fixed) inputs for the TP motion in the STEADY INPUTS section of the file—**uTPInSteady**, **uDotTPInSteady**, and **uDotDotTPInSteady** following the same convention as Table 1 (without time). Setting **InputsMod** = 2 allows the user to input a time-series file whose name is specified via the **InputsFile** parameter. The time-series input file is a text-formatted file. This file has no header lines, **NSteps** rows, and each i^{th} row has the first column showing time as $t = (i - 1) * \text{TimeStep}$ (the data will not be interpolated to other times). The remainder of each row is made of white-space-separated columns of floating point values representing the necessary motion inputs as shown in Table 1. All motions are specified in the global, inertial-frame coordinate system. SubDyn does not check for physical consistency between the displacement, velocity, and acceleration motions specified for the TP reference point in the driver file.

Table 1. TP Reference Point Inputs Time-Series Data File Contents

Column Number	Input	Units
1	Time step value	<i>s</i>
2-4	TP reference point translational displacements along X, Y, and Z	<i>m</i>
5-7	TP reference point rotational displacements about X, Y, and Z (small angle assumptions apply)	<i>rad/s</i>
8-10	TP reference point translational velocities along X, Y, and Z	<i>m/s</i>
11-13	TP reference point rotational velocities about X, Y, and Z	<i>rad/s</i>
14-16	TP reference point translational accelerations along X, Y, and Z	<i>m/s^2</i>
17-19	TP reference point rotational accelerations about X, Y, and Z	<i>rad/s^2</i>

Applied loads

The next section of the input file provides options to apply loads at given joints of the structure. **nAppliedLoads** [-] specifies the number of applied loads listed in the subsequent table. The user can specify a combination of steady loads and unsteady loads (both are added together). The loads are in the global coordinate system. The steady loads are given as columns of the table (Fx, Fy, Fz, Mx, My, Mz), whereas the unsteady loads are provided in a CSV file. The CSV filename is provided in the last entry of the table. If the filename is empty, the unsteady loads are not read. An example of applied loads table is given below:

----- LOADS -----
↪-----

(continues on next page)

(continued from previous page)

1 nAppliedLoads - Number of applied loads at given nodes							
ALJointID	Fx	Fy	Fz	Mx	My	Mz	UnsteadyFile
(-)	(N)	(N)	(N)	(Nm)	(Nm)	(Nm)	(-)
15	100	0	0	0	0	0	""
23	0	0	0	0	0	0	"Force_TS.csv"

In the above example, a steady applied force of 100N is applied at the joint with ID=15 of the structure, and an unsteady load is applied to joint 23. The time series of unsteady loads is a CSV file with 7 columns (Time, Fx, Fy, Fz, Mx, My, Mz) and one line of header. The time vector needs to be increasing, but does not need to be linear or cover the full range of the simulation. Interpolation is done in between time stamps, and the first and last values are used for times smaller and larger than the simulation time range respectively. An example of time series is shown below:

#Time_[s]	Fx_[N]	Fy_[N]	Fz_[N]	Mx_[Nm]	My_[Nm]	Mz_[Nm]
0.0	0.0	0.0	0.0	0.0	0.0	0.0
10.0	100.0	0.0	0.0	0.0	0.0	0.0
11.0	0.0	0.0	0.0	0.0	0.0	0.0

SubDyn Primary Input File

The SubDyn input file defines the substructure geometry, integration and simulation options, finite-element parameters, and output channels. The geometry of members is defined by joint coordinates of the undisplaced substructure in the global reference system (inertial-frame coordinate system), with the origin at the intersection of the undeflected tower centerline with MSL or ground level for land-based structures. A member connects two joints; multiple members can use a common joint. The hydrodynamic and gravity loads are applied at the nodes, which are the resultant of member refinement into multiple (**N**Div input) elements (nodes are located at the ends of each element), as calculated by the module. Member properties include outer diameter, thickness, material density, and Young's and shear moduli. Member properties are specified at the joints; if properties change from one joint to the other, they will be linearly interpolated for the inner nodes. Unlike the geometric properties, the material properties are not allowed to change within a single member.

Future releases will allow for members of different cross-sections, i.e., noncircular members. For this reason, the input file has (currently unused) sections dedicated to the identification of direction cosines that in the future will allow the module to identify the correct orientation of noncircular members. The current release only accepts tubular (circular) members.

The file is organized into several functional sections. Each section corresponds to an aspect of the SubDyn model and substructure.

If this manual refers to an ID in a table entry, it is an integer identifier for the table entry and must be unique for a given table entry.

A sample SubDyn primary input file is given in [Section 4.2.5](#).

The input file begins with two lines of header information, which is for the user but is not used by the software.

Simulation Control Parameters

Users can set the **Echo** flag to TRUE to have SubDyn echo the contents of the SubDyn input file (useful for debugging errors in the input file). The echo file has the naming convention of **OutRootName.SD.ech**. **OutRootName** is either specified in the SUBDYN section of the driver input file when running SubDyn standalone, or by FAST, when running a coupled simulation, from FAST's main input file.

SDDeltaT specifies the fixed time step of the integration in seconds. The keyword 'DEFAULT' may be used to indicate that the module should employ the time step prescribed by the driver code (FAST/standalone driver program).

IntMethod specifies the integration algorithm to use. There are four options: 1) Runge-Kutta 4th-order explicit (RK4); 2) Adams-Bashforth 4th-order explicit predictor (AB4); 3) Adams-Bashforth-Moulton 4th-order explicit predictor-corrector (ABM4); 4) Adams-Moulton implicit 2nd-order (AM2). See Section on how to properly select this and the previous parameter values.

SttcSolve is a flag that specifies whether the static improvement method (SIM, see [Section 4.2.5](#)) shall be employed. Through this method, all (higher frequency) modes that are not considered by the C-B reduction are treated quasi-statically. This treatment helps minimize the number of retained modes needed to capture effects such as static gravity and buoyancy loads, and high-frequency loads transferred from the turbine. Recommended to set to True.

GuyanLoadCorrection is a flag to specify whether the extra moment due to the lever arm from the Guyan deflection of the structure is to be added to the loads passed to SubDyn, and, whether the FEM representation should be expressed in the rotating frame in the floating case (the rotation is induced by the rigid body Guyan modes). See [Section 4.2.5](#) for details. Recommended to set to True.

FEA and Craig-Bampton Parameters

FEMMod specifies one of the following options for finite-element formulation: 1) Euler-Bernoulli; 3) Timoshenko. Tapered formulations (2 and 4) have yet to be implemented and will be available in a future release.

NDiv specifies the number of elements per member. Analysis nodes are located at the ends of elements and the number of analysis nodes per member equals **NDiv** + 1. **NDiv** is applied uniformly to all members regardless of the member's length, hence it could result in small elements in some members and long elements in other members. Increasing the number of elements per member may increase accuracy, with the trade-off of increased memory usage and computation time. We recommend using **NDiv** > 1 when modeling tapered members.

CBMod is a flag that specifies whether or not the C-B reduction should be carried out by the module. If FALSE, then the full finite-element model is retained and **Nmodes** is ignored.

Nmodes sets the number of internal C-B modal DOFs to retain in the C-B reduction. **Nmodes** = 0 corresponds to a Guyan (static) reduction. **Nmodes** is ignored if **CBMod** is set to FALSE, meaning the full finite-element model is retained by keeping all modes (i.e. a modal analysis is still done, and all the modes are used as DOFs).

JDampings specifies value(s) of damping coefficients as a percentage of critical damping for the retained C-B modes. Distinct damping coefficients for each retained mode should be listed on the same line, separated by white space. If the number of **JDampings** is less than the number of retained modes, the last value will be replicated for all the remaining modes. (see [Section 4.2.5](#))

GuyanDampMod Guyan damping [0=none, 1=Rayleigh Damping, 2= user specified 6x6 matrix] (see [Section 4.2.5](#))

RayleighDamp Mass and stiffness proportional damping coefficients ((α, β) Rayleigh damping) [only if GuyanDampMod=1] Guyan damping matrix (6x6) [only if GuyanDampMod=2] (see [Section 4.2.5](#))

Guyan damping matrix: The 6 lines following this input line consists of the 6x6 coefficients of the damping matrix to be applied at the interface. (see [Section 4.2.5](#))

For more information on these parameters and guidelines on how to set them, see [Sections Section 4.2.5](#) and [Section 4.2.5](#).

Structure Joints

The finite-element model is based on a substructure composed of joints interconnected by members. **NJoints** is the user-specified number of joints, and determines the number of rows in the subsequent table. Because a member connects two joints, **NJoints** must be greater than or equal to two. Each joint listed in the table is identified by a unique integer, **JointID**; each integer between one and **NJoints** must be present in the table, but they need not be sequential. The (X,Y,Z) coordinate of each joint is specified in the substructure (SS) coordinate system, which coincides with the global inertial-frame coordinate system via **JointXss**, **JointYss**, and **JointZss**, respectively. This version of SubDyn does not consider overlap when multiple members meet at a common joint, therefore, it tends to overestimate the total substructure mass. Member overlap and node offset calculations will be considered in a future release of SubDyn. The fifth column specifies the **JointType** (see [Section 4.2.5](#)):

- Cantilever joints (*JointType=1*)
- Universal joint (*JointType=2*)
- Pin joint (*JointType=3*)
- Ball joint (*JointType=4*)

The three following columns specify the vector coordinates of the direction around which rotation is free for a pin joints. The last column, **JointStiff** specify a value of additional stiffness to be added to the “free” rotational DOFs of Ball, Pin and Universal joints.

Note for HydroDyn coupling: modeling a fixed-bottom substructure embedded into the seabed (e.g., through piles or suction buckets) requires that the lowest member joint(s) in HydroDyn lie(s) below the water depth. Placing a joint at or above the water depth will result in static and dynamic pressure loads being applied at the joint. When SubDyn is coupled to FAST, the joints and members need not match between HydroDyn and SubDyn—FAST’s mesh-mapping utility handles transfer of motion and loads across meshes in a physically relevant manner (Sprague et al. 2014), but consistency between the joints and members in HydroDyn and SubDyn is advised.

An example of joint table is given below

3 NJoints - Number of joints (-)								
JointID	JointXss	JointYss	JointZss	JointType	JointDirX	JointDirY	JointDirZ	JointStiff
(-)	(m)	(m)	(m)	(-)	(-)	(-)	(-)	(Nm/rad)
101	0.0	0.0	50.0	1	0.0	0.0	0.0	0.0
111	0.0	0.0	10.0	2	0.0	1.0	0.0	100.0
102	0.0	0.0	-45.0	1	0.0	0.0	0.0	0.0

Base Reaction Joints

SubDyn requires the user to specify the boundary joints. **NReact** should be set equal to the number of joints (defined earlier) at the bottom of the structure (i.e., seabed) that are fully constrained; **NReact** also determines the number of rows in the subsequent table. In SubDyn, **NReact** must be greater than or equal to one. Each joint listed in the table is identified by a unique integer, **RJointID**, which must correspond to the **JointID** value found in the STRUCTURE JOINTS table. The flags **RctTDXss**, **RctTDYss**, **RctTDZss**, **RctRDXss**, **RctRDYss**, **RctRDZss** indicate the fixity value for the three translations (TD) and three rotations (RD) in the SS coordinate system (global inertial-frame coordinate system). One denotes fixed and zero denotes free (instead of TRUE/FALSE). **SSIfile** points to the relative path and filename for an SSI information file. This version of SubDyn can, in fact, handle partially restrained joints by setting one or more DOF flags to 0 and providing the appropriate stiffness and mass matrix elements for that DOF via the **SSIfile**. If a DOF flag is set to 1, then the node DOF is considered restrained and the associated matrix elements potentially provided in the **SSIfile** will be ignored.

An example of base reaction and interface table is given below

----- BASE REACTION JOINTS							
1	NReact	- Number of Joints with reaction forces					
RJointID	RctTDXss	RctTDYss	RctTDZss	RctRDXss	RctRDYss	RctRDZss	SSIfile
(-)	(flag)	(flag)	(flag)	(flag)	(flag)	(flag)	(string)
61	1	1	1	1	1	1	"SSI.txt"
----- INTERFACE JOINTS							
1	NInterf	- Number of interface joints locked to the Transition Piece (TP)					
IJointID	ItfTDXss	ItfTDYss	ItfTDZss	ItfRDXss	ItfRDYss	ItfRDZss	
(-)	(flag)	(flag)	(flag)	(flag)	(flag)	(flag)	
24	1	1	1	1	1	1	

Interface Joints

SubDyn requires the user to specify the interface joints. **NInterf** should be set equal to the number of joints at the top of the structure (i.e., TP); **NInterf** also determines the number of rows in the subsequent table. In SubDyn, **NInterf** must be greater than or equal to one. Note that these joints will be assumed to be rigidly connected to the platform reference point of ElastoDyn (see FAST documentation) when coupled to FAST, or to the TP reference point if SubDyn is run in stand-alone mode. Each joint listed in the table is identified by a unique integer, **IJointID**, which must correspond to the *JointID* value found in the STRUCTURE JOINTS table. The flags **ItfTDXss**, **ItfTDYss**, **ItfTDZss**, **ItfRDXss**, **ItfRDYss**, **ItfRDZss** indicate the fixity value for the three translations (TD) and three rotations (RD) in the SS coordinate system (global inertial-frame coordinate system). One denotes fixed and zero denotes free (instead of TRUE/FALSE). This version of SubDyn cannot handle partially restrained joints, so all flags must be set to one; different degrees of fixity will be considered in a future release.

Members

NMembers is the user-specified number of members and determines the number of rows in the subsequent table. Each member listed in the table is identified by a unique integer, **MemberID**. Each integer between one and **NMembers** must be present in the table, but they need not be sequential. For each member distinguished by **MemberID**, **MJointID1** specifies the starting joint and **MJointID2** specifies the ending joint, corresponding to an identifier (**JointID**) from the STRUCTURE JOINTS table. Likewise, **MPropSetID1** corresponds to the identifier **PropSetID** from the MEMBER X-SECTION PROPERTY table (discussed next) for starting cross-section properties and **MPropSetID2** specifies the identifier for ending cross-section properties, allowing for tapered members. The sixth column specify the member type **MType**. A member is one of the three following types (see [Section 4.2.5](#)):

- Beams (*MType=1*), Euler-Bernoulli (*FEMMod=1*) or Timoshenko (*FEMMod=3*)
- Pretension cables (*MType=2*)
- Rigid link (*MType=3*)

COSMID refers to the IDs of the members' cosine matrices for noncircular members; the current release ignores this column.

An example of member table is given below

2	NMembers	- Number of frame members				
MemberID	MJointID1	MJointID2	MPropSetID1	MPropSetID2	MType	COSMID
(-)	(-)	(-)	(-)	(-)	(-)	(-)
10	101	102	2	2	1	
11	102	103	2	2	1	

Member Cross-Section Properties

Members in SubDyn are assumed to be straight, circular, possibly tapered, and hollow cylinders. Future releases will allow for generic cross-sections to be employed. These special cross-section members will be defined in the second of two tables in the input file (Member X-Section Property data 2/2), which is currently ignored.

For the circular cross-section members, properties needed by SubDyn are material Young's modulus, **YoungE**, shear modulus, **ShearG**, and density, **MatDens**, member outer diameter, **XsecD**, and member thickness, **XsecT**. Users will need to create an entry in the first table within this section of the input file distinguished by **PropSetID**, for each unique combination of these five properties. The member property-set table contains **NPropSets** rows. The member property sets are referred to by their **PropSetID** in the MEMBERS table, as described in Section . Note, however, that although diameter and thickness will be linearly interpolated within an individual member, SubDyn will not allow *material* properties to change within an individual member.

The second table in this section of the input file (not to be used in this release) will have **NXPropSets** rows (assumed to be zero for this release), and have additional entries when compared to the previous table, including: cross-sectional area (**XsecA**), cross-sectional shear area along the local principal axes x and y (**XsecAsx**, **XsecAsy**), cross-sectional area second moment of inertia about x and y (**XsecJxx**, **XsecJyy**), and cross-sectional area polar moment of inertia (**XsecJ0**). The member cosine matrix section (see Section) will help determine the correct orientation of the members within the assembly.

Cable Properties

Members that are specified as pretension cables (**MType=2**), have their properties defined in the cable properties table. The table lists for each cable property: the property ID (**PropSetID**), the cable tension stiffness (**EA**), the material density (**MatDens**), the pretension force (**T0**), and the control channel (**CtrlChannel**). The control channel is only used if ServoDyn provides dedicated control signal, in which case the cable tension (given in terms of a length change Δl) is dynamically changed (see Section 4.2.5). The FEM representation of pretension cable is given in Section 4.2.5.

An example of cable properties table is given below:

----- CABLE PROPERTIES -----				
	2	NCablePropSets - Number of cable cable properties		
PropSetID	EA	MatDens	T0	CtrlChannel
(-)	(N)	(kg/m)	(N)	(-)
11	210E7	7850.0	2E7	1
10	210E7	7850.0	1E7	0

Rigid link Properties

Members that are specified as rigid links (**MType=3**), have their properties defined in the rigid link properties table. The table lists the material density (**MatDens**) for each rigid link property. The FEM representation of rigid links is given in Section 4.2.5.

An example of rigid link properties table is given below

----- RIGID LINK PROPERTIES -----	
	1 NRigidPropSets - Number of rigid link properties
PropSetID	MatDens
(-)	(kg/m)
12	7850.0
3	7000.0

Member Cosine Matrices COSM (i,j)

This table is not currently used by SubDyn, but in future releases it will need to be populated if members with cross-sections other than circular will be employed.

NCOSMs rows, one for each unique member orientation set, will need to be provided. Each row of the table will list the nine entries of the direction cosine matrices (COSM11, COSM12, ... COSM33) for matrix elements (1,1), (1,2), ... (3,3) that establish the orientation of the local member axes (x,y principal axes in the cross-sectional plane, z along the member longitudinal axis) with respect to the SS coordinate system (local-to-global transformation matrices).

Joint Additional Concentrated Masses

SubDyn can accept **NCmass** lumped masses/inertias defined at the joints. The subsequent table will have **NCmass** rows, in which for each joint distinguished by **CMJointID** (corresponding to an identifier, **JointID**, from the STRUCTURE JOINTS table), **JMass** specifies the lumped mass value, and **JMXX**, **JMYX**, **JMZZ** specify the mass second moments of inertia with respect to the SS coordinate system (not the element system). Latest version of SubDyn accept 6 additional columns (**JMXY**, **JMXZ**, **JMYZ**, **MCGX**, **MCGY**, **MCGZ**) to specify off-diagonal terms.

The additional mass matrix added to the node is computed in the SS system as follows:

$$M_{\text{add}} = \begin{bmatrix} m & 0 & 0 & 0 & zm & -ym \\ 0 & m & 0 & -zm & 0 & xm \\ 0 & 0 & m & ym & -xm & 0 \\ 0 & -zm & ym & J_{xx} + m(y^2 + z^2) & J_{xy} - mxy & J_{xz} - mxz \\ zm & 0 & -xm & J_{xy} - mxy & J_{yy} + m(x^2 + z^2) & J_{yz} - myz \\ -ym & xm & 0 & J_{xz} - mxz & J_{yz} - myz & J_{zz} + m(x^2 + y^2) \end{bmatrix}$$

with m the parameter **JMass**, and x, y, z , the CG offsets.

An example of concentrated mass table is given below:

2 NCmass - Number of joints with concentrated masses; (SS coord system)										
CMJointID	JMass	JMXX	JMYX	JMZZ	JMXY	JMXZ	JMYZ	MCGX	MCGY	MCGZ
(-)	(kg)	(kgm ²)	(kgm ²)	(kgm ²)	(kgm ²)	(kgm ²)	(kgm ²)	(m)	(m)	(m)
1	4090	0	0	0	0	0	0	0	0	0
3	4.2e6	0	0	3.3e9	0	0	0	0	0	0

Output: Summary and Outfile

In this section of the input file, the user sets flags and switches for the desired output behavior.

Specifying **SDSum** = TRUE causes SubDyn to generate a summary file with name **OutRootName.SD.sum***. **OutRootName** is either specified in the SUBDYN section of the driver input file when running SubDyn in stand-alone mode, or in the FAST input file when running a coupled simulation. See Section 4.2 for summary file details.

The following two inputs specified whether mode shapes should be written to disk. **OutCBModes** is a flag that controls the output of the Guyan and Craig-Bampton modes. Similarly, **OutFEMModes**, controls the output of the FEM modes (full system with constraints prior to the CB-reduction). For now, only the first 30 FEM modes are written to disk, but all CB modes selected by the users are written. For both inputs, the following options are available: 0, no output, 1, outputs in JSON format. The JSON files contain nodes coordinates, connectivity between the nodes, displacements for each modes and nodes, and frequencies for each modes. The reading of these files should be straightforward using Matlab or Python using a JSON format parser. The files can be opened to visualize the modes using the tool viz3danim (see the [live version](#) , or its [github repository](#)).

Currently, **OutCOSM** is ignored. In future releases, specifying **OutCOSM** = TRUE will cause SubDyn to include direction cosine matrices (undeflected) in the summary file for only those members requested in the list of output channels.

Specifying **OutAll** = TRUE causes SubDyn to output forces and moments at all of the joints (not internal nodes). That is, the static (elastic) and dynamic (inertia) components of the three forces and three moments at the end node of each member connected to a given joint are output for all joints. These outputs are included within the **OutRootName.SD.out*** output file in addition to those directly specified through the output channels section below.

If **OutSwitch** is set to one, outputs are sent to a file with the name **OutRootName.SD.out***. If **OutSwitch** is set to two, outputs are sent to the calling program (FAST) for writing in its main output file (not available in stand-alone mode). If **OutSwitch** is set to three, both file outputs occur. In stand-alone mode, setting **OutSwitch** to two results in no output file being produced.

If **TabDelim** is set to TRUE and **OutSwitch** is set to one, the output file **OutRootName.SD.out*** will be tab-delimited.

With **OutDec** set to an integer value greater than one, the output file data rate will be decimated, and only every **OutDec**-th value will be written to the file. This applies only to SubDyn's output file (**OutRootName.SD.out***)—not FAST's.

The **OutFmt** and **OutSFmt** parameters control the formatting of SubDyn's output file for the output data and the channel headers, respectively. SubDyn currently does not check the validity of these format strings. They need to be valid Fortran format strings. **OutSFmt** is used for the column header and **OutFmt** is used for the channel data. Therefore, in order for the headers and channel data to align properly, the width specification should match. For example:

"ES11.4" OutFmt

"A11" OutSFmt.

Member Output List

SubDyn can output load and kinematic quantities at up to nine locations for up to nine different members, for a total of 81 possible local member output locations. **NMOutputs** specifies the number of members that output is requested for. The user must create a table entry for each requested member. Within a row of this table, **MemberID** is the ID specified in the MEMBERS table, and **NOutCnt** specifies how many nodes along the member will generate output. **NodeCnt** specifies those node numbers (a separate entry on the same line for each node) for output as an integer index from the start-joint (node 1) to the end-joint (node **NDiv** + 1) of the member. The outputs specified in the SDOutList section determines which quantities are actually output at these locations.

Output Channels- SDOutList Section

This section specifies which quantities are output by SubDyn. Enter one or more lines containing quoted strings that in turn contain one or more output parameter names. Separate output parameter names by any combination of commas, semicolons, spaces, and/or tabs. If a parameter name is prefixed with a minus sign, "-", underscore, "_", or the characters "m" or "M", SubDyn will multiply the value for that channel by -1 before writing the data. The parameters are written in the order they are listed in the input file. SubDyn allows the use of multiple lines so that users can break their lists into meaningful groups and so the lines can be shorter. Comments may also be entered after the closing quote on any of the lines. Entering a line with the string "END" at the beginning of the line or at the beginning of a quoted string found at the beginning of the line will cause SubDyn to quit scanning for more lines of channel names. Modal kinematics and member-node-, base-, and interface-related kinematic and load quantities can be selected. Member-node-related data follow the organization described in Section . If SubDyn encounters an unknown/invalid channel name, it prints an error message and halts execution. Please refer to [Section 4.2.5](#) for a complete list of possible output parameters and their names.

SSI Input File

Individual SSI files (*SSIfiles*) can be provided for each restrained node, therefore the maximum number of SSIfiles is **NReact**. In an SSIfile, up to 21 elements for the SSI mass matrix and up to 21 SSI stiffness matrix elements can be provided. The mass and stiffness elements account for both pile and soil effects. No additional damping can be provided at this point.

The order of the elements is not important, because each element value is accompanied by a string label that identifies the actual element. The stiffness matrix accepted labels are: 'Kxx', 'Kxy', 'Kyy', 'Kxz', 'Kyz', 'Kzz', 'Kxtx', 'Kytx', 'Kztx', 'Ktxtx', 'Kxty', 'Kyty', 'Kzty', 'Ktxty', 'Ktyty', 'Kxtz', 'Kytz', 'Kztz', 'Ktxtz', 'Ktytz', 'Kztztz'.

If any matrix element is not provided it will be set to infinity (i.e., machine 'huge') by default.

For the mass matrix the accepted labels are: 'Mxx', 'Mxy', 'Myx', 'Mxz', 'Myz', 'Mzz', 'Mxtx', 'Mytx', 'Mztx', 'Mtxtx', 'Mxty', 'Myty', 'Mzty', 'Mtxty', 'Mtyty', 'Mxtz', 'Mytz', 'Mztz', 'Mtxtz', 'Mtytz', 'Mztztz'. If any matrix element is not provided it will be set to 0 by default. The labels contain 'K' or 'M' to specify stiffness or mass matrix elements, and then the directions they apply to, e.g., 'Kxy' refers to the force along x due to a unit displacement along y; the 't' refers to the rotation about one of the 'x', 'y', or 'z' axes in the global coordinate system.

Units are in SI system (N/m; N/m/rad; Nm/rad, Kg, kgm, kgm2).

Note that by selecting fixities of 1 in the various DOFs of the restrained nodes, the columns and rows associated with those DOFs will be removed, therefore the associated matrix elements will be ignored.

A sample SubDyn SSI input file is given in [Section 4.2.5](#).

Output Files

SubDyn produces three types of output files: an echo file, a summary file, and a time-series results file. The following sections detail the purpose and contents of these files.

Echo File

If the user sets the **Echo** flag to TRUE in the SubDyn driver file or the primary SubDyn input file, the contents of those files will be echoed to a file with the naming conventions, **OutRootName.dvr.ech** for the driver input file and **OutRootName.SD.ech** for the primary SubDyn input file. **OutRootName** is either specified in the SUBDYN section of the driver input file, or in the FAST input file. The echo files are helpful for debugging the input files. The contents of an echo file will be truncated if SubDyn encounters an error while parsing an input file. The error usually corresponds to the line after the last successfully echoed line.

Summary File

SubDyn generates a summary file with the naming convention, **OutRootName.SD.sum** if the **SDSum** parameter is set to TRUE. This file summarizes key information about the substructure model, including:

- Undisplaced node geometry: a list of all of the (**NNodes**) nodes and the X,Y,Z coordinates in the global SS coordinate system. Note that **NNodes** may be greater or equal to **NJoints**, depending on **NDiv** (primary input file parameters).
- Element connectivity and properties at end nodes: a list of all (**NElems**) elements, the start and end nodes (**Node_I**, **Node_J**) and the ID of the property set (**Prop_I**, **Prop_J**) at the start and end nodes. **NElems** may be greater or equal to **NMembers**, depending on **NDiv** (primary input file parameters).
- Property sets. If tapered members are used, additional property sets may be included beyond those specified in the main input file, based on interpolated diameter and thickness values. Headers and their meanings are identical to those described in Section .

- Reaction DOFs and interface DOFs and their associated fixity; the actual indices of the DOFs (**DOF_ID**) associated with reaction and interface nodes are listed together with the (1/0) flag to distinguish the fixity level.
- Concentrated mass schedule. This is an echo of the equivalent section in the primary input file. Refer to Section .
- Member schedule including connectivity to joints, nodes, and their masses. A table lists all of the members by identifier (**MemberID**), with their start and end nodes (**Joint1_ID**, **Joint2_ID**), associated mass (**Mass**), and list of node identifiers along the length of the members.
- Direction cosine matrices for the members. Each row (columns 2-10) corresponds to the direction cosine matrix entries (**DC(1,1)** through **DC(3,3)**) for the member whose identifier is listed in the first column. The direction cosine matrices specify the transformation from the global reference to the local coordinate system for each member.
- Sorted eigenfrequencies [in Hertz (Hz)] for the full substructural system (neglecting a possible coupling to ElastoDyn through FAST), assuming the TP reference point is a free end. There are a total of **NDOFs** eigenfrequencies and eigenvectors.
- Sorted eigenfrequencies (in Hz) for the C-B reduced system, assuming the TP reference point is a fixed end. There are a total of **Nnodes** C-B reduced eigenfrequencies and eigenvectors.
- Full substructural system eigenvectors. Each column represents an eigenvector associated with the corresponding eigenfrequency identified previously in the file.
- C-B reduced system eigenvectors (**PhiM** matrix). Each column represents an eigenvector associated with the corresponding eigenfrequency identified previously in the file.
- **PhiR** matrix or displacements of the internal nodes caused by unit rigid body motions of the interface DOFs (see Section). Each column of the matrix represents the internal DOF displacements for a given unit rigid-body motion along an interface DOF for each base and interface joint.
- Substructure equivalent stiffness and mass matrices referred to the TP reference point (**KBBt** and **MBBt**), based on a Guyan reduction. These are useful to calculate effects of substructure flexibility while calculating tower eigenmodes for ElastoDyn.
- Rigid-body-equivalent mass matrix relative to global origin (**MRB**); a 6x6 mass matrix.
- Substructure total (dry) mass.
- Substructure center of mass coordinates in the global coordinate system.

The various sections of the summary file and variables are self-explanatory and easily identifiable in the file.

Results File

The SubDyn time-series results are written to a text-based file with the naming convention **OutRootName.SD.out** when **OutSwitch** is set to either one or three. If SubDyn is coupled to FAST and **OutSwitch** is set to two or three, then FAST will generate a master results file that includes the SubDyn results. The results in **OutRootName.SD.out** are in table format, where each column is a data channel (the first column always being the simulation time), and each row corresponds to a simulation time step. The data channels are specified in the **SDOutList** section of the input file. The column format of the SubDyn-generated file is specified using the **OutFmt** and **OutSFmt** parameters of the input file.

Modeling Considerations

SubDyn was designed as a flexible tool for modeling a wide range of substructures for both land-based and offshore applications. This section provides some general guidance to help construct models that are compatible with SubDyn.

Please refer to the theory in Section 6 for detailed information about SubDyn’s coordinate systems, and the theoretical approach we have followed in SubDyn.

Model Discretization

SubDyn allows for the specification of arbitrary multimember structure geometries. The user defines the geometry of a structure in SubDyn using joints and members. Specifically, the user specifies a list of joints that represent the endpoints of beams, and the connectivity between one or more members at each joint. Members and their cross-sectional properties are then defined between two joints. Members can be further subdivided into multiple (**NDiv**) elements to increase the model resolution. Nodes, where the numerical calculations take place, are located at the endpoints of each element. To keep the mesh as uniform as possible when using **NDiv**, the initial member definition should also have a roughly uniform mesh. For tapered members, we recommend setting **NDiv** > 1. Improper discretization of the members may decrease the accuracy of the model.

When SubDyn is coupled to FAST, the joints and members need not match between HydroDyn and SubDyn—FAST’s mesh-mapping utility handles the transfer of motion and loads across meshes in a physically relevant manner [MJJ14], but consistency between the joints and members in HydroDyn and SubDyn is advised.

For offshore applications, because of the exponential decay of hydrodynamic loads with depth, HydroDyn requires higher resolution near the water free surface to properly capture loads as waves oscillate about the still water level (SWL). We recommend that the HydroDyn discretization not exceed element lengths of 0.5 m in the region of the free surface (5 to 10 m above and below SWL), 1.0 m between 25- and 50-m depth, and 2.0 m in deeper waters.

When SubDyn is hydro-elastically coupled to HydroDyn through FAST for the analysis of fixed-bottom offshore systems, we recommend that the length ratio between elements of SubDyn and HydroDyn not exceed 10 to 1. As such, we recommend that the SubDyn discretization not exceed element lengths of 5 m in the region of the free surface, 10 m down to 25- to 50-m depth, and 20 m in deeper waters. These are not absolute rules, but rather a good starting point that will likely require refinement for a given substructure. Additional considerations for SubDyn discretization include aspects that will impact structural accuracy, such as member weight, substructure modes and/or natural frequencies, load transfer, tapered members, and so on.

Members in SubDyn are assumed to be straight circular (and possibly tapered) cylinders. The use of more generic cross-sectional shapes will be considered in a future release.

Foundations

There are two methods that can be used to model foundation flexibility or soil-structure interaction in SubDyn. The first method makes use of the SSI stiffness and mass matrices at the partially restrained bottom joints as described in Sections 3.3.4, 3.4, and 6. The second method mimics the flexibility of the foundation through the apparent (or effective) fixity (AF) length approach, which idealizes a pile as a cantilever beam that has properties that are different above and below the mudline. The beam above the mudline should have the real properties (i.e., diameter, thickness, and material) of the pile. The beam below the mudline is specified with effective properties and a fictive length (i.e., the distance from the mudline to the cantilevered base) that are tuned to ensure that the overall response of the pile above the mudline is the same as the reality. The response can only be identical under a particular set of conditions; however, it is common for the properties of the fictive beam to be tuned so that the mudline displacement and rotation would be realistic when loaded by a mudline shear force and bending moment that are representative of the loading that exists when the offshore wind turbine is operating under normal conditions.

Note that in HydroDyn, all members that are embedded into the seabed (e.g., through piles or suction buckets) must have a joint that is located below the water depth. In SubDyn, the bottom joint(s) will be considered clamped or partially

restrained and therefore need not be located below the seabed when not applying the AF approach. For example, if the water depth is set to 20 m, and the user is modeling a fixed-bottom monopile with a rigid foundation, then the bottom-most joint in SubDyn can be set at $Z = -20$ m; HydroDyn, however, needs to have a Z -coordinate such that $Z < -20$ m. This configuration avoids HydroDyn applying static and dynamic pressure loads from the water on the bottom of the structure. When the AF approach is applied, the bottom-most joint in SubDyn should be set at $Z < -20$ m.

Member Overlap

As mentioned earlier, the current version of SubDyn is incapable of treating the overlap of members at the joints, resulting in an overestimate of the mass and potentially of the structure stiffness. One strategy to overcome this shortcoming employs virtual members to simulate the portion of each member within the overlap at a joint. The virtual members should be characterized by low self-mass and high stiffness. This can be achieved by introducing virtual joints at the approximate intersection of the finite-sized members, and then specifying additional members from these new joints to the original (centerline) joints. The new virtual members then use reduced material density and increased Young's and shear moduli. Care is advised in the choice of these parameters as they may render the system matrix singular. Inspection of the eigenvalue results in the summary file should confirm whether acceptable approximations have been achieved.

Substructure Tower/Turbine Coupling

When SubDyn is coupled to FAST, the 6 DOFs of the platform in ElastoDyn must be enabled to couple loads and displacements between the turbine and the substructure. The platform reference-point coordinates in ElastoDyn should also be set equal to the TP reference-point's coordinates (commonly indicating either the tower-base flange location, or TP centroid, or TP center of mass) that the user may have set in the stand-alone mode for checking the SubDyn model. A rigid connection between the SubDyn interface joints and TP reference point (\equiv platform reference point) is assumed.

For full lattice support structures or other structures with no transition piece, the entire support structure up to the yaw bearing may be modeled within SubDyn. Modeling the tower in SubDyn as opposed to ElastoDyn, for example, allows the ability to include more than the first two fore-aft and side-to-side bending modes, thus accounting for more general flexibility of the tower and its segments; however, for tubular towers, the structural model in ElastoDyn tends to be more accurate because ElastoDyn considers geometric nonlinearities not treated in SubDyn. When modeling full-lattice towers using SubDyn, the platform reference point in ElastoDyn can be located at the yaw bearing; in this case, the tower-bending DOFs in ElastoDyn should be disabled.

If FAST is run with SubDyn but not HydroDyn, the water depth will be automatically set to 0 m. This will influence the calculation of the reaction loads. Reactions are always provided at the assumed mudline, therefore, they would not be correctly located for an offshore turbine as a result. Thus, it is recommended that HydroDyn always be enabled when modeling bottom-fixed offshore wind turbines.

ElastoDyn also needs tower mode shapes specified (coefficients of best-fit sixth-order polynomials), derived using appropriate tower-base boundary conditions. They can be derived with an appropriate software (finite-element analysis, energy methods, or analytically) and by making use of the SubDyn-derived equivalent substructure stiffness and mass matrices (the **KBBt** and **MBBt** matrices found in the SubDyn summary file) to prescribe the boundary conditions at the base of the tower.

For instance, using NREL's BModes software, the SubDyn-obtained matrices can be used in place of the hydrodynamic stiffness (**hydro_K**) and mass matrices (**hydro_M**) (**mooring_K** can be set to zero). By setting the **hub_conn** boundary condition to two (free-free), BModes will calculate the mode shapes of the tower when tower cross-sectional properties are supplied. To obtain eigenmodes that are compatible with the FAST modal treatment of the tower (i.e., no axial or torsional modes and no distributed rotational-inertia contribution to the eigenmodes), the tower-distributed properties should be modified accordingly in BModes (e.g., by reducing mass moments of inertia towards zero and by increasing torsional and axial stiffness while assuring convergence of the results; see also <https://wind.nrel.gov/forum/wind/viewtopic.php?f=4&t=742>).

The rotational inertia of the undeflected tower about its centerline is not currently accounted for in ElastoDyn. Thus, when the nacelle-yaw DOF is enabled in ElastoDyn there will not be any rotational inertia of the platform-yaw DOF (which rotates the tower about its centerline) when both the platform-yaw inertia in ElastoDyn is zero and the tower is undeflected. To avoid a potential division-by-zero error in ElastoDyn when coupled to SubDyn, we recommend setting the platform-yaw inertia (**PtfmYIner**) in ElastoDyn equal to the total rotational inertia of the undeflected tower about its centerline. Note that the platform mass and inertia in ElastoDyn can be used to model heavy and rigid transition pieces that one would not want to model as a flexible body in either the ElastoDyn tower or SubDyn substructure models.

Damping of the Guyan modes:

There are three ways to specify the damping associated with the motion of the interface node.

1. SubDyn Guyan damping matrix using Rayleigh damping
2. SubDyn Guyan damping matrix using user defined 6x6 matrix
3. HydroDyn additional linear damping matrix (**AddBLin**)

The specification of the Guyan damping matrix in SubDyn is discussed in [Section 4.2.5](#).

Old:

The C-B method assumes no damping for the interface modes. This is equivalent to having six undamped rigid-body DOFs at the TP reference point in the absence of aerodynamic or hydrodynamic damping. Experience has shown that negligible platform-heave damping can cause numerical problems when SubDyn is coupled to FAST. One way to overcome this problem is to augment overall system damping with an additional linear damping for the platform-heave DOF. This augmentation can be achieved quite easily by calculating the damping from Eq. (4.92) and specifying this as the (3,3) element of HydroDyn's additional linear damping matrix, **AddBLin**. Experience has shown that a damping ratio of 1% of critical ($\zeta = 0.01$) is sufficient. In Eq. (4.92), $K_{33}^{(SD)}$ is the equivalent heave stiffness of the substructure (the (3,3) element of the **KBBt** (i.e., \tilde{K}_{BB}) matrix found in the SubDyn summary file, see also Section 6), $M_{33}^{(SD)}$ is the equivalent heave mass of the substructure (the (3,3) element of the **MBBt** (i.e., \tilde{M}_{BB}) matrix found in the SubDyn summary file, see also Section 6), and $M^{(ED)}$ is the total mass of the rotor, nacelle, tower, and TP (found in the ElastoDyn summary file).

$$C_{33}^{(HD)} = 2\zeta \sqrt{K_{33}^{(SD)} \left(M_{33}^{(SD)} + M^{(ED)} \right)} \quad (4.92)$$

To minimize extraneous excitation of the platform-heave DOF, it is useful to set the initial platform-heave displacement to its natural static-equilibrium position, which can be approximated by Eq. (4.93), where g is the magnitude of gravity. $PtfmHeave$ from Eq. (4.93) should be specified in the initial conditions section of the ElastoDyn input file.

$$PtfmHeave = - \frac{\left(M_{33}^{(SD)} + M^{(ED)} \right) g}{K_{33}^{(SD)}} \quad (4.93)$$

Self-Weight Calculations

SubDyn will calculate the self-weight of the members and apply appropriate forces and moments at the element nodes. Lumped masses will also be considered as concentrated gravity loads at prescribed joints. The array of self-weight forces can be seen in the summary file if the code is compiled with DEBUG compiler directives. In general, SubDyn assumes that structural motions of the substructure are small, such that (1) small-angle assumptions apply to structural rotations and (2) the so-called P- Δ effect is negligible, and therefore undeflected node locations are used for self-weight calculations.

Note On Other Load Calculations

When SubDyn is coupled to HydroDyn through FAST, the hydrodynamic loads, which include buoyancy, marine-growth weight, and wave and current loads, will be applied to the effective, deflected location of the nodes by the mesh-mapping routines in the glue code. Those loads, however, are based on wave kinematics at the undeflected position (see Jonkman et al. 2014 for more information).

Craig-Bampton Guidelines

When SubDyn is coupled with FAST, it is important to choose a sufficient number of C-B modes, ensuring that the vibrational modes of the coupled system are properly captured by the coupled model. We recommend that all modes up to at least 2-3 Hz be captured; wind, wave, and turbine excitations are important for frequencies up to 2-3 Hz. Eigenanalysis of the linearized, coupled system will make checking this condition possible and aid in the selection of the number of retained modes; however, the linearization process has yet to be implemented in FAST v8. Until full-system linearization is made available, experience has shown that it is sufficient to enable all C-B modes up to 10 Hz (the natural frequencies of the C-B modes are written to the SubDyn summary file). If SIM (see Section [4.2.5](#)) is not enabled, in addition to capturing physical modes up to a given frequency, the highest C-B mode must include the substructure axial modes so that gravity loading from self-weight is properly accounted for within SubDyn. This inclusion likely requires enabling a high number of C-B modes, reducing the benefit of the C-B reduction. Thus, we recommend employing the C-B reduction with SIM enabled. Because of the fixed-fixed treatment of the substructure boundary conditions in the C-B reduction, the C-B modes will always have higher natural frequencies than the physical modes.

Integration Time Step Guidelines

Another consideration when creating SubDyn input files is the time step size. SubDyn offers three explicit time-integrators — the fourth-order Runge-Kutta (RK4), fourth-order Adams-Bashforth (AB4), fourth-order Adams-Bashforth-Moulton (ABM4) methods — and the implicit second-order Adams-Moulton (AM2) method. Users have the option of using the global time step from the glue code or an alternative SubDyn-unique time step that is an integer multiple smaller than the glue-code time step. It is essential that a small enough time step is used to ensure solution accuracy (by providing a sufficient sampling rate to characterize all key frequencies of the system), numerical stability of the selected explicit time-integrator, and that the coupling with FAST is numerically stable.

For the RK4 and ABM4 methods, we recommend that the SubDyn time step follow the relationship shown in Eq. (4.94), where f_{max} is the higher of (1) the highest natural frequency of the retained C-B modes and (2) the highest natural frequency of the physical modes when coupled to FAST. Although the former can be obtained from the SubDyn summary file, the latter is hard to estimate before the full-system linearization of the coupled FAST model is realized. Until then, experience has shown that the highest physical mode when SubDyn is coupled to FAST is often the platform-heave mode of ElastoDyn, with a frequency given by Eq. (4.95), where the variables are defined in Section 5.3.

$$dt_{max} = \frac{1}{10f_{max}} \quad (4.94)$$

$$f = \frac{1}{2\pi} \sqrt{\frac{K_{33}^{(SD)}}{M_{33}^{(SD)} + M^{(ED)}}} \quad (4.95)$$

For the AB4 method, the recommended time step is half the value given by Eq. (4.94).

For AM2, being implicit, the required time step is not driven by natural frequencies within SubDyn, but should still be chosen to ensure solution accuracy and that the coupling to FAST is numerically stable.

SubDyn Theory

Overview

This section focuses on the theory behind the SubDyn module.

SubDyn relies on two main engineering approaches: (1) a linear frame finite-element model (LFEM), and (2) a dynamics system reduction via the Craig-Bampton (C-B) method together with a static-improvement method (SIM), greatly reducing the number of modes needed to obtain an accurate solution.

There are many nonlinearities present in offshore wind substructure models, including material nonlinearity, axial shortening caused by bending, large displacements, and so on. The material nonlinearity is not considered here because most offshore multimember support structures are designed to use steel and the maximum stress is intended to be below the yield strength of the material. [DSRJ13] demonstrate that a linear finite-element method is suitable when analyzing wind turbine substructures. In this work, several wind turbine configurations that varied in base geometry, load paths, sizes, supported towers, and turbine masses were analyzed under extreme loads using nonlinear and linear models. The results revealed that the nonlinear behavior was mainly caused by the mono-tower response and had little effect on the multimember support structures. Therefore, an LFEM model for the substructure is considered appropriate for wind turbine substructures. The LFEM can accommodate different element types, including Euler-Bernoulli and Timoshenko beam elements of either constant or longitudinally tapered cross sections (Timoshenko beam elements account for shear deformation and are better suited to represent low aspect ratio beams that may be used within frames and to transfer the loads within the frame).

The large number of DOFs ($\sim 10^3$) associated with a standard finite-element analysis of a typical multimember structure would hamper computational efficiency during wind turbine system dynamic simulations. As a result, the C-B system reduction was implemented to speed up processing time while retaining a high level of fidelity in the overall system response. The C-B reduction is used to recharacterize the substructure finite-element model into a reduced DOF model that maintains the fundamental low-frequency response modes of the structure. In the SubDyn initialization step, the large substructure physical DOFs (displacements) are reduced to a small number of modal DOFs and interface (boundary) DOFs, and during each time step, only the equations of motion of these DOFs need to be solved. SubDyn only solves the equations of motion for the modal DOFs, the motion of the interface (boundary) DOFs are either prescribed when running SubDyn in stand-alone mode or solved through equations of motion in ElastoDyn when SubDyn is coupled to FAST.

Retaining just a few DOFs may, however, lead to the exclusion of axial modes (normally of very high frequencies), which are important to capture static load effects, such as those caused by gravity and buoyancy. The so-called SIM was implemented to mitigate this problem. SIM computes two static solutions at each time step: one based on the full system stiffness matrix and one based on the C-B reduced stiffness matrix. At each time step the time-varying, C-B based, dynamic solution is superimposed on the difference between the two static solutions, which amounts to quasi-statically accounting for the contribution of those modes not directly included within the dynamic solution.

In SubDyn, the substructure is considered to be clamped, or connected via linear spring-like elements, at the bottom nodes (normally at the seabed) and rigidly connected to the TP at the substructure top nodes (interface nodes). The user can provide 6x6, equivalent stiffness and mass matrices for each of the bottom nodes to account for soil-pile interaction. As described in other sections of this document, the input file defines the substructure geometry, material properties, and constraints. Users can define: element types; full finite-element mode or C-B reduction; the number of modes to be retained in the C-B reduction; modal damping coefficients; whether to take advantage of SIM; and the number of elements for each member.

The following sections discuss the integration of SubDyn within the FAST framework, the main coordinate systems used in the module, and the theory pertaining to the LFEM, the C-B reduction, and SIM. The state-space formulations to be used in the time-domain simulation are also presented. The last section discusses the calculation of the base reaction calculation. For further details, see also [SDRJ13].

Integration with the FAST Modularization Framework

Based on a new modularization framework [JJø13], FAST joins an aerodynamics module, a hydrodynamics module, a control and electrical system (servo) module, and structural-dynamics (elastic) modules to enable coupled nonlinear aero-hydro-servo-elastic analysis of land-based and offshore wind turbines in the time domain. Fig. 4.34 shows the basic layout of the SubDyn module within the FAST modularization framework.

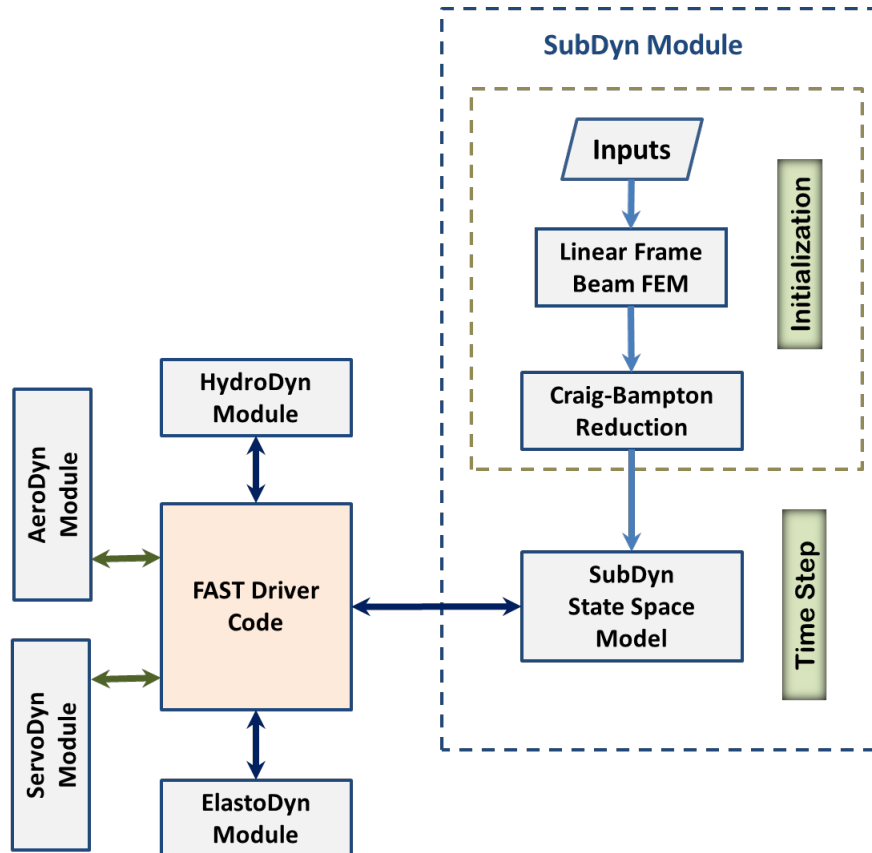


Fig. 4.34: SubDyn layout within the modularization framework

In the existing loosely coupled time-integration scheme, the glue-code transfers data at each time step. Such data includes hydrodynamic loads, substructure response, loads transmitted to the TP, and TP response among SubDyn, HydroDyn, and ElastoDyn. At the interface nodes, the TP displacement, rotation, velocity, and acceleration are inputs to SubDyn from ElastoDyn, and the reaction forces at the TP are outputs of SubDyn for input to ElastoDyn. SubDyn also outputs the substructure displacements, velocities, and accelerations for input to HydroDyn to calculate the hydrodynamic loads that become inputs for SubDyn. In addition, SubDyn can calculate the member forces, as requested by the user. Within this scheme, SubDyn tracks its states and integrates its equations through its own solver.

In a tightly coupled time-integration scheme (yet to be implemented), SubDyn sets up its own equations, but its states and those of other modules are tracked and integrated by a solver within the glue-code that is common to all of the modules.

SubDyn is implemented in a state-space formulation that forms the equation of motion of the substructure system with physical DOFs at the boundaries and modal DOFs representing all interior motions. At each time step, loads and motions are exchanged between modules through the driver code; the modal responses are calculated inside SubDyn's state-space model; and the next time-step responses are calculated by the SubDyn integrator for loose coupling and the global system integrator for tight coupling.

Coordinate Systems

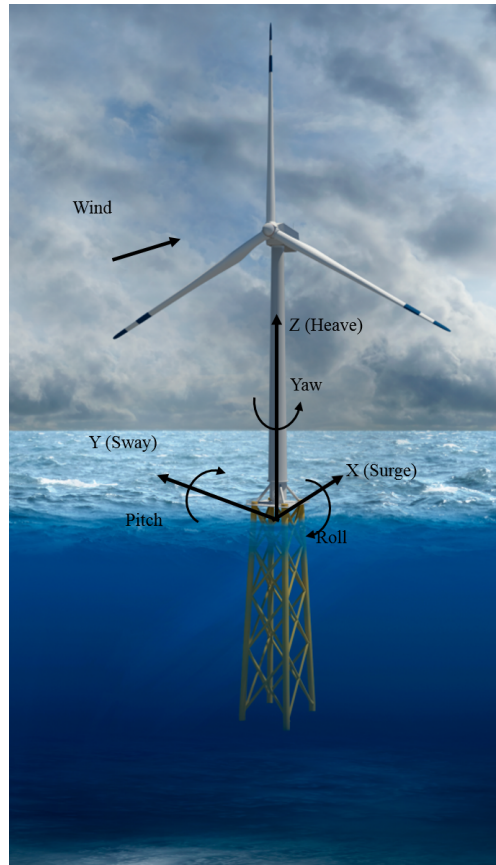


Fig. 4.35: Global (coincident with the substructure) coordinate system. Also shown are the DOFs associated with the TP reference point.

Global and Substructure Coordinate System: (X, Y, Z) or (X_{SS}, Y_{SS}, Z_{SS}) (Fig. 4.35)

- The global axes are represented by the unit vectors \hat{I} , \hat{J} , and \hat{K} .
- The origin is set at the intersection between the undeflected tower centerline and the horizontal plane identified by the mean sea level (MSL) for offshore systems or ground level for land-based systems.
- The positive Z (Z_{SS}) axis is vertical and pointing upward, opposite gravity.
- The positive X (X_{SS}) axis is along the nominal (zero-degree) wind and wave propagation direction.
- The Y (Y_{SS}) axis is transverse and can be found assuming a right-handed Cartesian coordinate system (directed to the left when looking in the nominal downwind direction).

Member or Element Local Coordinate System (x_e, y_e, z_e) (Fig. 4.36)

- Axes are represented by the unit vectors $\hat{i}_e, \hat{j}_e, \hat{k}_e$.
- The origin is set at the shear center of the cross section at the start node (S, MJointID1).
- The local z_e axis is along the elastic axis of the member, directed from the start node (S) to the end node (E, MJointID2). Nodes are ordered along the member main axis directed from start joint to end joint (per user's input definition).
- The local x_e axis is parallel to the global XY plane, and directed such that a positive, less than or equal to 180° rotation about it, would bring the local z_e axis parallel to the global Z axis.
- The local y_e axis can be found assuming a right-handed Cartesian coordinate system.

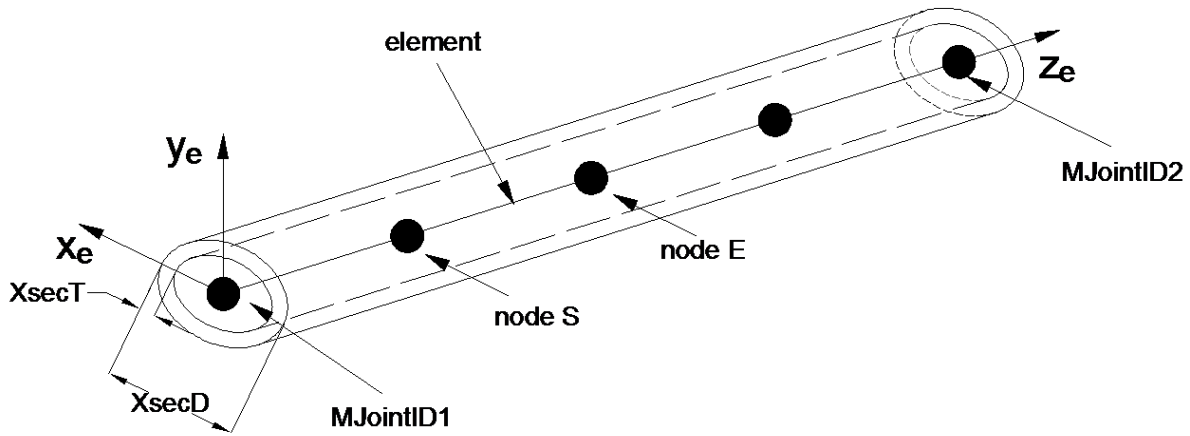


Fig. 4.36: The element coordinate system. The sketched member contains four elements, and the second element is called out with nodes S and E.

Local to Global Transformation

The transformation from local to global coordinate system can be expressed by the following equation:

$$\begin{bmatrix} \Delta X \\ \Delta Y \\ \Delta Z \end{bmatrix} = [\mathbf{D}_c] \begin{bmatrix} \Delta x_e \\ \Delta y_e \\ \Delta z_e \end{bmatrix} \quad (4.96)$$

where $\begin{bmatrix} \Delta x_e \\ \Delta y_e \\ \Delta z_e \end{bmatrix}$ is a generic vector in the local coordinate system, and $\begin{bmatrix} \Delta X \\ \Delta Y \\ \Delta Z \end{bmatrix}$ the same vector but in the global coordinate system; and $[\mathbf{D}_c]$ is the direction cosine matrix of the member axes and can be obtained as follows:

$$[\mathbf{D}_c] = \begin{bmatrix} \frac{Y_E - Y_S}{L_{exy}} & \frac{(X_E - X_S)(Z_E - Z_S)}{L_{exy} L_e} & \frac{X_E - X_S}{L_e} \\ -\frac{X_E - X_S}{L_{exy}} & \frac{(Y_E - Y_S)(Z_E - Z_S)}{L_{exy} L_e} & \frac{Y_E - Y_S}{L_e} \\ 0 & -\frac{L_{exy}}{L_e} & \frac{Z_E - Z_S}{L_e} \end{bmatrix} \quad (4.97)$$

Where (X_S, Y_S, Z_S) and (X_E, Y_E, Z_E) are the start and end joints of the member (or nodes of the element of interest) in global coordinate system ; $L_{exy} = \sqrt{(X_E - X_S)^2 + (Y_E - Y_S)^2}$ and $L_e = \sqrt{(X_E - X_S)^2 + (Y_E - Y_S)^2 + (Z_E - Z_S)^2}$.

If $X_E = X_S$ and $Z_E = Z_S$, the $[\mathbf{D}_c]$ matrix can be found as follows:

if $Z_E > Z_S$ then

$$[\mathbf{D}_c] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (4.98)$$

else

$$[\mathbf{D}_c] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \quad (4.99)$$

In the current SubDyn release, the transpose (global to local) of these direction cosine matrices for each member is returned in the summary file. Given the circular shape of the member cross sections, the direction cosine matrices have little importance on the member load verification. To verify joints following the standards (e.g., [ISO07] [API14]), however, the bending moments need to be decomposed into in-plane and out-of-plane components, where the plane is that defined by either a pair of braces (for an X-joint), or by the pair brace(s) plus leg (for a K-joint). It is therefore important to have the direction cosines of the interested members readily available to properly manipulate and transform the local shear forces and bending moments.

When member cross sections other than circular are allowed in future releases, the user will need to input cosine matrices to indicate the final orientation of the member principal axes with respect to the global reference frame.

Finite-Element Model - Elements and Constraints

Definitions

Figure Fig. 4.37 is used to illustrate some of the definitions used. The model of the substructure is assumed to consists of different members. A member is delimited by two joints. A joint is defined by the coordinates of a point of the undeflected structure and a type (*JointType*). The type of a joint defines the boundary condition or constraint of all the members that are attached to this joint. The following joints are supported:

- Cantilever joints (*JointType*=1)
- Universal joint (*JointType*=2)
- Pin joint (*JointType*=3)
- Ball joint (*JointType*=4)

A member is one of the three following types:

- Beams ($MType=1$), Euler-Bernoulli ($FEMMod=1$) or Timoshenko ($FEMMod=3$)
- Pretension cables ($MType=2$)
- Rigid link ($MType=3$)

Beam members may be split into several elements to increase the accuracy of the model (using the input parameter $NDiv$). Member of other types (rigid links and pretension cables) are not split. In this document, the term *element* refers to: a sub-division of a beam member or a member of another type than beam (rigid-link or pretension cable). The term *joints* refers to the points defining the extremities of the members. Some joints are defined in the input file, while others arise from the subdivision of beam members. The end points of an elements are called nodes and each node consists of 6 degrees of freedom (DOF) for the element implemented. In the current implementation, no geometrical offsets are assumed between a joint and the node of an element, or between the nodes of connected elements.

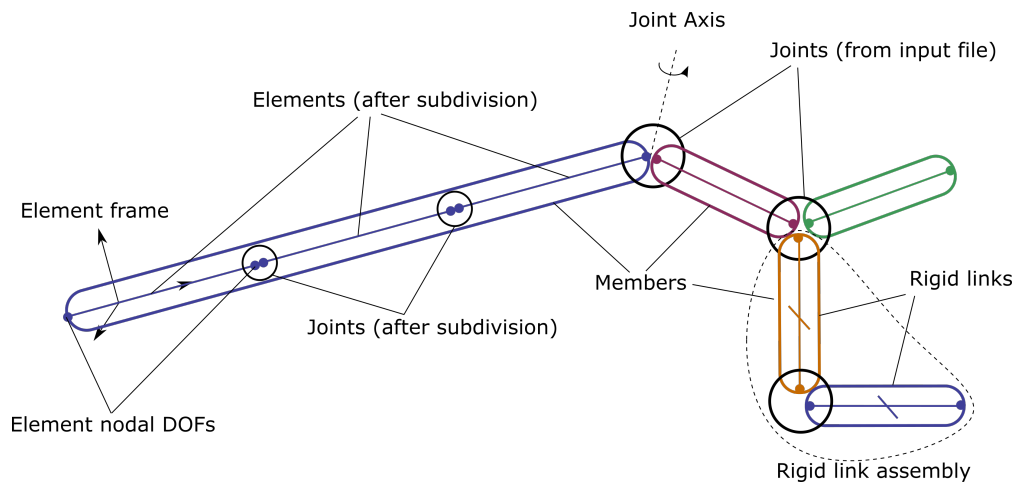


Fig. 4.37: Definitions of members, element, joints, nodes and rigid assemblies.

FEM process - from elements to system matrices

The process to obtain a FE representation of the system (performed at initialization) is as follows:

- **Elements:** The mass and stiffness matrices of each element are computed and transformed to global coordinates using directional cosine matrices
- **Assembly:** The element matrices are inserted into the full system matrices. The DOFs of cantilever joints are mapped to each other. The translational DOFs of the nodes linked by a joint different from a cantilever joint are mapped to each other, but the rotational DOFs of each individual nodes are retained in this system. The vector of degrees of freedom of this full system is noted \mathbf{x}
- **Constraints elimination:** A direct-elimination technique is used to apply the constraints introduced by the joints and the rigid links. The elimination consists in forming a matrix \mathbf{T} and a reduced set of degrees of freedom $\tilde{\mathbf{x}}$ such that $\mathbf{x} = \mathbf{T}\tilde{\mathbf{x}}$.
- **CB-reduction:** The Craig-Bampton reduction technique is used to obtain a reduced set of degrees of freedom (interface DOFs and Craig-Bampton modes)
- **Boundary conditions:** The displacements boundary conditions are then applied (e.g. for a fixed bottom foundation)

The remaining of the section focuses on the element matrices, and the account of the constraints introduced by the joints and rigid links. The Craig-Bampton reduction is described in [Section 4.2.5](#).

$$\begin{bmatrix}
\frac{13A_z L_e}{35} + \frac{6J_y}{5L_e} & 0 & 0 & 0 & \frac{11A_z L_e^2}{210} + \frac{J_y}{5L_e} & 0 & \frac{9A_z L_e}{70} - \frac{6J_y}{5L_e} & 0 & 0 & 0 \\
0 & \frac{12EJ_x}{L_e^3(1+K_{sx})} & 0 & -\frac{6EJ_x}{L_e^2(1+K_{sx})} & 0 & 0 & 0 & -\frac{12EJ_x}{L_e^3(1+K_{sx})} & 0 & -\frac{6EJ_x}{L_e^2(1+K_{sx})} \\
0 & \frac{EA_z}{L_e} & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{EA_z}{L_e} & 0 \\
0 & 0 & \frac{(4+K_{sx})EJ_x}{L_e(1+K_{sx})} & 0 & 0 & 0 & 0 & \frac{6EJ_x}{L_e^2(1+K_{sx})} & 0 & \frac{(2-K_{sx})EJ_x}{L_e(1+K_{sx})} \\
0 & 0 & 0 & \frac{(4+K_{sy})EJ_y}{L_e(1+K_{sy})} & 0 & -\frac{6EJ_y}{L_e^2(1+K_{sy})} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{GJ_z}{L_e} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \frac{12EJ_y}{L_e^3(1+K_{sy})} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{12EJ_x}{L_e^3(1+K_{sx})} & 0 & 0 & \frac{6EJ_x}{L_e^2(1+K_{sx})} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{EA_z}{L_e} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{(4+K_{sx})EJ_x}{L_e(1+K_{sx})} & 0
\end{bmatrix}
\quad (4.102)$$

where A_z is the element cross-section area, J_x, J_y, J_z are the area second moments of inertia with respect to principal axes of the cross section; L_e is the length of the undisplaced element from start-node to end-node; ρ, E , and G are material density, Young's, and Shear moduli, respectively; K_{sx}, K_{sy} are shear correction factors as shown below (they are set to zero if the E-B formulation is chosen):

$$\begin{aligned}
K_{sx} &= \frac{12EJ_y}{GA_{sx}L_e^2} \\
K_{sy} &= \frac{12EJ_x}{GA_{sy}L_e^2}
\end{aligned}
\quad (4.103)$$

where the shear areas along the local x and y (principal) axes are defined as:

$$\begin{aligned}
A_{sx} &= k_{ax}A_z \\
A_{sy} &= k_{ay}A_z
\end{aligned}
\quad (4.104)$$

and

$$k_{ax} = k_{ay} = \frac{6(1+\mu)^2 \left(1 + \left(\frac{D_i}{D_o}\right)^2\right)^2}{\left(1 + \left(\frac{D_i}{D_o}\right)^2\right)^2 (7 + 14\mu + 8\mu^2) + 4 \left(\frac{D_i}{D_o}\right)^2 (5 + 10\mu + 4\mu^2)}
\quad (4.105)$$

Eq. (4.105) is from [SKM13] for hollow circular cross sections, with μ denoting Poisson's ratio.

Before assembling the global system stiffness (K) and mass (M) matrices, the individual $[k_e]$ and $[m_e]$ are modified to the global coordinate system via $[\mathbf{D}_c]$ as shown in the following equations:

$$[k] = \begin{bmatrix} [\mathbf{D}_c] & 0 & 0 & 0 \\ & [\mathbf{D}_c] & 0 & 0 \\ & & [\mathbf{D}_c] & 0 \\ & & & [\mathbf{D}_c] \end{bmatrix} [k_e] \begin{bmatrix} [\mathbf{D}_c] & 0 & 0 & 0 \\ & [\mathbf{D}_c] & 0 & 0 \\ & & [\mathbf{D}_c] & 0 \\ & & & [\mathbf{D}_c] \end{bmatrix}^T
\quad (4.106)$$

$$[m] = \begin{bmatrix} [\mathbf{D}_c] & 0 & 0 & 0 \\ & [\mathbf{D}_c] & 0 & 0 \\ & & [\mathbf{D}_c] & 0 \\ & & & [\mathbf{D}_c] \end{bmatrix} [m_e] \begin{bmatrix} [\mathbf{D}_c] & 0 & 0 & 0 \\ & [\mathbf{D}_c] & 0 & 0 \\ & & [\mathbf{D}_c] & 0 \\ & & & [\mathbf{D}_c] \end{bmatrix}^T
\quad (4.107)$$

where m and k are element matrices in the global coordinate system.

Pretension Cable Element Formulation

The master stiffness equations of FEM assumes that the forces vanish if all displacements also vanish, that is, the relation between force and displacement is linear, $\mathbf{f} = \mathbf{K}\mathbf{u}$. This assumption does not hold if the material is subject to so-called initial strain, initial stress or prestress. Such effects may be produced by temperature changes and pretensions (or lack-of-fit fabrications). These effects are for instance discussed in the notes of Felippa [Fel04].

Pretension cables may be modelled by assuming an initial elongation of a truss element and considering the restoring force this initial elongation may have in both the longitudinal and orthogonal direction.

Derivation

A pretension cable oriented along the z -direction is considered. To simplify the derivation, the left point is assumed fixed and only the right point deflects. The notations are illustrated in Fig. 4.38.

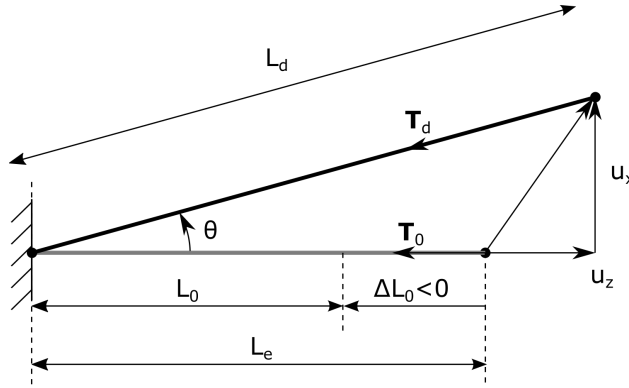


Fig. 4.38: Notations used for the derivation of the pretension cable equation

The length of the element prior to the pretension is written L_0 , and its axial stiffness is $k = EA/L_0$. In this equilibrium position the stress in the cable is zero. The user inputs for this elements are selected as: the un-displaced joint locations (while pre-tensioned) \mathbf{x}_1 and \mathbf{x}_2 , the elongation stiffness EA , and the change in length $\Delta L_0 = L_0 - L_e (< 0)$. The pretension force T_0 is a derived input. The following quantities are defined:

$$L_e = \|\mathbf{x}_2 - \mathbf{x}_1\|, \quad \epsilon_0 = \frac{T_0}{EA}, \quad L_0 = \frac{L_e}{1 + \epsilon_0}$$

The different variables are defined as function of the inputs as follows:

$$L_0 = L_e + \Delta L_0 \quad T_0 = -EA \frac{\Delta L_0}{L_0}, \quad \epsilon_0 = \frac{T_0}{EA} = \frac{-\Delta L_0}{L_0} = \frac{-\Delta L_0}{L_e + \Delta L_0}$$

The degrees of freedom for the deflections of the cable, (u_x, u_z) , are measured from a position which is not the equilibrium position, but a position that is offset from the equilibrium position, such that the pretensioned length of the element is $L_e > L_0$. The stress in the cable for $u_z = 0$ is noted $\epsilon_0 = (L_e - L_0)/L_0$, or $L_e = L_0(1 + \epsilon_0)$. The initial tension in the cable is $\mathbf{T}_0 = -k(L_e - L_0)\mathbf{e}_z = -EA\epsilon_0\mathbf{e}_z$. In its deflected position, the length of the cable is:

$$L_d = \sqrt{(L_e + u_z)^2 + u_x^2} = L_e \sqrt{1 + \frac{2u_z}{L_e} + \frac{u_z^2}{L_e^2} + \frac{u_x^2}{L_e^2}} \approx L_e \left(1 + \frac{u_z}{L_e}\right)$$

where the deflections are assumed small compared to the element length L_e , $u_x \ll L_e$ and $u_z \ll L_e$, and only the first order terms are kept. The tension force in the deflected cable is then $\mathbf{T}_d = -k(L_d - L_0)\mathbf{e}_r$ where the radial vector is the vector along the deflected cable such that:

$$\mathbf{e}_r = \cos \theta \mathbf{e}_z + \sin \theta \mathbf{e}_x, \quad \text{with} \quad \cos \theta = \frac{L_e + u_z}{L_d} \approx 1, \quad \sin \theta = \frac{u_x}{L_d} \approx \frac{u_x}{L_e} \left(1 - \frac{u_z}{L_e}\right) \approx \frac{u_x}{L_e}$$

The components of the tension force are then:

$$\begin{aligned} T_{d,z} &= -k(L_d - L_0) \cos \theta \approx -\frac{EA}{L_0}(L_e - L_0 + u_z)1 = -EA\epsilon_0 - \frac{EA}{L_0}u_z \\ T_{d,x} &= -k(L_d - L_0) \sin \theta \approx -\frac{EA}{L_0}(L_e - L_0 + u_z)\frac{u_x}{L_e} \approx -EA\epsilon_0\frac{u_x}{L_e} = -\frac{EA\epsilon_0}{L_0(1+\epsilon_0)}u_x \end{aligned}$$

Finite element formulation of a pretension cable

The rotational degrees of freedom are omitted for conciseness since these degrees of freedom are not considered in this cable element. The linear formulation from is applied to both nodes of a finite element, interpreting the force at each node as the internal force that the element exert on the nodes. Using this convention, the pretension cable element can be represented with an element stiffness matrix \mathbf{K}_e and an additional nodal load vector $\mathbf{f}_{e,0}$ such that the static equilibrium equation of the element writes $\mathbf{f}_e = \mathbf{K}_e \mathbf{u} + \mathbf{f}_{e,0}$, with:

$$\begin{bmatrix} f_{x,1} \\ f_{y,1} \\ f_{z,1} \\ f_{x,2} \\ f_{y,2} \\ f_{z,2} \end{bmatrix} = \frac{EA}{L_0} \begin{bmatrix} \frac{\epsilon_0}{1+\epsilon_0} & 0 & 0 & -\frac{\epsilon_0}{1+\epsilon_0} & 0 & 0 \\ 0 & \frac{\epsilon_0}{1+\epsilon_0} & 0 & 0 & -\frac{\epsilon_0}{1+\epsilon_0} & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \\ -\frac{\epsilon_0}{1+\epsilon_0} & 0 & 0 & \frac{\epsilon_0}{1+\epsilon_0} & 0 & 0 \\ 0 & -\frac{\epsilon_0}{1+\epsilon_0} & 0 & 0 & \frac{\epsilon_0}{1+\epsilon_0} & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_{x,1} \\ u_{y,1} \\ u_{z,1} \\ u_{x,2} \\ u_{y,2} \\ u_{z,2} \end{bmatrix} + EA\epsilon_0 \begin{bmatrix} 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad (4.108)$$

The relation above is expressed in the element coordinate system. The stiffness matrix and force vector are transferred to the global system during the assembly process. Inserting $\epsilon_0 = 0$ in the above equations leads to the formulation of a truss element. The linear model above is only valid for $L_d - L_0 > 0$, that is $(L_e - L_0 + u_{z,2} - u_{z,1}) > 0$, and the implementation should abort if this condition is not reached at a given time. If the cable has a positive mass density ρ , the mass matrix of the element is given by:

$$\mathbf{M}_e = \rho L_e \begin{bmatrix} 13/35 & 0 & 0 & & 9/70 & 0 & 0 & \\ 0 & 13/35 & 0 & \mathbf{0}_3 & 0 & 9/70 & 0 & \mathbf{0}_3 \\ 0 & 0 & 1/3 & & 0 & 0 & 1/6 & \\ & & & \mathbf{0}_3 & & \mathbf{0}_3 & & \mathbf{0}_3 \\ 9/70 & 0 & 0 & & 13/35 & 0 & 0 & \\ 0 & 9/70 & 0 & \mathbf{0}_3 & 0 & 13/35 & 0 & \mathbf{0}_3 \\ 0 & 0 & 1/6 & & 0 & 0 & 1/3 & \\ & & & \mathbf{0}_3 & & \mathbf{0}_3 & & \mathbf{0}_3 \end{bmatrix}$$

with L_e the *undisplaced* length of the element (not L_0).

Controlled pretension cable

The controller changes the rest length of the cable at each time step, effectively changing the pretension properties of the cable. At a given time, the restlength of the cable is $L_r(t) = L_e + \Delta L$ (instead of L_0), and the pretension force is $T(t)$ (instead of T_0). The pretension force is given as:

$$T(t) = EA \frac{-\Delta L(t)}{L_r(t)} = EA \frac{-\Delta L(t)}{L_e + \Delta L(t)} \quad (4.109)$$

At $t = 0$, when no controller action is present, the pretension force and length are:

$$T(0) = T_0 = EA \frac{-\Delta L_0}{L_e + \Delta L_0}, \quad \Delta L(0) = \Delta L_0 = \frac{-L_e T_0}{EA + T_0} \quad (4.110)$$

The quantity ΔL is the change in restlength, and it is given as:

$$\Delta L(t) = \Delta L_0 + \Delta L_c(t) \quad (4.111)$$

where ΔL_c is the change of length prescribed by the controller, and ΔL_0 is the change of length attributed to the initial pretension. This choice is such that the controller input is nominally 0. Cable extension beyond the element length (L_e) is not allowed in SubDyn, therefore ΔL is limited to negative values ($L_r = L_e + \Delta L \leq L_e$). The tension force at a given time is given by inserting (4.111) into (4.109):

$$T(t) = -EA \frac{\Delta L_0 + \Delta L_c}{L_e + \Delta L_0 + \Delta L_c}$$

In the following we provide details on the implementation and the approximation introduced. The “equations of motions” for a cable element are written:

$$M_e \ddot{u}_e = f_e$$

If the pretension force is constant (equal to T_0), and additional external loads are neglected, then the element force is:

$$f_e = f_e(t, T_0) = -K_c(T_0)u_e + f_c(T_0) + f_g \quad (4.112)$$

where $f_c(T_0)$ and $K_c(T_0)$ are given in (4.108). If the pretension force is varying with time ($T = T(t)$), then the force is:

$$f_e(t) = -K_c(T)u_e + f_c(T) + f_g \quad (4.113)$$

where (4.113) is evaluated with $\epsilon = \frac{T}{EA}$ and $L = \frac{L_e}{1+\epsilon}$. We seek to express (4.113), as a correction term added to the equation of a constant pretension cable (i.e. (4.112), with $T(0) = T_0$). We add $\pm f_e(t, T_0)$ to , leading to:

$$\begin{aligned} f_e(t) &= [-K_c(T_0)u_e + f_c(T_0) + f_g] - [-K_c(T_0)u_e + f_c(T_0) + f_g] + [-K_c(T)u_e + f_c(T) + f_g] \\ &= \left[\underbrace{-K_c(T_0)u_e}_{\text{in } CB} + \underbrace{f_c(T_0) + f_g}_{\text{in } F_G} \right] + f_{c,\text{control}}(T) \end{aligned}$$

where $f_{c,\text{control}}$ is the correction term accounting for the time variation of T :

$$f_{c,\text{control}}(T) = (K_c(T_0) - K_c(T))u_e + f_c(T) - f_c(T_0)$$

This equation is transformed to the global system using the direction cosine matrices of the element. The part involving u introduces non-linearities, and is currently neglected. Using , the additional control force for a given element is:

$$f_{c,\text{control}}(T) \approx f_c(T) - f_c(T_0) = (T - T_0) \begin{bmatrix} 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Constraints introduced by Rotational Joints

As mentioned in Section 4.2.5, the account of constraints is done via a direct elimination technique. The technique is implemented by computing a transformation matrix T which gives the relationship between the reduced set of DOF (accounting for constraints) and the full set of DOFs. When no constraints are present this matrix is the identity matrix. This section describes how the T matrix is obtained for rotational joints.

Formulation Joints between two nodes k and l are here considered. Before accounting for the constraint introduced by the joints, 12 degrees of freedom are present: $(\mathbf{u}_k, \boldsymbol{\theta}_k, \mathbf{u}_l, \boldsymbol{\theta}_l)$. After application of the constraints, the new set of degrees of freedom is noted $(\tilde{\mathbf{u}}_{kl}, \tilde{\boldsymbol{\theta}}_{kl})$. The degrees of freedom retained for each joint type is shown in the table below. The meaning of the different θ -variable will be made explicit in the subsequent paragraphs.

Table 4.7: Nodal degrees of freedom (DOF) for different joint types.

Joint type	n_c	n_{DOF}	$\tilde{\mathbf{u}}_{kl}$	$\tilde{\boldsymbol{\theta}}_{kl}$
Cantilever	6	$12 \rightarrow 6$	u_x, u_y, u_z	$\theta_x, \theta_y, \theta_z$
Pin	5	$12 \rightarrow 7$	u_x, u_y, u_z	$\theta_1, \theta_2, \theta_3, \theta_4$
Universal	4	$12 \rightarrow 8$	u_x, u_y, u_z	$\theta_1, \theta_2, \theta_3, \theta_4, \theta_5$
Ball	3	$12 \rightarrow 9$	u_x, u_y, u_z	$\theta_{x,k}, \theta_{y,k}, \theta_{z,k}, \theta_{x,l}, \theta_{y,l}, \theta_{z,l}$

For all the joints considered, the translational DOF of the two nodes are made equal, which may be formally expressed as:

$$\begin{bmatrix} \mathbf{u}_k \\ \mathbf{u}_l \end{bmatrix} = \begin{bmatrix} \mathbf{I}_3 \\ \mathbf{I}_3 \end{bmatrix} \tilde{\mathbf{u}}_{kl}$$

Since this relation is the same for all the joints, the relation between the degrees of freedom is taken care in the assembly step. The constraints of each joints will hence be expressed in the following form:

$$\begin{bmatrix} \boldsymbol{\theta}_k \\ \boldsymbol{\theta}_l \end{bmatrix} = \mathbf{T}_{kl} \tilde{\boldsymbol{\theta}}_{kl}$$

Cantilever joint For a cantilever joint between two elements, the reduction is:

$$\begin{bmatrix} \boldsymbol{\theta}_k \\ \boldsymbol{\theta}_l \end{bmatrix} = \mathbf{T}_{kl} \tilde{\boldsymbol{\theta}}_{kl}, \quad \text{with} \quad \tilde{\boldsymbol{\theta}}_{kl} = [\boldsymbol{\theta}_k], \quad \mathbf{T}_{kl} = \begin{bmatrix} \mathbf{I}_3 \\ \mathbf{I}_3 \end{bmatrix}$$

This relationship is taken care of during the assembly process directly, and readily extended to n elements.

Ball/spherical joint For a spherical joint between two elements, the reduction is as follows:

$$\begin{bmatrix} \boldsymbol{\theta}_k \\ \boldsymbol{\theta}_l \end{bmatrix} = \mathbf{T}_{kl} \tilde{\boldsymbol{\theta}}_{kl}, \quad \text{with} \quad \tilde{\boldsymbol{\theta}}_{kl} = \begin{bmatrix} \boldsymbol{\theta}_k \\ \boldsymbol{\theta}_l \end{bmatrix}, \quad \mathbf{T}_{kl} = \begin{bmatrix} \mathbf{I}_3 & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_3 \end{bmatrix}$$

For n elements $[e_1, \dots, e_n]$ connected by a ball joint (constraint c), the relationship is extended as follows:

$$\begin{bmatrix} \boldsymbol{\theta}_{e_1} \\ \vdots \\ \boldsymbol{\theta}_{e_n} \end{bmatrix} = \mathbf{T}^c \tilde{\boldsymbol{\theta}}^c, \quad \text{with} \quad \tilde{\boldsymbol{\theta}}^c = \begin{bmatrix} \boldsymbol{\theta}_{e_1} \\ \vdots \\ \boldsymbol{\theta}_{e_n} \end{bmatrix}, \quad \mathbf{T}^c = \begin{bmatrix} \mathbf{I}_3 & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \mathbf{I}_3 \end{bmatrix}$$

Pin/revolute joint A pin joint is characterized by a direction around which no moment is transferred. The unit vector indicating this direction is noted $\hat{\mathbf{p}}$. Two orthogonal vectors \mathbf{p}_1 and \mathbf{p}_2 are then defined, forming an orthonormal base with $\hat{\mathbf{p}}$, oriented arbitrarily (see Fig. 4.39).

The variables $\tilde{\theta}_1 \dots \tilde{\theta}_4$ are then defined as:

$$\begin{aligned} \tilde{\theta}_1 &= \mathbf{p}_1^t \cdot \boldsymbol{\theta}_k = \mathbf{p}_1^t \cdot \boldsymbol{\theta}_l \\ \tilde{\theta}_2 &= \mathbf{p}_2^t \cdot \boldsymbol{\theta}_k = \mathbf{p}_2^t \cdot \boldsymbol{\theta}_l \\ \tilde{\theta}_3 &= \hat{\mathbf{p}}^t \cdot \boldsymbol{\theta}_k \\ \tilde{\theta}_4 &= \hat{\mathbf{p}}^t \cdot \boldsymbol{\theta}_l \end{aligned}$$

which may be written in matrix form as:

$$\begin{bmatrix} \tilde{\theta}_1 \\ \tilde{\theta}_2 \\ \tilde{\theta}_3 \\ \tilde{\theta}_4 \end{bmatrix} = \mathbf{A} \begin{bmatrix} \boldsymbol{\theta}_k \\ \boldsymbol{\theta}_l \end{bmatrix} = \begin{bmatrix} \mathbf{p}_1^t/2 & \mathbf{p}_1^t/2 \\ \mathbf{p}_2^t/2 & \mathbf{p}_2^t/2 \\ \hat{\mathbf{p}}^t & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{p}}^t \end{bmatrix} \begin{bmatrix} \boldsymbol{\theta}_k \\ \boldsymbol{\theta}_l \end{bmatrix}$$

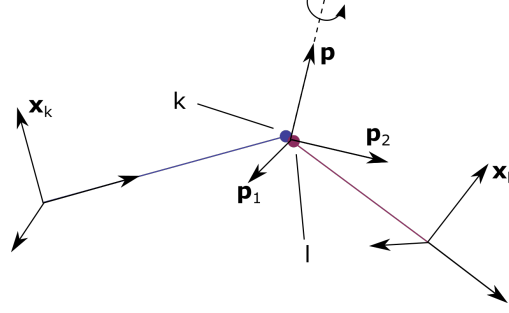


Fig. 4.39: Notations used for the derivation of the pin-joint constraint

The relations are inverted using a pseudo inverse, defined as $A^{-1*} = A^t(AA^t)^{-1}$. Using the pseudo-inverse, this equation is rewritten in the form of as:

$$\begin{bmatrix} \theta_k \\ \theta_l \end{bmatrix} = T_{kl} \tilde{\theta}_{kl}, \quad \text{with} \quad \tilde{\theta}_{kl} = \begin{bmatrix} \tilde{\theta}_1 \\ \tilde{\theta}_2 \\ \tilde{\theta}_3 \\ \tilde{\theta}_4 \end{bmatrix}, \quad T_{kl} = \begin{bmatrix} p_1^t/2 & p_1^t/2 \\ p_2^t/2 & p_2^t/2 \\ \hat{p}^t & 0 \\ 0 & \hat{p}^t \end{bmatrix}^{-1*}$$

If n elements $[e_1, \dots, e_n]$, are connected at a pin joint (constraint c), the relationship is extended as follows:

$$\begin{bmatrix} \theta_{e_1} \\ \dots \\ \theta_{e_n} \end{bmatrix} = T^c \tilde{\theta}^c, \quad \text{with} \quad \tilde{\theta}^c = \begin{bmatrix} \tilde{\theta}_1 \\ \tilde{\theta}_2 \\ \tilde{\theta}_{e_1} \\ \dots \\ \tilde{\theta}_{e_n} \end{bmatrix}, \quad T^c = \begin{bmatrix} p_1^t/n & \dots & p_1^t/n \\ p_2^t/n & \dots & p_2^t/n \\ \hat{p}^t & & 0 \\ & \ddots & \\ 0 & & \hat{p}^t \end{bmatrix}^{-1*}$$

Universal joint A universal joint transfers the rotational moment around two misaligned axes. Such joints are connecting only two elements, labelled j and k , and the axes are taken as the z axis of each element. The axis vectors are expressed in the global coordinates system and written \hat{z}_j and \hat{z}_k . Similar notations are used for the x and y axes. The DOF corresponding to the shared rotation between the two axes is written $\tilde{\theta}_1$. Each element has two additional DOFs that are free to rotate, noted $\tilde{\theta}_x$ and $\tilde{\theta}_y$. The constraint relationship between the original DOFs and the reduced DOFs is obtained by projecting the rotational DOFs of each element against the different axes. The relations are inverted using the pseudo-inverse, defined as $A^{-1*} = A^t(AA^t)^{-1}$. The constraints are then defined with:

$$\tilde{\theta}_c = \begin{bmatrix} \tilde{\theta}_1 \\ \tilde{\theta}_{x_j} \\ \tilde{\theta}_{y_j} \\ \tilde{\theta}_{x_k} \\ \tilde{\theta}_{y_k} \end{bmatrix}, \quad T_c = \begin{bmatrix} \hat{z}_j/2 & \hat{z}_k/2 \\ \hat{x}_j & 0 \\ \hat{y}_j & 0 \\ 0 & \hat{x}_k \\ 0 & \hat{y}_k \end{bmatrix}^{-1*}$$

$$\tilde{\theta}_c = \begin{bmatrix} \tilde{\theta}_1 \\ \tilde{\theta}_{x,e_1} \\ \tilde{\theta}_{y,e_1} \\ \vdots \\ \tilde{\theta}_{x,e_n} \\ \tilde{\theta}_{y,e_n} \end{bmatrix}, \quad T_c = \begin{bmatrix} \hat{z}_{e_1}^t/2 & \dots & \hat{z}_{e_n}^t/n \\ \hat{x}_{e_1}^t & & 0 \\ \hat{y}_{e_1}^t & & 0 \\ 0 & \ddots & 0 \\ 0 & & \hat{x}_{e_n}^t \\ 0 & \dots & \hat{y}_{e_n}^t \end{bmatrix}^{-1*}$$

Rigid-links

Rigid links and rigid elements impose a relationship between several degrees of freedom, and as such, can be treated as *linear multipoint* constraints. Rigid members can be used to join dissimilar elements together or model a link of large stiffness between two elastic bodies (see Cook [Coo01]). Mass properties for rigid link may be provided in the input file, in which case the mass matrix of a beam element is used for this rigid link.

A rigid link between the nodes j and k is considered, referred to as the element $j - k$. The six degrees of freedom of a given node, three displacements and three rotations, are noted $\mathbf{x} = [u_x, u_y, u_z, \theta_x, \theta_y, \theta_z]^t$ in the global system. The fact that the nodes j and k are rigidly connected is formally expressed as follows:

$$\mathbf{x}_k = \mathbf{A}_{jk} \mathbf{x}_j, \quad \mathbf{A}_{jk} = \begin{bmatrix} 1 & 0 & 0 & 0 & (z_k - z_j) & -(y_k - y_j) \\ 0 & 1 & 0 & -(z_k - z_j) & 0 & (x_k - x_j) \\ 0 & 0 & 1 & (y_k - y_j) & -(x_k - x_j) & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \begin{bmatrix} \mathbf{x}_j \\ \mathbf{x}_k \end{bmatrix} = \mathbf{T} \mathbf{x}_j = \begin{bmatrix} \mathbf{I}_6 \\ \mathbf{A}'_{jk} \end{bmatrix} \mathbf{x}_j \quad (4.114)$$

where the nodal coordinates (x, y, z) are expressed in the global system. The matrix \mathbf{T} expresses the relation between the condensed coordinates and the original coordinates.

In the general case, several joints may be coupled together with rigid links. An assembly of n joints is here assumed with the 6-DOFs of each joints written $\mathbf{x}_1, \dots, \mathbf{x}_n$. It is further assumed that the first joint is selected as leader. For each joint $j \in \{2, \dots, n\}$ a matrix \mathbf{A}_{1j} is formed according to (4.114). The matrices are built using the global coordinates of each joint pairs. For this given rigid assembly (or constraint c), the relation between the joint DOFs and the reduced leader DOF is:

$$\mathbf{x}^c = \mathbf{T}^c \tilde{\mathbf{x}}^c \quad \text{with} \quad \mathbf{x}^c = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \dots \\ \mathbf{x}_n \end{bmatrix}, \quad \mathbf{T}^c = \begin{bmatrix} \mathbf{I}_6 \\ \mathbf{A}_{12} \\ \dots \\ \mathbf{A}_{1n} \end{bmatrix}, \quad \tilde{\mathbf{x}}^c = \mathbf{x}_1$$

SubDyn detects rigid link assemblies and selects a leader node for the assembly. If one of the node is an interface node, it is selected as a leader node. The following restriction apply: the follower node cannot be a boundary node.

The constraint are applied after the full system has been assembled.

Craig-Bampton Reduction (theory)

Full system

The FEM equations of motion of SubDyn are written as follows:

$$[M]\{\ddot{U}\} + [C]\{\dot{U}\} + [K]\{U\} = \{F\} \quad (4.115)$$

where $[M]$ and $[K]$ are the global mass and stiffness matrices of the substructure beam frame, assembled from the element mass and stiffness matrices. Additionally, $[M]$ and $[K]$ contain the contribution from any specified $[M_{SSI}]$ and $[K_{SSI}]$ that are directly added to the proper partially restrained node DOF rows and column indexed elements.

U and F are the displacements and external forces along all of the DOFs of the assembled system. The damping matrix $[C]$ is not assembled from the element contributions, because those are normally unknown, but can be specified in different ways, as discussed in Section 4.2.5. A derivative with respect to time is represented by a dot, so that \dot{U} and \ddot{U} are the first- and second-time derivatives of U , respectively.

The number of DOFs associated with Eq. (4.115) can easily grow to the thousands for typical beam frame substructures. That factor, combined with the need for time-domain simulations of turbine dynamics, may seriously slow down the

computational efficiency of aeroelastic codes such as FAST (note that a typical wind turbine system model in ElastoDyn has about 20 DOFs). For this reason, a C-B methodology was used to recharacterize the substructure finite-element model into a reduced DOF model that maintains the fundamental low-frequency response modes of the structure. With the C-B method, the DOFs of the substructure can be reduced to about 10 (user defined, see also Section [Section 4.2.5](#)). This system reduction method was first introduced by [\[Hur64\]](#) and later expanded by [\[CB68\]](#).

CB-reduced system

In this section we present the generic Craig-Bampton technique. The specific application in SubDyn is presented in following sections. In a C-B reduction, the structure nodes are separated into two groups: 1) the boundary nodes (identified with a subscript “ R ” in what follows) that include the nodes fully restrained at the base of the structure and the interface nodes; and 2) the interior nodes (or leftover nodes, identified with a subscript “ L ”). Note that the DOFs of partially restrained or “free” nodes at the base of the structure are included in the “ L ” subset in this version of SubDyn that contains SSI capabilities.

The derivation of the system reduction is shown below. The system equation of motion of Eq. (4.115) can be partitioned as follows:

$$\begin{bmatrix} M_{RR} & M_{RL} \\ M_{LR} & M_{LL} \end{bmatrix} \begin{bmatrix} \ddot{U}_R \\ \ddot{U}_L \end{bmatrix} + \begin{bmatrix} C_{RR} & C_{RL} \\ C_{LR} & C_{LL} \end{bmatrix} \begin{bmatrix} \dot{U}_R \\ \dot{U}_L \end{bmatrix} + \begin{bmatrix} K_{RR} & K_{RL} \\ K_{LR} & K_{LL} \end{bmatrix} \begin{bmatrix} U_R \\ U_L \end{bmatrix} = \begin{bmatrix} F_R \\ F_L \end{bmatrix} \quad (4.116)$$

where the subscript R denotes the boundary DOFs (there are R DOFs), and the subscript L the interior DOFs (there are L DOFs). In Eq. (4.116), the applied forces include external forces (e.g., hydrodynamic forces and those transmitted through the TP to the substructure), gravity and pretension forces which are considered static forces lumped at each node.

The fundamental assumption of the C-B method is that the contribution to the displacement of the interior nodes can be simply approximated by a subset q_m ($q_m \leq L$) of the interior generalized DOFs (q_L). The relationship between physical DOFs and generalized DOFs can be written as:

$$\begin{bmatrix} U_R \\ U_L \end{bmatrix} = \begin{bmatrix} I & 0 \\ \Phi_R & \Phi_L \end{bmatrix} \begin{bmatrix} U_R \\ q_L \end{bmatrix} \quad (4.117)$$

where I is the identity matrix; Φ_R is the ($L \times R$) matrix of Guyan modes, which represents the physical displacements of the interior nodes for static, rigid body motions at the boundary (interface nodes’ DOFs, because the restrained nodes DOFs are locked by definition). By considering the homogeneous, static version of (4.116), the second row can be manipulated to yield:

$$[K_{LR}]U_R + [K_{LL}]U_L = 0 \quad (4.118)$$

Rearranging and considering yields:

$$\Phi_R = -K_{LL}^{-1}K_{LR} \quad (4.119)$$

where the brackets have been removed for simplicity. If the structure is unconstrained, the matrix Φ_R corresponds to rigid body modes, ensuring that the internal nodes follow the rigid body displacements imposed by the interface DOFs. This has been verified analytically using the stiffness matrix of a single beam element. Φ_L ($L \times L$ matrix) represents the internal eigenmodes, i.e., the natural modes of the system restrained at the boundary (interface and bottom nodes), and can be obtained by solving the eigenvalue problem:

$$K_{LL}\Phi_L = \omega^2 M_{LL}\Phi_L \quad (4.120)$$

The eigenvalue problem in Eq. (4.120) leads to the reduced basis of generalized modal DOFs q_m , which are chosen as the first few (m) eigenvectors that are arranged by increasing eigenfrequencies. Φ_L is mass normalized, so that:

$$\Phi_L^T M_{LL} \Phi_L = I \quad (4.121)$$

By then reducing the number of generalized DOFs to m ($\leq L$), Φ_m is the matrix $((L \times m))$ chosen to denote the truncated set of Φ_L (keeping m of the total internal modes, hence m columns), and Ω_m is the diagonal $(m \times m)$ matrix containing the corresponding eigenfrequencies (i.e., $\Phi_m^T K_{LL} \Phi_m = \Omega_m^2$). In SubDyn, the user decides how many modes to retain, including possibly zero or all modes. Retaining zero modes corresponds to a Guyan (static) reduction; retaining all modes corresponds to keeping the full finite-element model.

The C-B transformation is therefore represented by the coordinate transformation matrix T_{Φ_m} as:

$$\begin{bmatrix} U_R \\ U_L \end{bmatrix} = T_{\Phi_m} \begin{bmatrix} U_R \\ q_m \end{bmatrix}, \quad T_{\Phi_m} = \begin{bmatrix} I & 0 \\ \Phi_R & \Phi_m \end{bmatrix} \quad (4.122)$$

By using Eq. (4.122), the interior DOFs are hence transformed from physical DOFs to modal DOFs. By pre-multiplying both sides of Eq. (4.116) by $T_{\Phi_m}^T$ on the left and T_{Φ_m} on the right, and making use of Eq. (4.121), Eq. (4.116) can be rewritten as:

$$\begin{bmatrix} M_{BB} & M_{Bm} \\ M_{mB} & I \end{bmatrix} \begin{bmatrix} \ddot{U}_R \\ \ddot{q}_m \end{bmatrix} + \begin{bmatrix} C_{BB} & C_{Bm} \\ C_{mB} & C_{mm} \end{bmatrix} \begin{bmatrix} \dot{U}_R \\ \dot{q}_m \end{bmatrix} + \begin{bmatrix} K_{BB} & 0 \\ 0 & K_{mm} \end{bmatrix} \begin{bmatrix} U_R \\ q_m \end{bmatrix} = \begin{bmatrix} F_B \\ F_m \end{bmatrix} \quad (4.123)$$

where

$$\begin{aligned} M_{BB} &= M_{RR} + M_{RL}\Phi_R + \Phi_R^T M_{LR} + \Phi_R^T M_{LL} \Phi_R \\ C_{BB} &= C_{RR} + C_{RL}\Phi_R + \Phi_R^T C_{LR} + \Phi_R^T C_{LL} \Phi_R \\ K_{BB} &= K_{RR} + K_{RL}\Phi_R \\ M_{mB} &= \Phi_m^T M_{LR} + \Phi_m^T M_{LL} \Phi_R \\ C_{mB} &= \Phi_m^T C_{LR} + \Phi_m^T C_{LL} \Phi_R \\ K_{mm} &= \Phi_m^T K_{LL} \Phi_m = \Omega_m^2 \\ C_{mm} &= \Phi_m^T C_{LL} \Phi_m \\ F_B &= F_R + \Phi_R^T F_L \\ F_m &= \Phi_m^T F_L \end{aligned} \quad (4.124)$$

and $M_{Bm} = M_{mB}^T$, $C_{Bm} = C_{mB}^T$.

FEM formulation in SubDyn

Boundary nodes: fixed DOFs and rigid connection to TP

In this section we present the treatment of the boundary nodes: fixed DOFs are eliminated, and interface DOFs are condensed via a rigid connection to the TP reference point.

The boundary nodes are partitioned into those at the interface, \bar{U}_R , and those at the bottom, which are fixed:

$$U_R = \begin{bmatrix} \bar{U}_R \\ 0 \end{bmatrix} \quad (4.125)$$

The overhead bar here and below denotes matrices/vectors after the fixed-bottom boundary conditions are applied.

The interface nodes are assumed to be rigidly connected among one another and to the TP reference point, hence it is convenient to use rigid-body TP DOFs (one node with 6 DOFs at the TP reference point) in place of the interface DOFs. The interface DOFs, \bar{U}_R , and the TP DOFs are related to each other as follows:

$$\bar{U}_R = T_I U_{TP} \quad (4.126)$$

where T_I is a $(6NIN) \times 6$ matrix, NIN is the number of interface nodes, and U_{TP} is the 6 DOFs of the rigid transition piece. The matrix T_I can be written as follows:

$$T_I = \begin{bmatrix} 1 & 0 & 0 & 0 & \Delta Z_1 & -\Delta Y_1 \\ 0 & 1 & 0 & -\Delta Z_1 & 0 & -\Delta X_1 \\ 0 & 0 & 1 & \Delta Y_1 & -\Delta X_1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & 0 & 0 & \Delta Z_i & -\Delta Y_i \\ 0 & 1 & 0 & -\Delta Z_i & 0 & -\Delta X_i \\ 0 & 0 & 1 & \Delta Y_i & -\Delta X_i & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}, (i = 1, 2, \dots, NIN) \quad (4.127)$$

with

$$\begin{aligned} \Delta X_i &= X_{INi} - X_{TP} \\ \Delta Y_i &= Y_{INi} - Y_{TP} \\ \Delta Z_i &= Z_{INi} - Z_{TP} \end{aligned} \quad (4.128)$$

where $(X_{INi}, Y_{INi}, Z_{INi})$ are the coordinates of the i^{th} interface node and (X_{TP}, Y_{TP}, Z_{TP}) are the coordinates of the TP reference point within the global coordinate system.

In terms of TP DOFs, the system equation of motion (4.123) after the boundary constraints are applied (the rows and columns corresponding to the DOFs of the nodes that are restrained at the seabed are removed from the equation of motion) becomes:

$$\begin{bmatrix} \tilde{M}_{BB} & \tilde{M}_{Bm} \\ \tilde{M}_{mB} & I \end{bmatrix} \begin{bmatrix} \ddot{U}_{TP} \\ \ddot{q}_m \end{bmatrix} + \begin{bmatrix} \tilde{C}_{BB} & \tilde{C}_{Bm} \\ \tilde{C}_{mB} & C_{mm} \end{bmatrix} \begin{bmatrix} \dot{U}_{TP} \\ \dot{q}_m \end{bmatrix} + \begin{bmatrix} \tilde{K}_{BB} & 0 \\ 0 & K_{mm} \end{bmatrix} \begin{bmatrix} U_{TP} \\ q_m \end{bmatrix} = \begin{bmatrix} \tilde{F}_{TP} \\ F_m \end{bmatrix} \quad (4.129)$$

with

$$\begin{aligned} \tilde{M}_{BB} &= T_I^T \bar{M}_{BB} T_I, \quad \tilde{C}_{BB} = T_I^T \bar{C}_{BB} T_I, \quad \tilde{K}_{BB} = T_I^T \bar{K}_{BB} T_I \\ \tilde{M}_{Bm} &= T_I^T \bar{M}_{Bm}, \quad \tilde{C}_{Bm} = T_I^T \bar{C}_{Bm} \\ \tilde{F}_{TP} &= T_I^T F_B \end{aligned} \quad (4.130)$$

and $\tilde{M}_{mB} = \tilde{M}_{Bm}^T, \tilde{C}_{mB} = \tilde{C}_{Bm}^T$.

Equation (4.129) represents the equations of motion of the substructure after the C-B reduction. The total DOFs of the substructure are reduced from $(6 \times \text{total number of nodes})$ to $(6 + m)$.

During initialization, SubDyn calculates: the parameter matrices $\tilde{M}_{BB}, \tilde{M}_{mB}, \tilde{M}_{Bm}, \tilde{K}_{BB}, \Phi_m, \Phi_R, T_I$; constant load arrays; and the internal frequency matrix Ω_m . The substructure response at each time step can then be obtained by using the state-space formulation discussed in the next section.

Floating or fixed-bottom structure

Different formulations are used in SubDyn depending if the structure is “fixed-bottom” or “floating”.

The structure is considered to be “floating” if there is no reaction nodes.

The structure is considered to be “fixed-bottom” in any other case.

Loads

In this section, we detail the loads acting on the boundary (R) and interior (L) nodes, and the transition piece (TP) node.

External loads that are accounted for by SubDyn, such as the gravity loads or the pretension loads, are noted with the subscript g . External loads acting on the substructure and coming from additional modules, consisting for instance of hydrodynamic, mooring or soil loads, are noted with the subscript e . The coupling loads that ElastoDyn would transmit to SubDyn are noted with the subscript cpl . In the modular implementation, SubDyn does not receive these coupling loads from ElastoDyn, but instead receives displacements of the transition piece, and outputs the corresponding loads. This will be relevant for the state-space formulation, but for the purpose of this section, the coupling loads can be thought to be coming from ElastoDyn.

The external loads at the boundary nodes (R) consist of the SubDyn gravitational and cable loads (g), the ElastoDyn coupling loads (cpl), and the external loads from other modules (e):

$$F_R = F_{R,e} + F_{R,g} + F_{R,cpl} \quad (4.131)$$

The external loads acting on the internal nodes are similarly decomposed:

$$F_L = F_{L,e} + F_{L,g} \quad (4.132)$$

The loads at the transition piece node (TP) are related to the interface boundary nodes (\bar{R}) via the transformation matrix T_I , which assumes that the \bar{R} and TP nodes are rigidly connected:

$$F_{TP} = T_I^T \bar{F}_R \quad (4.133)$$

In particular, the coupling force exchanged between ElastoDyn and SubDyn is:

$$F_{TP,cpl} = T_I^T \bar{F}_{R,cpl} \quad (4.134)$$

The Guyan TP force, \tilde{F}_{TP} , and the CB force, F_m , given in Eq. (4.2.5) are then decomposed as follows:

$$\begin{aligned} \tilde{F}_{TP} &= F_{TP,cpl} + T_I^T [\bar{F}_{R,e} + \bar{F}_{R,g} + \bar{\Phi}_R^T (F_{L,e} + F_{L,g})] \\ F_m &= \Phi_m^t (F_{L,e} + F_{L,g}) \end{aligned} \quad (4.135)$$

Corrections to the baseline formulation (“GuyanLoadCorrection”)

The baseline FEM implementation needs to be corrected to account for the fact that loads are provided to SubDyn at the displaced positions, and to account for the rigid body motions in the floating case. The corrections are activated by setting the parameter **GuyanLoadCorrection** to True.

Rotation of coordinate system for floating

In the floating case, the FEM formulation needs to be rotated to the body frame. This is done when **GuyanLoadCorrection** is set to True. The CB and static modes are solved in a rotating frame of reference, that follows the rigid-body rotation of the Guyan modes. More details on this special case is found in section [Section 4.2.5](#).

Additional lever arm from external loads

The external loads that are applied on the substructure are computed at the location of the deflected structure. On the other hand, the finite element formulation expect loads to be provided relative to the undeflected position of the structure, or, if rigid body motions are present, relative to a reference undeflected position (see Figure Fig. 4.40). Nodal forces at a displaced node can be directly applied to the reference nodal position, but the mapping introduces a moment at the reference nodal position.

The parameter **GuyanLoadCorrection** in the input file is used to account for this extra nodal moment occurring due to the fact that the finite element loads are expected to be expressed at a reference position and not at the displaced position.

The mapping of nodal forces is done as follows when the parameter **GuyanLoadCorrection** is set to True. First, a reference undeflected position of the structure is defined, with two possible configurations whether the structure is “fixed” at the sea bed, or not. The two configurations are illustrated in Figure Fig. 4.40.

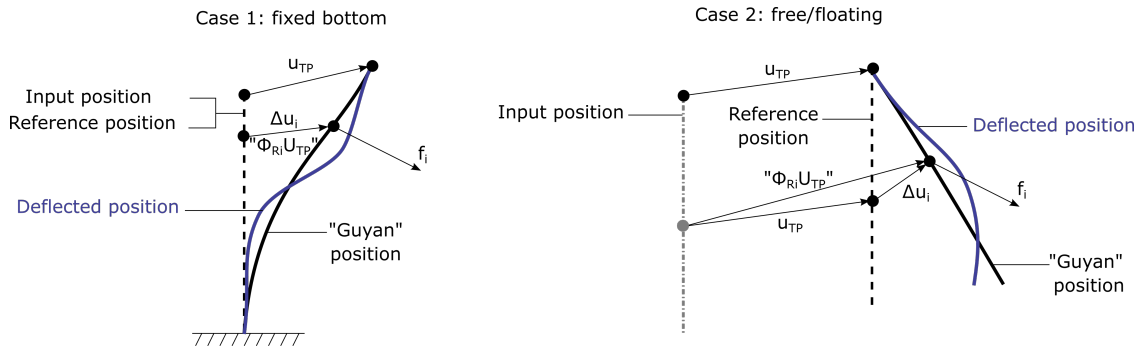


Fig. 4.40: Illustration for the additional moment occurring due to the distance between the deflected position of the structure and the reference position used for the finite element representation. For simplicity, the loads are assumed to act at the Guyan position instead of the true deflected position.

Second, the external loads are assumed to be applied on the “Guyan” deflected structure, instead of the fully deflected structure. The Craig-Bampton displacements are omitted to avoid the non-linear dependency between the input loads and the Craig-Bampton states. With this assumption, the external loads at the Guyan position are mapped to the reference position.

The additional moment is included for all external forces, including the gravitational forces. For a given node $i \in [R, L]$, and nodal force $f_i = f_{i,g} + f_{i,e}$, the following additional moment is computed:

$$\Delta m_i = \Delta u_i \times [f_i, g + f_i, e]$$

with the vector $\Delta u_i = \{\Delta u_{ix}, \Delta u_{iy}, \Delta u_{iz}\}$, defined differently depending on the reference position (fixed or free) and whether the node is an internal (L) or boundary node (R):

$$\begin{aligned} \text{(fixed bottom:)} \quad \Delta u_{ij} &= [\bar{\Phi}_R T_I]_{ij} U_{TP} \quad \text{for } i \in L, \text{ and, } \quad \Delta u_{ij} = [T_I]_{ij} U_{TP} \quad \text{for } i \in \bar{R} \quad (4.136) \\ \text{(free/floating:)} \quad \Delta u_{ij} &= [\bar{\Phi}_R T_I]_{ij} U_{TP} - U_{TP} \quad \text{for } i \in L, \text{ and, } \quad \Delta u_{ij} = [T_I]_{ij} U_{TP} - U_{TP} \quad \text{for } i \in \bar{R} \quad (4.137) \end{aligned}$$

where $j \in [x, y, z]$ and the subscript ij in $[\bar{\Phi}_R T_I]_{ij}$ indicates the row corresponding to node i and translational degree of freedom j . Boundary DOFs that are fixed have no displacements and thus no extra moment contribution. Boundary DOFs that are free are part of the internal DOF L in the implementation. The gravitational and cable forces at each node (that were computed at the initialization and stored in the constant vector F_G) are used to obtain $f_{i,g}$. It is noted that the g -contribution to the moment, Δm_i , is not a constant and needs to be computed at each time step.

To avoid adding more notations, all the load vectors used in this document will have the additional moment implicitly included when **GuyanLoadCorrection=True**. This applies e.g.: to $F_{R,e}, F_{L,e}, F_{R,g}, F_{L,g}$, where the following re-

placement is implied:

$$F_{R,e} = \begin{Bmatrix} \vdots \\ f_{ix,e} \\ f_{iy,e} \\ f_{iz,e} \\ m_{ix,e} \\ m_{iy,e} \\ m_{iz,e} \\ \vdots \end{Bmatrix} \longrightarrow F_{R,e} = \begin{Bmatrix} \vdots \\ f_{ix,e} \\ f_{iy,e} \\ f_{iz,e} \\ m_{ix,e} + \Delta m_{ix,e} \\ m_{iy,e} + \Delta m_{iy,e} \\ m_{iz,e} + \Delta m_{iz,e} \\ \vdots \end{Bmatrix} \quad (\text{GuyanLoadCorrection=True})$$

The dependency of the load vectors on U_{TP} introduces some complications for the state space representation, where for instance the B and F_X matrices should be modified to account for the dependency in U_{TP} in Eq. (4.148). The equation remains valid even if $F_{L,e}$ and $F_{L,g}$ contains a dependency in U_{TP} , but the matrix B shouldn't be used for the linearization (numerical differentiation is then preferred for simplicity). Similar considerations apply for Eq. (4.157).

The coupling load $F_{TP,cpl}$ given in Eq. (4.152) corresponds to the reaction force at the TP reference position. In the “free boundary condition” case, there is no need to correct this output load since the reference position is at the deflected position. For the “fixed boundary condition” case, the reference position does not correspond to the deflected position, so the reaction moment needs to be transferred to the deflected position as follows:

$$F_{TP,cpl} = \begin{Bmatrix} f_{TP,cpl} \\ m_{TP,cpl} \end{Bmatrix} \longrightarrow F_{TP,cpl} = \begin{Bmatrix} f_{TP,cpl} \\ m_{TP,cpl} - u_{TP} \times f_{TP,cpl} \end{Bmatrix} \quad (\text{GuyanLoadCorrection=True and Fixed BC})$$

The output equation $y_1 = -F_{TP,cpl}$ is then modified to include this extra contribution.

Damping specifications

There are three ways to specify the damping associated with the motion of the interface node in SubDyn: no damping, Rayleigh damping or user defined 6x6 matrix.

NOTE: Damping associated with joints is not documented yet and would change the developments below.

When **GuyanDampMod=0**, SubDyn assumes zero damping for the Guyan modes, and modal damping for the CB modes, with no cross couplings:

$$\begin{aligned} C_{BB} &= \tilde{C}_{BB} = 0 \\ C_{Bm} &= C_{mB} = \tilde{C}_{Bm} = \tilde{C}_{mB} = 0 \\ C_{mm} &= \tilde{C}_{mm} = 2\zeta\Omega_m \end{aligned} \quad (4.138)$$

In other words, the only damping matrix term retained is the one associated with internal DOF damping. This assumption has implications on the damping at the interface with the turbine system, as discussed in Section [Substructure Tower/Turbine Coupling](#). The diagonal ($m \times m$) ζ matrix contains the modal damping ratios corresponding to each retained internal mode. In SubDyn, the user provides damping ratios (in percent of critical damping coefficients) for the retained modes.

When **GuyanDampMod=1**, SubDyn assumes Rayleigh Damping for the Guyan modes, and modal damping for the CB modes, with no cross couplings:

$$\begin{aligned} \tilde{C}_{BB} &= \alpha \tilde{M}_{BB} + \beta \tilde{K}_{BB} \\ \tilde{C}_{Bm} &= \tilde{C}_{mB} = 0 \\ \tilde{C}_{mm} &= 2\zeta\Omega_m \end{aligned} \quad (4.139)$$

where α and β are the mass and stiffness proportional Rayleigh damping coefficients. The damping is directly applied to the tilde matrices, that is, the matrices related to the 6 DOF of the TP node.

The case **GuyanDampMod=2**, is similar to the previous case, except that the user specifies the 6×6 terms of \tilde{C}_{BB} .

Static-Improvement Method

To account for the effects of static gravity (member self-weight) and buoyancy forces, one would have to include all of the structural axial modes in the C-B reduction. This inclusion often translates into hundreds of modes to be retained for practical problems. An alternative method is thus promoted to reduce this limitation and speed up SubDyn. This method is denoted as SIM, and computes two static solutions at each time step: one based on the full system stiffness matrix and one based on the reduced stiffness matrix. The dynamic solution then proceeds as described in the previous sections, and at each time step the time-varying dynamic solution is superimposed on the difference between the two static solutions, which amounts to quasi-statically accounting for the contribution of those modes not directly included within the dynamic solution.

The SIM formulation provides a correction for the displacements of the internal nodes. The uncorrected displacements are now noted \hat{U}_L , while the corrected displacements are noted U_L . The SIM correction consists in an additional term U_L obtained by adding the total static deflection of all the internal DOFs (U_{L0}), and subtracting the static deflection associated with C-B modes (U_{L0m}), as cast in (4.140) :

$$U_L = \hat{U}_L + U_{L,\text{SIM}} = \hat{U}_L + \underbrace{U_{L0} - U_{L0m}}_{U_{L,\text{SIM}}} = \underbrace{\Phi_R U_R + \Phi_m q_m}_{\hat{U}_L} + \underbrace{\Phi_L q_{L0}}_{U_{L0}} - \underbrace{\Phi_m q_{m0}}_{U_{L0m}} \quad (4.140)$$

where q_{m0} and q_{L0} are the m and L modal coefficients that are assumed to be operating in a static fashion. These coefficients are calculated under the C-B hypothesis that the boundary nodes are fixed. The static displacement vectors can be calculated as follows:

$$K_{LL} U_{L0} = F_{L,e} + F_{L,g} \quad (4.141)$$

By pre-multiplying both sides times Φ_L^T , Eq. (4.141) can be rewritten as: $\Phi_L^T K_{LL} \Phi_L q_{L0} = \Phi_L^T (F_{L,e} + F_{L,g}) = \tilde{F}_L$ or, recalling that $\Phi_L^T K_{LL} \Phi_L = \Omega_L^2$, as: $\Omega_L^2 q_{L0} = \tilde{F}_L$, or equivalently in terms of U_{L0} :

$$U_{L0} = \Phi_L [\Omega_L^2]^{-1} \tilde{F}_L \quad (4.142)$$

Similarly:

$$K_{LL} U_{L0m} = F_{L,e} + F_{L,g} \rightarrow U_{L0m} = \Phi_m [\Omega_m^2]^{-1} \tilde{F}_m \quad (4.143)$$

with $\tilde{F}_m = \Phi_m^T (F_{L,e} + F_{L,g})$. Note that: $\dot{U}_{L0} = \dot{q}_{L0} = \dot{U}_{L0m} = \dot{q}_{m0} = 0$ and $\ddot{U}_{L0} = \ddot{q}_{L0} = \ddot{U}_{L0m} = \ddot{q}_{m0} = 0$.

In the floating case the loads F_L is rotated to the body coordinate system when “GuyanLoadCorrection” is True (see Section 4.2.5 for more details and Section 4.2.5 for the final equations used).

State-Space Formulation

A state-space formulation of the substructure structural dynamics problem was devised to integrate SubDyn within the FAST modularization framework. The state-space formulation was developed in terms of inputs, outputs, states, and parameters. The notations highlighted here are consistent with those used in Jonkman (2013). Inputs (identified by u) are a set of values supplied to SubDyn that, along with the states, are needed to calculate future states and the system’s output. Outputs (y) are a set of values calculated by and returned from SubDyn that depend on the states, inputs, and/or parameters through output equations (with functions Y). States are a set of internal values of SubDyn that are influenced by the inputs and used to calculate future state values and the output. In SubDyn, only continuous states are considered. Continuous states (x) are states that are differentiable in time and characterized by continuous time differential equations (with functions X). Parameters (p) are a set of internal system values that are independent of the states and inputs. Furthermore, parameters can be fully defined at initialization and characterize the system’s state equations and output equations.

In SubDyn, the inputs are defined as:

$$u = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{bmatrix} = \begin{bmatrix} U_{TP} \\ \dot{U}_{TP} \\ \ddot{U}_{TP} \\ F_{L,e} \\ F_{R,e} \end{bmatrix} \quad (4.144)$$

where F_L are the hydrodynamic forces on every interior node of the substructure from HydroDyn, and F_{HDR} are the analogous forces at the boundary nodes; U_{TP} , \dot{U}_{TP} , and \ddot{U}_{TP} are TP deflections (6 DOFs), velocities, and accelerations, respectively. For SubDyn in stand-alone mode (uncoupled from FAST), $F_{L,e}$ and $F_{R,e}$ are assumed to be zero.

In first-order form, the states are defined as:

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} q_m \\ \dot{q}_m \end{bmatrix} \quad (4.145)$$

From the system equation of motion, the state equation corresponding to Eq. (4.129) can be written as a standard linear system state equation:

$$\dot{x} = X = Ax + Bu + F_X \quad (4.146)$$

These state matrices are obtained by isolating the mode accelerations, \ddot{q}_m from the second block row of Eq. (4.129) as:

$$\ddot{q}_m = \underbrace{\Phi_m^T (F_{L,e} + F_{L,g})}_{F_m} - \tilde{M}_{mB} \ddot{U}_{TP} - \tilde{C}_{mB} \dot{U}_{TP} - \tilde{C}_{mm} \dot{q}_m - \tilde{K}_{mm} q_m \quad (4.147)$$

leading to the following identification:

$$A = \begin{bmatrix} 0 & I \\ -\tilde{K}_{mm} & -\tilde{C}_{mm} \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -\tilde{C}_{mB} & -\tilde{M}_{mB} & \Phi_m^T & 0 \end{bmatrix}, \quad F_X = \begin{bmatrix} 0 \\ \Phi_m^T F_{L,g} \end{bmatrix} \quad (4.148)$$

In SubDyn, the outputs to the ElastoDyn module are the coupling (reaction) forces at the transition piece $F_{TP,cpl}$:

$$y_1 = Y_1 = -F_{TP,cpl} \quad (4.149)$$

By examining Eq. (4.129) and Eq. (4.135), the force is extracted from the first block row as:

$$\begin{aligned} F_{TP,cpl} = & \tilde{M}_{BB} \ddot{U}_{TP} + \tilde{M}_{Bm} \ddot{q}_m \\ & + \tilde{C}_{BB} \dot{U}_{TP} + \tilde{C}_{Bm} \dot{q}_m + \tilde{K}_{BB} U_{TP} - T_I^T (\bar{F}_{R,e} + \bar{F}_{R,g} + \bar{\Phi}_R (F_{L,e} + F_{L,g})) \end{aligned} \quad (4.150)$$

Inserting the expression of \ddot{q}_m into F_{TP} leads to:

$$\begin{aligned} F_{TP,cpl} = & \left[-\tilde{M}_{Bm} \tilde{K}_{mm} \right] q_m + \left[\tilde{C}_{Bm} - \tilde{M}_{Bm} \tilde{C}_{mm} \right] \dot{q}_m \\ & + \left[\tilde{K}_{BB} \right] U_{TP} + \left[\tilde{C}_{BB} - \tilde{M}_{Bm} \tilde{C}_{mB} \right] \dot{U}_{TP} + \left[\tilde{M}_{BB} - \tilde{M}_{Bm} \tilde{M}_{mB} \right] \ddot{U}_{TP} \\ & + \left[\tilde{M}_{Bm} \Phi_m^T - T_I^T \bar{\Phi}_R^T \right] (F_{L,e} + F_{L,g}) + \left[-T_I^T \right] (\bar{F}_{R,e} + \bar{F}_{R,g}) \end{aligned} \quad (4.151)$$

The output equation for y_1 can now be identified as:

$$-Y_1 = F_{TP,cpl} = C_1 x + D_1 \ddot{u} + F_{Y1} \quad (4.152)$$

where

$$\begin{aligned}
C_1 &= [-\tilde{M}_{Bm}\tilde{K}_{mm} \quad \tilde{C}_{Bm} - \tilde{M}_{Bm}\tilde{C}_{mm}] \\
D_1 &= [\tilde{K}_{BB} \quad \tilde{C}_{BB} - \tilde{M}_{Bm}\tilde{C}_{mB} \quad \tilde{M}_{BB} - \tilde{M}_{Bm}\tilde{M}_{mB} \quad \tilde{M}_{Bm}\Phi_m^T - T_I^T\bar{\Phi}_R^T \quad -T_I^T] \\
F_{Y1} &= [\tilde{M}_{Bm}\Phi_m^T F_{L,g} - T_I^T(\bar{F}_{R,g} + \bar{\Phi}_R^T F_{L,g})] \\
\bar{u} &= \begin{bmatrix} U_{TP} \\ \dot{U}_{TP} \\ \ddot{U}_{TP} \\ F_{L,e} \\ \bar{F}_{R,e} \end{bmatrix}
\end{aligned} \tag{4.153}$$

Note that the overbar is used on the input vector to denote that the forces apply to the interface nodes only.

The outputs to HydroDyn and other modules are the deflections, velocities, and accelerations of the substructure. Two meshes are introduced to store these motions, noted y_2 and y_3 . The two meshes have different displacements for the floating case, where y_2 only has the Guyan motion, whereas y_3 has the full elastic motion. This distinction is not made in the following equations. The full elastic motion is assumed in y_2 in this section. For more details, see section [Section 4.2.5](#). The output motion is:

$$y_2 = Y_2 = \begin{bmatrix} \bar{U}_R \\ U_L \\ \dot{\bar{U}}_R \\ \dot{U}_L \\ \ddot{\bar{U}}_R \\ \ddot{U}_L \end{bmatrix} \tag{4.154}$$

From the CB coordinate transformation (Eq. (4.122)), and the link between boundary nodes and TP node (Eq. (4.126)) the motions are given as:

$$\begin{aligned}
\bar{U}_R &= T_I U_{TP}, & \bar{U}_L &= \bar{\Phi}_R \bar{U}_R + \Phi_m q_m + U_{L,SIM} \\
\dot{\bar{U}}_R &= T_I \dot{U}_{TP}, & \dot{\bar{U}}_L &= \bar{\Phi}_R \dot{\bar{U}}_R + \Phi_m \dot{q}_m \\
\ddot{\bar{U}}_R &= T_I \ddot{U}_{TP}, & \ddot{\bar{U}}_L &= \bar{\Phi}_R \ddot{\bar{U}}_R + \Phi_m \ddot{q}_m
\end{aligned} \tag{4.155}$$

The expression for y_2 motions contains the optional SIM contribution (see [Section 4.2.5](#)). Using the expression of \ddot{q}_m from Eq. (4.2.5), the internal accelerations are:

$$\ddot{\bar{U}}_L = \bar{\Phi}_R T_I \ddot{U}_{TP} + \Phi_m \left[\Phi_m^T (F_{L,e} + F_{L,g}) - \tilde{M}_{mB} \ddot{U}_{TP} - \tilde{C}_{mB} \dot{U}_{TP} - \tilde{C}_{mm} \dot{q}_m - \tilde{K}_{mm} q_m \right] \tag{4.156}$$

In the floating case, some subtle changes are introduced: 1) the Guyan part of the motion are replaced by the analytical rigid body motion, 2) the elastic displacements are set to zero to avoid a coupling issue with HydroDyn (see details in section [Section 4.2.5](#)). Because of 2), a third mesh was introduced, y_3 , which always contains the full elastic motion (full elastic displacements, velocities and accelerations, including the analytical rigid body motion in the floating case). The third mesh is used for instance by Moordyn.

The output equation for y_2 can then be written as:

$$Y_2 = C_2 x + D_2 u + F_{Y2} \tag{4.157}$$

where

$$\begin{aligned}
 C_2 &= \begin{bmatrix} 0 & 0 \\ \Phi_m & 0 \\ 0 & 0 \\ 0 & \Phi_m \\ 0 & 0 \\ -\Phi_m \tilde{K}_{mm} & -\Phi_m \tilde{C}_{mm} \end{bmatrix} \\
 D_2 &= \begin{bmatrix} T_I & 0 & 0 & 0 & 0 \\ \bar{\Phi}_R T_I & 0 & 0 & 0 & 0 \\ 0 & T_I & 0 & 0 & 0 \\ 0 & \bar{\Phi}_R T_I & 0 & 0 & 0 \\ 0 & 0 & T_I & 0 & 0 \\ 0 & -\Phi_m \tilde{C}_{mB} & \bar{\Phi}_R T_I - \Phi_m \tilde{M}_{mB} & \Phi_m \Phi_m^T & 0 \end{bmatrix} \\
 F_{Y2} &= \begin{bmatrix} 0 \\ U_{L,SIM} \\ 0 \\ 0 \\ 0 \\ \Phi_m \Phi_m^T F_{L,g} \end{bmatrix}
 \end{aligned} \tag{4.158}$$

Outputs and Time Integration

Nodal Loads Calculation

We start by introducing how element loads are computed, before detailing how nodal loads are obtained.

Element Loads:

SubDyn calculates 12-vector element loads in the element coordinate system using the global motion of the element:

$$\begin{aligned}
 \text{Element Inertia load: } F_{I,12}^e &= [D_{c,12}]^T [m] \ddot{U}_{e,12} \\
 \text{Element Stiffness load: } F_{S,12}^e &= [D_{c,12}]^T [k] [\hat{U}_e + U_{L,SIM}]_{12}
 \end{aligned} \tag{4.159}$$

where $[k]$ and $[m]$ are element stiffness and mass matrices expressed in the global frame, $D_{c,12}$ is a 12x12 matrix of DCM for a given element, the subscript 12 indicates that the 12 degrees of freedom of the element are considered, and U_e and \ddot{U}_e are element nodal deflections and accelerations respectively, which can be obtained from Eq. (4.154) and may contain the static displacement contribution $U_{L,SIM}$. There is no good way to quantify the damping forces for each element, so the element damping forces are not calculated.

Nodal loads

For a given element node, the loads are the 6-vector with index 1-6 or 7-12 for the first or second node respectively. By convention, the 6-vector is multiplied by -1 for the first node and +1 for the second node of the element:

$$F_6^{n_1} = -F_{12}^e(1 : 6), \quad F_6^{n_2} = +F_{12}^e(7 : 12) \tag{4.160}$$

The above applies for the inertial and stiffness loads.

Member nodal loads requested by the user

The user can output nodal loads for a set of members (see Section 4.2.5).

For the user requested member nodal outputs, the loads are either: 1) the appropriate 6-vector at the member end nodes, or, 2) the average of the 6-vectors from the two elements surrounding a node for the nodes in the middle of a member.

When averaging is done, the 12-vectors of both surrounding elements are expressed using the DCM of the member where outputs are requested.

“AllOuts” nodal loads

For “AllOuts” nodal outputs, the loads are not averaged and the 6-vector (with the appropriate signs) are directly written to file.

Reaction nodal loads (See [Section 4.2.5](#))

Reaction Calculation

The reactions at the base of the structure are the nodal loads at the base nodes.

Additionally, the user may request an overall reaction \vec{R} (six forces and moments) lumped at the center of the sub-structure (tower centerline) and mudline, i.e., at the reference point (0,0,-**WtrDpth**) in the global reference frame, with **WtrDpth** denoting the water depth.

To obtain this overall reaction, the forces and moments at the N_{React} restrained nodes are expressed in the global coordinate frame and gathered into the vector F_{React} , which is a $(6*N_{\text{React}})$ array. For a given reaction node, the 6-vector of loads is obtained by summing the nodal load contributions from all the elements connected to that node expressed in the global frame (no account of the sign is done here), and subtracting the external loads (F_{HDR}) applied on this node. The loads from all nodes, F_{React} , are then rigidly-transferred to (0, 0, -WtrDpth) to obtain the overall six-element array \vec{R} :

$$\vec{R} = \begin{bmatrix} F_X \\ \vdots \\ M_Z \end{bmatrix} = T_{\text{React}} F_{\text{React}} \quad (4.161)$$

where T_{React} is a $(66N_{\text{React}})$ matrix, as follows:

$$T_{\text{React}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & \dots & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & \dots & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & \dots & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -\Delta Z_1 & \Delta Y_1 & 1 & 0 & 0 & \dots & 0 & -\Delta Z_{N_{\text{react}}} & \Delta Y_{N_{\text{react}}} & 1 & 0 & 0 \\ \Delta Z_1 & 0 & -\Delta X_1 & 0 & 1 & 0 & \dots & \Delta Z_{N_{\text{react}}} & 0 & -\Delta X_{N_{\text{react}}} & 0 & 1 & 0 \\ \Delta Y_1 & \Delta X_1 & 0 & 0 & 0 & 1 & \dots & \Delta Y_{N_{\text{react}}} & \Delta X_{N_{\text{react}}} & 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.162)$$

where X_i , Y_i , and Z_i ($i = 1..N_{\text{React}}$) are coordinates of the boundary nodes with respect to the reference point.

Time Integration

At time $t = 0$, the initial states are specified as initial conditions (all assumed to be zero in SubDyn) and the initial inputs are supplied to SubDyn. During each subsequent time step, the inputs and states are known values, with the inputs $u(t)$ coming from ElastoDyn and HydroDyn, and the states $x(t)$ known from the previous time-step integration. All of the parameter matrices are calculated in the SubDyn initiation module. With known $u(t)$ and $x(t)$, $\dot{x}(t)$ can be calculated using the state equation $\dot{x}(t) = X(u, x, t)$ (see Eq. (4.146)), and the outputs $y_1(t)$ and $y_2(t)$ can be calculated solving Eqs. (4.152) and (4.157). The element forces can also be calculated using Eq. (4.159). The next time-step states $x(t + \Delta t)$ are obtained by integration:

$$[u(t), \dot{x}(t), x(t)] \xrightarrow{\text{Integrate}} x(t + \Delta t) \quad (4.163)$$

For loose coupling, SubDyn uses its own integrator, whereas for tight coupling, the states from all the modules will be integrated simultaneously using an integrator in the glue-code. SubDyn’s built-in time integrator options for loose coupling are:

- Fourth-order explicit Runge-Kutta
- Fourth-order explicit Adams-Bashforth predictor
- Fourth-order explicit Adams-Bashforth-Moulton predictor-corrector
- Implicit second-order Adams-Moulton.

For more information, consult any numerical methods reference, e.g., [CC10].

Summary of the formulation implemented

This section summarizes the equations currently implemented in SubDyn, with the distinction between floating and fixed bottom cases. We introduce the operators R_{g2b} (rotation global to body) and R_{b2g} (rotation body to global), which act on the array on the right of the operator. The operators rotate the individual 3-vectors present in an array. When applied to load vectors (e.g. F_L), the rotations actually is applied to the loads on the full system, before the loads are transferred to the reduced system by use of the T matrix.

State equation

Fixed-bottom case

$$\ddot{q}_m = \Phi_m^T F_L - \tilde{M}_{mB} \ddot{U}_{TP} - \tilde{C}_{mm} \dot{q}_m - \tilde{K}_{mm} q_m \quad (4.164)$$

Note: F_L contains the “extra moment” if user-requested with **GuyanLoadCorrection**.

Floating case without “GuyanLoadCorrection”

$$\ddot{q}_m = \Phi_m^T F_L - \tilde{M}_{mB} \ddot{U}_{TP} - \tilde{C}_{mm} \dot{q}_m - \tilde{K}_{mm} q_m \quad (4.165)$$

Notes: F_L does not contain the “extra moment”.

Floating case with “GuyanLoadCorrection”

$$\ddot{q}_m = \Phi_m^T R_{g2b} F_L - \tilde{M}_{mB} R_{g2b} \ddot{U}_{TP} - \tilde{C}_{mm} \dot{q}_m - \tilde{K}_{mm} q_m \quad (4.166)$$

Notes: F_L does not contain the “extra moment”. The (external + gravity) loads and the acceleration of the TP are rotated to the body coordinate system.

Output: interface reaction

Fixed bottom case

$$\begin{aligned} -Y_1 = F_{TP,cpl} = \left\{ \begin{matrix} f_{TP,cpl} \\ m_{TP,cpl} \end{matrix} \right\} &= \begin{bmatrix} -\tilde{M}_{Bm} \tilde{K}_{mm} \end{bmatrix} q_m + \begin{bmatrix} -\tilde{M}_{Bm} \tilde{C}_{mm} \end{bmatrix} \dot{q}_m \\ &+ \begin{bmatrix} \tilde{K}_{BB} \end{bmatrix} U_{TP} + \begin{bmatrix} \tilde{C}_{BB} \end{bmatrix} \dot{U}_{TP} + \begin{bmatrix} \tilde{M}_{BB} - \tilde{M}_{Bm} \tilde{M}_{mB} \end{bmatrix} \ddot{U}_{TP} \\ &+ \begin{bmatrix} \tilde{M}_{Bm} \Phi_m^T \end{bmatrix} F_L + \begin{bmatrix} -T_I^T \bar{\Phi}_R^T \end{bmatrix} F_L + \begin{bmatrix} -T_I^T \end{bmatrix} \bar{F}_R \end{aligned} \quad (4.167)$$

Note: F_L and \bar{F}_R contains the “extra moment” if user-requested. If this is the case, the following additional term is added to the moment part of Y_1 , $m_{Y_1,extra} = u_{TP} \times f_{TP,cpl}$.

Floating case without “GuyanLoadCorrection”

$$\begin{aligned} -Y_1 = F_{TP,cpl} &= \begin{bmatrix} -\tilde{M}_{Bm} \tilde{K}_{mm} \end{bmatrix} q_m + \begin{bmatrix} -\tilde{M}_{Bm} \tilde{C}_{mm} \end{bmatrix} \dot{q}_m \\ &+ \begin{bmatrix} \tilde{K}_{BB} \end{bmatrix} U_{TP} + \begin{bmatrix} \tilde{C}_{BB} \end{bmatrix} \dot{U}_{TP} + \begin{bmatrix} \tilde{M}_{BB} - \tilde{M}_{Bm} \tilde{M}_{mB} \end{bmatrix} \ddot{U}_{TP} \\ &+ \begin{bmatrix} \tilde{M}_{Bm} \Phi_m^T \end{bmatrix} F_L + \begin{bmatrix} -T_I^T \bar{\Phi}_R^T \end{bmatrix} F_L + \begin{bmatrix} -T_I^T \end{bmatrix} \bar{F}_R \end{aligned} \quad (4.168)$$

Note: F_L and \bar{F}_R do not contain the “extra moment”.

Floating case with “GuyanLoadCorrection”

$$\begin{aligned}
 -Y_1 = F_{TP,cpl} = & R_{b2g} \left[-\tilde{M}_{Bm} \tilde{K}_{mm} \right] q_m + R_{b2g} \left[-\tilde{M}_{Bm} \tilde{C}_{mm} \right] \dot{q}_m \\
 & + \left[\tilde{K}_{BB} \right] U_{TP} + \left[\tilde{C}_{BB} \right] \dot{U}_{TP} + \left[\tilde{M}_{BB} - \tilde{M}_{Bm} \tilde{M}_{mB} \right] \ddot{U}_{TP} \\
 & + R_{b2g} \left[\tilde{M}_{Bm} \Phi_m^T \right] R_{g2b} F_L + \left[-T_I^T \bar{\Phi}_R^T \right] F_{L,extra} + \left[-T_I^T \right] \bar{F}_{R,extra}
 \end{aligned} \tag{4.169}$$

Notes: 1) $F_{L,extra}$ and $F_{R,extra}$ contain the “extra moment” in the Guyan contribution; 2) For the Craig-Bampton contribution, the loads are rotated to the body coordinate system using the operator R_{g2b} (global to body); 3) The rotation $R_{b2g} \tilde{M}_{Bm} \tilde{M}_{mB} R_{g2b}$ is not carried out since it introduced stability issues.

Output: nodal motions

Fixed-bottom case

$$\begin{aligned}
 \bar{U}_R &= T_I U_{TP}, & \bar{U}_L &= \bar{\Phi}_R \bar{U}_R + \Phi_m q_m + U_{L,SIM} \\
 \dot{\bar{U}}_R &= T_I \dot{U}_{TP}, & \dot{\bar{U}}_L &= \bar{\Phi}_R \dot{\bar{U}}_R + \Phi_m \dot{q}_m \\
 \ddot{\bar{U}}_R &= T_I \ddot{U}_{TP}, & \ddot{\bar{U}}_L &= \bar{\Phi}_R \ddot{\bar{U}}_R + \Phi_m \left[\Phi_m^T F_L - \tilde{M}_{mB} \ddot{U}_{TP} - \tilde{C}_{mm} \dot{q}_m - \tilde{K}_{mm} q_m \right]
 \end{aligned} \tag{4.170}$$

Note: F_L contains the “extra moment” if user-requested with **GuyanLoadCorrection**. The meshes y_2 and y_3 are identical (Guyan displacements computed using Φ_R , elastic displacements are included, together with the elastic velocities/accelerations).

Floating case

$$\begin{aligned}
 \bar{U}_R &= U_{R,rigid}, & \bar{U}_L &= U_{L,rigid} + 0 \cdot R_{b2g} (\Phi_m q_m + U_{L,SIM}) \\
 \dot{\bar{U}}_R &= \dot{U}_{R,rigid}, & \dot{\bar{U}}_L &= \dot{U}_{L,rigid} + R_{b2g} \Phi_m \dot{q}_m \\
 \ddot{\bar{U}}_R &= \ddot{U}_{R,rigid}, & \ddot{\bar{U}}_L &= \ddot{U}_{L,rigid} + R_{b2g} \Phi_m \left[\Phi_m^T R_{g2b} F_L - \tilde{M}_{mB} R_{g2b} \ddot{U}_{TP} - \tilde{C}_{mm} \dot{q}_m - \tilde{K}_{mm} q_m \right]
 \end{aligned} \tag{4.171}$$

where: 1) F_L does not contain the extra moment, 2) the operators R_{g2b} and R_{b2g} are when GuyanLoadCorrection is True, 3) the elastic displacements are set to 0 for stability purposes (assuming that these are small) in y_2 (used by HydroDyn), but not set to 0 for y_3 (used by MoorDyn). 4) the Guyan motion ($U_{L,rigid}$) is computed using the exact rigid body motions. For a given node P , located at the position $r_{IP,0}$ from the interface in the undisplaced configuration, the position (from the interface point), displacement, translational velocity and acceleration due to the rigid body motion are:

$$\begin{aligned}
 r_{IP} &= R_{b2g} r_{IP,0}, & u_P &= r_{IP} - r_{IP,0} + u_{TP}, \\
 \dot{u}_P &= \dot{u}_{TP} + \omega_{TP} \times r_{IP}, & \ddot{u}_P &= \ddot{u}_{TP} + \dot{\omega}_{TP} \times r_{IP} + \omega_{TP} \times (\omega_{TP} \times r_{IP})
 \end{aligned}$$

where ω_{TP} is the angular velocity at the transition piece. The small angle rotations, angular velocities and accelerations of each nodes, due to the rigid body rotation, are the same as the interface values, θ_{TP} , ω_{TP} and $\dot{\omega}_{TP}$, so that:

$$U_{P,rigid} = \{u_P; \theta_{TP}\}^T, \quad \dot{U}_{P,rigid} = \{\dot{u}_P; \omega_{TP}\}^T, \quad \ddot{U}_{P,rigid} = \{\ddot{u}_P; \dot{\omega}_{TP}\}^T$$

where P is a point belonging to the R- or L-set of nodes.

Outputs to file:

Motions: nodal motions written to file are in global coordinates, and for the floating case they contain the elastic motion $\bar{U}_L = U_{L,\text{rigid}} + \Phi_m q_m + U_{L,\text{SIM}}$ (whereas these elastic motions are not returned to the glue code)

Loads: Nodal loads are written to file in the element coordinate system. The procedure are the same for fixed-bottom and floating cases.

Known Limitations and Future Work

The following list contains known current limitations in the code:

- Tight coupling is not yet supported.
- Only nontapered two-node Euler-Bernoulli (E-B) or Timoshenko (T) element formulations are available. (In the future, tapered E-B and tapered Timoshenko element formulations will be implemented.)
- Only straight circular members are permitted. (In the future, a generic cross section will be allowed.)
- The number of elements per member (**NDiv**) is constant throughout the structure.
- Internal matrices are not stored in sparse form, limiting the total number of possible nodes/DOFs to about 300/1800.
- The dynamics system reduction is performed in the absence of external loading (e.g., hydrodynamic added mass).
- Gravitational loading does not impact the global substructure stiffness.
- Loads (gravitational, inertial, hydrodynamic) can only be applied as concentrated loads at element nodes; distributed loads (per unit length) are not yet supported.
- The overlap of multiple members connected to a single joint is not modeled with super-elements.
- Member-level outputs are only available for up to nine nodes of up to nine members (although the **OutAll** flag can generate further outputs at the member end nodes).
- No graphics/animation capability is yet available to visualize the substructure geometry, modes, motion, and loads.

Appendix A. OC4 Jacket Input File

SubDyn's primary input file (OC4 Jacket SubDyn's Input File):

This file includes information on the integration method (e.g., Adams-Bashforth 4th order), numerical-solution parameters (e.g., integration time interval, static solver flag, number of modes to retain within the Craig-Bampton reduction), finite element analysis information (beam element model, number of elements per member), and the geometric definition of the beam members via joints, member connectivity, and member cross-sectional properties. This file also specifies any SSI input files (soil/pile stiffness and mass matrices).

Appendix B. OC4 Jacket Driver File

SubDyn's Driver Input File (OC4 Jacket Driver File):

This file includes information on the environmental conditions (gravity and water depth), numerical-solution parameters (e.g., integration time interval, number of time-steps), TP reference point coordinates in global reference frame, rotation angle of the structure geometry in degrees about the global Z axis, the input mode for the TP reference point displacements, velocities, and accelerations (steady-state or time-series from file) and any related input file if not steady-state input.

Appendix C. OC4 Jacket SSI Input File

SubDyn's SSI File (OC4 Jacket SSI File):

This file includes information on the stiffness of embedded-pile/soil combination.

Appendix D. List of Output Channels

This is a list of all possible output parameters for the SubDyn module. The names are grouped by meaning, but can be ordered in the OUTPUT CHANNELS section of the SubDyn input file as the user sees fit. $M_{\alpha}N_{\beta}$, refers to node β of member α , where α is a number in the range [1,9] and corresponds to row α in the MEMBER OUTPUT LIST table (see Section) and β is a number in the range [1,9] and corresponds to node β in the **NodeCnt** list of that table entry.

Some outputs are in the SS reference coordinate system (global inertial-frame coordinate system), and end with the suffix *ss*; others refer to the local (member) reference system and they have suffixes “Xe”, “Ye”, or “Ze” (see Section 7).

Table C-1. List of Output Channels.

Channel Name(s)	Units	Description
<i>Base and Interface Reaction Loads</i>		
ReactFXss, ReactFYss, ReactFZss, ReactMXss, ReactMYss, React- MZss,	(N), (N), (N), (Nm), (Nm), (Nm)	Total base reaction forces and mo- ments at the (0.,0.,- WtrDpth) location in SS coordinate system
IntfFXss, IntfFYss, IntfFZss, IntfMXss, IntfMYss, IntfMZss,	(N), (N), (N), (Nm), (Nm), (Nm)	Total interface reaction forces and moments at the TP reference point (platform reference point) location in SS coor- dinate system
<i>Interface Kinematics</i>		
IntfTDXss, IntfTDYss, IntfTDZss, IntfRDXss, IntfRDYss IntfRDZss	(m), (m), (m), (rad), (rad), (rad)	Displacements and rotations of the TP reference point (platform reference point) location in SS coordinate system
IntfTAXss, IntfTAYss, IntfTAZss, IntfRAXss, IntfRAYss IntfRAZss	(m/s^2) , (m/s^2) , (m/s^2) , (rad/s^2) , (rad/s^2) , (rad/s^2)	Translational and rotational acceler- ations of the TP reference point (platform reference point) location in SS coordinate system
<i>Modal Parameters</i>		
SSqm01-SSqm99	(-)	C-B modal variables (up to first 99)
SSqmd01-SSqmd99	(1/s)	First time-derivatives of C-B modal variables (up to first 99)
SSqmdd01-SSqmdd99	$(1/s^2)$	Second time-derivatives of C-B modal variables (up to first 99)
<i>Node Kinematics</i>		
$M\alpha N\beta$ TDxss, $M\alpha N\beta$ TDyss, $M\alpha N\beta$ TDzss,	(m)	Nodal translational displacements of $M\alpha N\beta$ (up to 81 designated locations) in SS coordinate system
$M\alpha N\beta$ RDxe, $M\alpha N\beta$ RDye, $M\alpha N\beta$ RDze	(rad)	Nodal rotational displacements of $M\alpha N\beta$ (up to 81 designated locations) in member local coordinate system
$M\alpha N\beta$ TAxe, $M\alpha N\beta$ TAye, $M\alpha N\beta$ TAze	(m/s^2)	Nodal translational accelerations of $M\alpha N\beta$ (up to 81 designated locations) in member local coordinate system
$M\alpha N\beta$ RAxe, $M\alpha N\beta$ RAye, $M\alpha N\beta$ RAze	(rad/s^2)	Nodal rotational accelerations of $M\alpha N\beta$ (up to 81 designated locations) in member local coordinate system
<i>Node Forces and Moments</i>		
$M\alpha N\beta$ FKxe, $M\alpha N\beta$ FKye, $M\alpha N\beta$ FKze $M\alpha N\beta$ MKxe, $M\alpha N\beta$ MKye, $M\alpha N\beta$ MKze	(N), (N), (N), (Nm), (Nm), (Nm)	Static (elastic) component of reac- tion forces and moments at $M\alpha N\beta$ along local member co- ordinate system
$M\alpha N\beta$ FMxe, $M\alpha N\beta$ FMye, $M\alpha N\beta$ FMze $M\alpha N\beta$ MMxe, $M\alpha N\beta$ MMye, $M\alpha N\beta$ MMze	(N), (N), (N), (Nm), (Nm), (Nm)	Dynamic (inertial) component of re- action forces and moments at $M\alpha N\beta$ along local member co- ordinate system

Appendix E. Compiling Stand-Alone SubDyn

See the FAST documentation for instructions on how to compile SubDyn coupled to FAST. Future versions of the manual will include compiling instructions for building the stand-alone SubDyn program.

Appendix F. Major Changes in SubDyn

When first released, SubDyn (v0.4) was included as an undocumented feature of FAST v8 and packaged as a stand-alone archive. Since v0.4, SubDyn has been well integrated into FAST v8 and OpenFast, and the stand-alone form is also available. This appendix outlines significant modifications to SubDyn made since v0.4. Following are the main changes that the user may notice, but for more information, refer to the *changelog.txt* text file within the official archive and the GitHub log.

V1.05.00 (October 2021)

- Version 1.05.00 integrates with OpenFAST version 3.1+
- SubDyn driver supports loads at given nodes
- Outputs of Craig-Bampton/Guyan and FEM Modes to JSON format for visualization
- Streamlined yaml file output, with rigid body mass matrix at different points
- Bug fix for rigid assemblies.
- Bug fix for mass reported in summary file

V1.04.00 (September 2020)

- Version 1.04.00 integrates with OpenFAST version 2.4
- Member types: beam, rigid link, pretension cable
- Joint types: cantilever, universal, pin, ball
- Input of all terms for concentrated mass
- Guyan damping matrix
- Extra lever arm
- Coupling with SoilDyn
- Inclusion of soil-structure interaction (SSI) via flexible degrees of fixity at the restrained nodes and a new input file that allows for 6x6 stiffness and mass matrices that simulate boundary conditions at those nodes.
- Controllable pretension cable elements

V1.03.00a-rrd (September 2017)

- Version 1.03.00a-rrd integrates with the [OpenFast software](#).

V1.01.01a-rrd (September 2014)

Version 1.01.01a-rrd integrates with the [FAST v8 software](#) v8.09.00a-bjj.

- Finite-element eigenvalue bug fixes: the full system eigenvalues were incorrectly reported in the summary file, although with no further consequences on the results. This bug is now fixed.
- Shear area correction factor improvement: the shear area correction factor in the Timoshenko treatment is now aligned with Steinboeck et al. (2013).
- The formulation for the TP reaction has been rearranged to adhere to the theory manual, with no consequences on the output results.

V1.01.00a-rrd (June 2014)

Version 1.00.01a-rrd integrates with the [FAST v8 software](#) v8.08.00c-bjj.

The new implementation has well-defined data exchange interfaces ([following the FAST modularization framework](#)) that should make integration of SubDyn into other multiphysics software packages much simpler.

Several improvements and bug fixes have been implemented since version v0.4 and the module has undergone an extensive verification effort with good results.

- Eigensolver bug fixes: the LAPACK solver proved to be unstable in single precision, and now it is solely run in double precision, regardless of the precision used in other parts of the code.
- The input file format has changed. Please refer to the sample input file in Appendix A and the following notes:
 - First header line has been removed.
 - Simulation Control Section:
 - * **SDeltaT**: The “DEFAULT” keyword (in place of 0.0) is now used to indicate that the glue-code time step will be used for time integration by SubDyn.
 - * **IntMethod**: Allowed values are now 1-4 (in place of 0-3).
 - * **SttcSolve**: New flag introduced. If set to TRUE, the static-improvement method (SIM) will be used.
 - FEA and Craig-Bampton Parameters Section:
 - * In v0.4, the damping coefficients had to be specified for all retained Craig-Bampton modes, or just once for all the modes (if **CBMod** = FALSE). In this version, the user can input any number of damping coefficients. In case the number of retained C-B modes (**NModes**) is larger than the input number of damping coefficients (**JDampings**), the last damping value will be replicated for all the remaining modes.
 - Base Reaction Joints, Interface Joints, Member, and Member Cosine Matrices Sections:
 - * One line with units, below the headers, is expected in all the tables of the input file.
 - Output: Summary and Outfile Section:
 - * This section now also contains the parameters previously assigned under the Section titled “Output: Fast/Subdyn Output-File Variables”

- Some of the quantities in the summary file have been fixed. Some of the output matrices were, in fact, being output with wrong values because of an index mismatch. The new summary file is shorter and does not contain some of the CB method matrices, unless the compiler directive, `DEBUG`, is set.
- **SIM.** This new implementation helps minimize the number of needed modes to capture the contribution of certain loads (such as static gravity and buoyancy loads or high-frequency loads transferred from the turbine). In the previous version, a large number of internal modes were needed to engage substructural modes excited by static and high-frequency forces. These modes are no longer needed and fewer modes can be retained while still achieving accurate results (see also [Section 4.2.5](#)). With SIM enabled, all modes that are not considered by the Craig-Bampton reduction are treated quasi-statically.
- There is now the possibility of retaining no internal C-B modes, thus relying solely only on SIM, in those cases where the substructure's first eigenfrequencies are much higher than the expected energy-containing modes of the entire system.
- The coupling of SubDyn within FAST now includes full hydro-elastic coupling with the HydroDyn hydrodynamics module.

4.2.6 ExtPtfm

Usage

The *ExtPtfm* module uses superelement properties of the support structure provided by the user (e.g., mass, stiffness, damping, and time series of excitation forces) to compute the reaction of the support-structure at the interface. The module uses a linear formulation and internally keeps track of Guyan and Craig-Bampton degrees of freedom.

Typical sequentially coupled workflow with *ExtPtfm*

The overall workflow includes the following steps:

- The substructure designer performs a time-domain simulation of the isolated substructure under a given sea state, using an external tool such as a finite-element tool that has the capability to generate a superelement model. The underlying model behind the external tool and the time series of loads are reduced using the CB technique described in [Theory](#), where the leader DOF are selected as the ones at the substructure interface node. Results from the reduction are written to a file containing the reduced system matrices, \mathbf{M}_r , \mathbf{C}_r , \mathbf{K}_r , and the time series of reduced loads, \mathbf{f}_r .
- The file is imported in *OpenFAST* by the *ExtPtfm* module, and a time-domain simulation of the full wind turbine is run with the reduced representation of the substructure. At every time step, the *ExtPtfm* module receives as inputs the displacement, velocity, and acceleration of the interface point and returns the loads at this point.
- *OpenFAST* exports times series of loads and displacements at the interface, which are then returned to the substructure designer. These inputs are used as boundary conditions to the external tool and then another time-domain simulation of the substructure is run. Stress concentrations are computed, and code checks are performed.

Using the module

The ExtPtfm module is activated by setting the flag CompSub to 2 in the main OpenFAST input file. The variable SubFile in this same file needs to be set to a valid input file for the module (see [Input Files](#) for the input file specifications).

```
--- [...]
  2 CompSub - Compute sub-structural dynamics (switch) {0=None; 1=SubDyn; 2=ExtPtfm}
--- [...]
"Turbine_ExtPtfm.dat" SubFile - Sub-structure input file (SubDyn or ExtPtfm)
```

Recommendations

Time step superelements may contain high frequencies. As a rule of thumb, it is recommended to use a time step satisfying the following criteria: $\Delta t = \frac{1}{10 f_{\max}}$, where f_{\max} is the maximum frequency present in the superelement or full OpenFAST model.

Number of Craig-Bampton modes It is recommended to perform a sensitivity analyses of the results for an increasing number of Craig-Bampton modes to get an idea of how many modes are needed to reach convergence and see which modes contributes the most to the system response.

Input Files

The different input files used by ExtPtfm are described in this section. ExtPtfm uses the SI system (kg, m, s, N) No lines should be added or removed from the input files.

ExtPtfm Module Input File

Prior to OpenFAST 2.3, the ExtPtfm module had no “module” input file, and the Guyan ASCII input file was given directly. This is no longer supported, and a module input file is required. An example of [OpenFAST setup with ExtPtfm](#) is available [here](#). An example of [ExtPtfm module input file](#) is available [here](#). The format is similar to other OpenFAST module. The input parameters are:

- **DT:** Time step for numerical integration (s). The user may specify a time step here or use “default” to rely on the glue-code time-step.
- **IntMethod:** numerical method for the time integration. The Runge-Kutta, Adams–Bashforth and Adams–Bashforth-Moulton methods are available.
- **FileFormat:** file format used for the reduction inputs. Available formats are GuyanASCII (see [Guyan input file \(GuyanASCII\)](#)) and FlexASCII (see [Superelement input file \(FlexASCII\)](#))
- **Red_Filename:** path of file containing the Guyan/Craig-Bampton inputs.
- **RedCst_Filename:** path of file containing the Guyan/Craig-Bampton inputs that are constant. This input is not used yet but may be introduced in the future to accommodate for reduction file formats that use two files: one that contains the constants that are structure dependent (static loads from e.g. gravity and matrices), and one that contain the time-varying signals that are simulation dependent (e.g. loads (on top of the constants loads) and wave elevation)
- **ActiveDOFList:** list of size NActiveDOFList containing the CB modes indices that are active. This list is not read if NActiveDOFList≤0. When specified, all system matrices are reshaped as $M_{\text{new}} = M(I, I)$ where I is the list of indices, potentially unsorted and noncontiguous. Setting NActiveDOFList=0 is equivalent to a Guyan reduction. Setting NActiveCBDOF = −1 uses all the CB DOF provided in the input file.

- `InitPosList`: list of size `NInitPosList` containing the initial positions for the CB modes. This list is not read if `NInitPosList` ≤ 0 , in which case all the CB DOF positions are initialized to 0.
- `InitVelList`: list of size `NInitVelList` containing the initial velocities for the CB modes. This list is not read if `NInitVelList` ≤ 0 , in which case all the CB DOF velocities are initialized to 0.
- `SumPrint`: Print summary data to `<RootName>.sum`
- `OutFile`, `TabDelim`, `OutFmt`, `TStart`: Output flags, currently unused
- `OutList`: Specifies the list of outputs that the user requests. These outputs are described in [Output channels](#).

Output channels

Outputs are written to disk via the '.out' or '.outb' files exported by *OpenFAST*. The time series of loads and displacements at the interface can be selected in *ElastoDyn* (e.g. `PtFmPitch`) Additional “write outputs” are implemented in *ExtPtfm*, according to the list given below. The symbols used in the theory section ([Theory](#)) are also given in the table.

Table 4.8: Output channels for the *ExtPtfm* module

Channel name	Description	Symbol	Units
IntrfFx	Platform interface force - Directed along the x-direction	$f_C[1]$	(N)
IntrfFy	Platform interface force - Directed along the y-direction	$f_C[2]$	(N)
IntrfFz	Platform interface force - Directed along the z-direction	$f_C[3]$	(N)
IntrfMx	Platform interface moment - Directed along the x-direction	$f_C[4]$	(Nm)
IntrfMy	Platform interface moment - Directed along the y-direction	$f_C[5]$	(Nm)
IntrfMz	Platform interface moment - Directed along the z-direction	$f_C[6]$	(Nm)
InpF_Fx	Reduced input force at interface point - Directed along the x-direction	$f_{r1}[1]$	(N)
InpF_Fy	Reduced input force at interface point - Directed along the y-direction	$f_{r1}[2]$	(N)
InpF_Fz	Reduced input force at interface point - Directed along the z-direction	$f_{r1}[3]$	(N)
InpF_Mx	Reduced input moment at interface point - Directed along the x-direction	$f_{r1}[4]$	(Nm)
InpF_My	Reduced input moment at interface point - Directed along the y-direction	$f_{r1}[5]$	(Nm)
InpF_Mz	Reduced input moment at interface point - Directed along the z-direction	$f_{r1}[6]$	(Nm)
CBQ_XXX	Displacement of CB DOF number XXX (e.g. CBQ_001)	$x_2[XXX]$	(-)
CBQD_XXX	Velocity of CB DOF number XXX (e.g. CBQD_001)	$\dot{x}_2[XXX]$	(-)
CBQD2_XXX	Acceleration of CB DOF number XXX	$\ddot{x}_2[XXX]$	(-)
CBF_XXX	Reduced input modal force in CB DOF number XXX	$f_{r2}[XXX]$	(-)
WaveElevExt	Wave elevation provided in the external file	η	(m)

Guyan input file (GuyanASCII)

The Guyan input files format is a legacy file format used for superelements that only contains 6 interface degrees of freedom.

Example

An example of ASCII file that was used for Guyan-Reduced sub-structure is given below, where numerical values are implied instead of M11, t1, Fx1 etc.

```
Comment
#Mass
  M11 ... M16
    [...]
  M61 ... M66
#Damping
  [6 x 6 matrix]
#Stiffness
  [6 x 6 matrix]
# time-varying force
# Time  Fx  Fy  Fz  Mx  My  Mz
# s      (N) (N) (N) (N-m) (N-m) (N-m)
t1  Fx1 Fy1 Fz1 Mx1 My1 Mz1
    [...]
tN  FxN FyN FzN MxN MyN MzN
```

Specifications

The format is a fixed form format where the line number are assumed. The format specifications are defined below:

- ASCII file
- Line 1 is an arbitrary comment
- Line 2 must contain ‘#mass’ (case insensitive). If not, the file format is invalidated. This requirement is important to differentiate between this format and other ASCII formats.
- Lines: 9 and 16 are comment lines that are ignored
- Lines 3-8, 10-15, 17-22 contain six float values, forming the elements of the mass, damping and stiffness matrices respectively. These values corresponds to \mathbf{M}_r , \mathbf{K}_r and \mathbf{D}_r .
- Lines 23-25 are comment lines and are ignored
- The remaining lines of the files contain 7 float values, corresponding to the values: t , $[F_x(t), F_y(t), F_z(t), M_x(t), M_y(t), M_z(t)] = \mathbf{f}_{r1}(t)$. The number of time steps is here noted N but it is not specified in the file.

In particular the same file format should be used for Guyan and Craig-Bampton reduced substructures. The following sections define formats that can serve for both purposes.

Superelement input file (FlexASCII)

This superelement input file is used to provide the system matrices and time series of loads for superelements with an arbitrary number of Craig-Bampton modes.

Example

An example of superelement file is available [here](#). A “dummy” example is given below, where numerical values are implied for: n , dt , t , $M11$, $F1$ etc.

```
!Comment
!Comment Flex 5 Format
!Dimension:                n
!Time increment in simulation:  dt
!Total simulation time in file:  T
!Mass Matrix (Units (kg,m))
!Dimension:                n
  M11 ... M1n
  [...]
  Mn1 ... Mnn
!Stiffness Matrix (Units (N,m))
!Dimension:                n
  [n x n matrix]
!Damping Matrix (Units (N,m,kg))
!Dimension:                n
  [n x n matrix]
!Loading and Wave Elevation (Units (N,m))
!Dimension: 1 time column -  n force columns - 1 wave elevation column
  t1  F11 ... F1n eta1
      [...]
  tN  F1N ... FnN etaN
```

Specifications

The file follows the following specifications:

- ASCII file
- Line 1: arbitrary comment that needs to start with an exclamation mark ‘!’
- Line 2: comment which must contain the string ‘Flex 5 format’ (case insensitive)
- The following lines are header lines that should start with an exclamation mark.
- The header lines are either comments or lines containing keyword/value pairs
- The following (case insensitive) keywords are currently supported for the header:
 - ‘!dimension:’ followed by the integer $n = 6 + n_{CB}$
 - ‘!time increment in simulation:’ followed by the time step dt
 - ‘!total simulation time in file:’ followed by the simulation length T
- The remaining lines consists of the following special (case insensitive) keywords:

- ‘!mass matrix’: followed by some text. The next line provide a dimension, but it is ignored. The dimension line is then followed by n lines each containing n float values. These values corresponds to M_r .
 - ‘!stiffness matrix’: similar to the mass matrix, the values corresponds to K_r .
 - ‘!damping matrix’: similar to the mass matrix, the values corresponds to D_r .
 - ‘!loading’: followed by some text. The next line contains the dimensions but is ignored. The remaining lines of the file after this keyword should each contain $n+2$ values, corresponding to the time t , the loads $f_r(t)$ and the wave elevation $\eta(t)$. NOTE: the wave elevation is intended for outputs only, but it is not outputted yet. The number of lines N should be consistent with the definition of dt and T from the header. The inputs are linearly interpolated if the time step is different from the time step of *ExtPtfm*.
- For now, the units information and the dimension information after the keywords are ignored. Only the dimension provided in the header is read and should be respected throughout the file. The reason for discarding these information is that at the time of writing, there is no guarantee that this information is always provided, and the format specifications of the units and dimension were not specified.

Theory

This section focuses on the theory behind the ExtPtfm module. The theory was published in the following article [ep-BSA+20] ([access article here](#)) which may be used to reference this documentation.

ExtPtfm relies on a dynamics system reduction via the Craig-Bampton (C-B) method [ep-CB68].

Reduction of the equations of motion

The dynamics of a structure are defined by $M\ddot{x} + C\dot{x} + Kx = f$, where M , C , K are the mass, damping, and stiffness matrices; x is the vector of DOF; and f is the vector of loads acting on the DOF. This system of equations is typically set up for the support structure by a commercial software. The typical number of DOF for a jacket substructure is about 10^3 to 10^4 . The DOF are first partitioned and rearranged into a set of leader and follower DOF, labelled with the subscript l and f , respectively. In the case of the substructure, the six degrees of freedom corresponding to the three translations and rotations of the interface point between the substructure and the tower are selected as leader DOF. Assuming symmetry of the system matrices, the rearranged equation of motions are:

$$\begin{bmatrix} M_{\ell\ell} & M_{\ell f} \\ M_{\ell f}^t & M_{ff} \end{bmatrix} \begin{bmatrix} \ddot{x}_\ell \\ \ddot{x}_f \end{bmatrix} + \begin{bmatrix} C_{\ell\ell} & C_{\ell f} \\ C_{\ell f}^t & C_{ff} \end{bmatrix} \begin{bmatrix} \dot{x}_\ell \\ \dot{x}_f \end{bmatrix} + \begin{bmatrix} K_{\ell\ell} & K_{\ell f} \\ K_{\ell f}^t & K_{ff} \end{bmatrix} \begin{bmatrix} x_\ell \\ x_f \end{bmatrix} = \begin{bmatrix} f_\ell \\ f_f \end{bmatrix} \quad (4.172)$$

The CB reduction assumes that the followers’ motion consists of two parts: (1) the elastic motion that would occur in response to the motion of the leader DOF if the inertia of the followers and the external forces were neglected; and (2) the internal motion that would result from the external forces directly exciting the internal DOF. The first part is effectively obtained from Eq. (4.172) by assuming statics and solving for x_f , leading to:

$$x_{f,\text{Guyan}} = -K_{ff}^{-1} K_{\ell f}^t x_{\ell,\text{Guyan}} = \Phi_1 x_{\ell,\text{Guyan}}, \quad \text{where} \quad \Phi_1 = -K_{ff}^{-1} K_{\ell f}^t \quad (4.173)$$

Eq. (4.173) provides the motion of the followers as a function of the leaders’ motion under the assumptions of the Guyan reduction [ep-Guy65].

The CB method further considers the isolated and undamped eigenvalue problem of the follower DOF: $(K_{ff} - \nu_i^2 M_{ff}) \phi_i = 0$ where ν_i and ϕ_i are the i^{th} angular frequency and mode shape, respectively; this problem is “constrained” because it inherently assumes that the leader DOF are fixed (i.e., zero). The method next selects n_{CB} mode shapes, gathering them as column vectors into a matrix noted Φ_2 . These mode shapes can be selected as the ones with the lowest frequency or a mix of low- and high-frequency mode shapes. Typically, n_{CB} is several orders of magnitude smaller than the original number of DOF, going from $\sim 10^3$ DOF to ~ 20 modes for a wind turbine substructure. The scaling of the modes is chosen such that $\Phi_2^t M_{ff} \Phi_2 = I$, where I is the identity matrix. Effectively, the CB method performs a change of coordinates from the full set, $x = [x_\ell \ x_f]^t$, to the reduced set, $x_r = [x_{r1} \ x_{r2}]^t$,

where x_{r1} corresponds directly to the leader DOF, whereas x_{r2} are the modal coordinates defining the amplitudes of each of the mode shapes selected. The change of variable is formally written as:

$$\begin{bmatrix} x_l \\ x_f \end{bmatrix} \approx \begin{bmatrix} I & 0 \\ \Phi_1 & \Phi_2 \end{bmatrix} \begin{bmatrix} x_{r1} \\ x_{r2} \end{bmatrix} \Leftrightarrow x \approx T x_r, \quad \text{with} \quad T = \begin{bmatrix} I & 0 \\ \Phi_1 & \Phi_2 \end{bmatrix} \quad (4.174)$$

The equations of motion are rewritten in these coordinates by the transformation: $M_r = T^t M T$, $K_r = T^t K T$, $f_r = T^t f$, leading to $M_r \ddot{x}_r + K_r x_r = f_r$, which is written in a developed form as:

$$\begin{bmatrix} M_{r11} & M_{r12} \\ M_{r12}^t & M_{r22} \end{bmatrix} \begin{bmatrix} \ddot{x}_{r1} \\ \ddot{x}_{r2} \end{bmatrix} + \begin{bmatrix} K_{r11} & 0 \\ 0 & K_{r22} \end{bmatrix} \begin{bmatrix} x_{r1} \\ x_{r2} \end{bmatrix} = \begin{bmatrix} f_{r1} \\ f_{r2} \end{bmatrix} \quad (4.175)$$

with

$$\begin{aligned} M_{r11} &= M_{\ell\ell} + \Phi_1^t M_{f\ell} + M_{\ell f} \Phi_1 + \Phi_1^t M_{ff} \Phi_1, & M_{r22} &= \Phi_2^t M_{ff} \Phi_2 = I \\ M_{r12} &= (M_{\ell f} + \Phi_1^t M_{ff}) \Phi_2, & f_{r2} &= \Phi_2^t f_f, & f_{r1} &= f_\ell + \Phi_1^t f_f \\ K_{r11} &= K_{\ell\ell} + K_{\ell f} \Phi_1, & K_{r22} &= \Phi_2^t K_{ff} \Phi_2 \end{aligned}$$

The expressions for the reduced damping matrix, $C_r = T^t C T$, are similar to the ones from the mass matrix, except that C_{r22} is not equal to the identity matrix. Some tools or practitioners may not compute the reduced damping matrix and instead set it based on the Rayleigh damping assumption, using the reduced mass and stiffness matrix. Setting $\Phi_2 \equiv 0$ in Eq. (4.174), or equivalently $n_{CB} \equiv 0$, leads to the Guyan reduction equations.

Coupling with another structure

This section illustrates how the equations of motions are set when a superelement is coupled to another structure. The modular approach presented below is the one implemented in OpenFAST. The superelement is here assumed to represent the substructure (and foundation), but it may be applied to other parts of the wind turbine, in particular the entire support structure. For simplicity, it is assumed here that all the substructure leader DOF have an interface with the remaining part of the structure. The interface DOF are labelled as index 1, the substructure internal DOF as index 2, and the remaining DOF are labelled 0. The subscript r used in the previous paragraph is dropped for the DOF but kept for the matrices. With this labelling, system 0–1 consists of the tower and rotor nacelle assembly, the system 1–2 is the substructure, and the vector, x_1 , is the six degrees of freedom at the top of the transition piece. The damping terms are omitted to simplify the equations, but their inclusion is straightforward. Two ways to set up the equations of motions are presented next, the monolithic or modular approaches (see e.g. [ep-BSA+20]).

Monolithic approach:

In this approach, the full system of equations is solved with all the DOF gathered into one state vector. The system of equations is obtained by assembling the individual mass and stiffness matrices of the different subsystems. Using Eq. (4.174), the equations of motion of the system written in a monolithic form are:

$$\begin{bmatrix} M_{00} & M_{01} & 0 \\ & M_{11} + M_{r11} & M_{r12} \\ \text{sym} & & M_{r22} \end{bmatrix} \begin{bmatrix} \ddot{x}_0 \\ \ddot{x}_1 \\ \ddot{x}_2 \end{bmatrix} + \begin{bmatrix} K_{00} & K_{01} & 0 \\ & K_{11} + K_{r11} & 0 \\ \text{sym} & & K_{r22} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 + f_{r1} \\ f_{r2} \end{bmatrix} \quad (4.176)$$

Modular approach: In this approach, the equations of motion are written for each subsystem. Couplings with other subsystems are introduced using external loads and constraints (which are unnecessary here). The coupling load vector at 1 between the two systems, usually consisting of three forces and three moments, is written as f_C . The equations of motion for system 0–1 are:

$$\begin{bmatrix} M_{00} & M_{01} \\ \text{sym} & M_{11} \end{bmatrix} \begin{bmatrix} \ddot{x}_0 \\ \ddot{x}_1 \end{bmatrix} + \begin{bmatrix} K_{00} & K_{01} \\ \text{sym} & K_{11} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \end{bmatrix} + \begin{bmatrix} 0 \\ f_C \end{bmatrix} \quad (4.177)$$

System 1 – 2 receives the opposite, f_C , from system 0 – 1, leading to the following set of equations for system 1–2:

$$\begin{bmatrix} M_{r11} & M_{r12} \\ \text{sym} & M_{r22} \end{bmatrix} \begin{bmatrix} \ddot{x}_1 \\ \ddot{x}_2 \end{bmatrix} + \begin{bmatrix} K_{r11} & 0 \\ \text{sym} & K_{r22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} f_{r1} \\ f_{r2} \end{bmatrix} - \begin{bmatrix} f_C \\ 0 \end{bmatrix} \quad (4.178)$$

State-space representation of the module *ExtPtfm*

The following sections detail the implementation of the CB approach into *ExtPtfm* to model fixed-bottom substructures.

ExtPtfm provides the coupling load at the interface, \mathbf{f}_C , given the motions of the interface node: \mathbf{x}_1 , $\dot{\mathbf{x}}_1$, $\ddot{\mathbf{x}}_1$. The six degrees of freedom, \mathbf{x}_1 (surge, sway, heave, roll, pitch, and yaw) and the coordinate system used at the interface are given in Fig. 4.41.

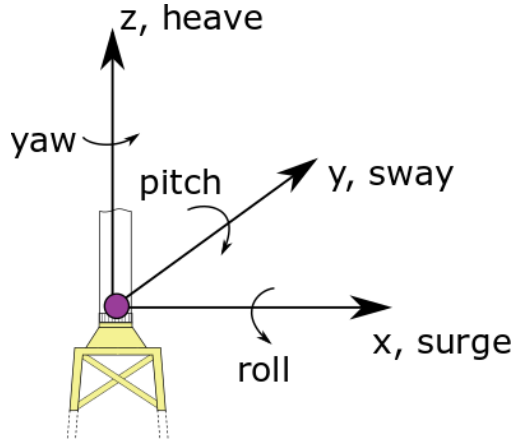


Fig. 4.41: Interface degrees of freedom

ExtPtfm is written in a form that consists of state and output equations. For a linear system, these equations take the following form:

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{X}(\mathbf{x}, \mathbf{u}, t) = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{f}_x \\ \mathbf{y} &= \mathbf{Y}(\mathbf{x}, \mathbf{u}, t) = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} + \mathbf{f}_y\end{aligned}\quad (4.179)$$

where \mathbf{x} is the state vector, \mathbf{u} the input vector, and \mathbf{y} the output vector of the module. The input vector of the module is the motion of the interface node, $\mathbf{u} = [\mathbf{x}_1, \dot{\mathbf{x}}_1, \ddot{\mathbf{x}}_1]^t$, whereas the output vector is the coupling load at the interface node, $\mathbf{y} = [\mathbf{f}_C]^t$. The state vector consists of the motions and velocities of the CB modes, $\mathbf{x} = [\mathbf{x}_2, \dot{\mathbf{x}}_2]^t$. The dimensions of each vector are: $\mathbf{x}(2n_{CB} \times 1)$, $\mathbf{u}(18 \times 1)$, $\mathbf{y}(6 \times 1)$.

Eq. (4.178) is rewritten in the state-space form of Eq. (4.179) as follows. The second block row of (4.178) is developed to isolate $\ddot{\mathbf{x}}_2$. Using $\mathbf{M}_{r22} = \mathbf{I}$ and reintroducing the damping matrix for completeness gives:

$$\ddot{\mathbf{x}}_2 = \mathbf{f}_{r2} - \mathbf{M}_{r12}^t \ddot{\mathbf{x}}_1 - \mathbf{K}_{r22} \mathbf{x}_2 - \mathbf{C}_{r12}^t \dot{\mathbf{x}}_1 - \mathbf{C}_{r22} \dot{\mathbf{x}}_2 \quad (4.180)$$

The matrices of the state-space relation from Eq. (4.179) are then directly identified as ([ep-BSA+20]):

$$\mathbf{A} = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ -\mathbf{K}_{r22} & -\mathbf{C}_{r22} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -\mathbf{C}_{r12}^t & -\mathbf{M}_{r12}^t \end{bmatrix}, \quad \mathbf{f}_x = \begin{bmatrix} \mathbf{0} \\ \mathbf{f}_{r2} \end{bmatrix}$$

Isolating \mathbf{f}_C from the first block row of Eq. (4.178) and using the expression of $\ddot{\mathbf{x}}_2$ from Eq. (4.180) leads to:

$$\begin{aligned}\mathbf{f}_C &= \mathbf{f}_{r1} - \mathbf{M}_{r11} \ddot{\mathbf{x}}_1 - \mathbf{C}_{r11} \dot{\mathbf{x}}_1 - \mathbf{C}_{r12} \dot{\mathbf{x}}_2 - \mathbf{K}_{r11} \mathbf{x}_1 \\ &\quad - \mathbf{M}_{r12} (\mathbf{f}_{r2} - \mathbf{M}_{r12}^t \ddot{\mathbf{x}}_1 - \mathbf{C}_{r12}^t \dot{\mathbf{x}}_1 - \mathbf{C}_{r22} \dot{\mathbf{x}}_2 - \mathbf{K}_{r22} \mathbf{x}_2)\end{aligned}$$

The matrices of for the output \mathbf{y} are then identified as ([ep-BSA+20]):

$$\begin{aligned}\mathbf{C} &= [\mathbf{M}_{r12} \mathbf{K}_{r22} \quad \mathbf{M}_{r12} \mathbf{C}_{r22} - \mathbf{C}_{r12}], & \mathbf{f}_y &= [\mathbf{f}_{r1} - \mathbf{M}_{r12} \mathbf{f}_{r2}] \\ \mathbf{D} &= [-\mathbf{K}_{r11} \quad -\mathbf{C}_{r11} + \mathbf{M}_{r12} \mathbf{C}_{r12}^t \quad -\mathbf{M}_{r11} + \mathbf{M}_{r12} \mathbf{M}_{r12}^t]\end{aligned}$$

All the block matrices and vectors labeled with “r” are provided to the module via an input file. At a given time step, the loads, $\mathbf{f}_r(t)$, are computed by linear interpolation of the loads given in the input file, and the state equation, , is solved for \mathbf{x} with the outputs returned to the glue code of *OpenFAST*.

The glue code can also perform the linearization of the full system at a given time or operating point, using the Jacobians of the state equations of each module. Since the formulation of *ExtPtfm* is linear, the Jacobian of the state and output equations, with respect to the states and inputs of the module, are:

$$\frac{\partial \mathbf{X}}{\partial \mathbf{x}} = \mathbf{A}, \quad \frac{\partial \mathbf{Y}}{\partial \mathbf{x}} = \mathbf{C}, \quad \frac{\partial \mathbf{X}}{\partial \mathbf{u}} = \mathbf{B}, \quad \frac{\partial \mathbf{Y}}{\partial \mathbf{u}} = \mathbf{D}$$

The linearization of *ExtPtfm* was implemented in the module, but some work remains to be done at the glue-code (*OpenFAST*) level to allow for full system linearization.

Example Files

Examples of inputs files are given below:

- [ExtPtfm Module input file](#)
- [Superelement input file](#)

4.2.7 ElastoDyn Users Guide and Theory Manual

This document offers a quick reference guide for the *ElastoDyn* software program. It is intended to be used by the general user in combination with other *OpenFAST* manuals. The manual will be updated as new releases are issued and as needed to provide further information on advancements or modifications to the software.

Much more documentation for this module is available in the legacy documentation listed in [General](#). Specifically, the FAST v6 User’s Guide as well as small updates in the FAST v8 README describe this module and contain general information on *OpenFAST*.

Note: We are in the process of migrating the documentation from FAST 8 to *OpenFAST*. For reference, various portions of old documentation are provided here. While most of it is still directly applicable to *OpenFAST*, portions may be out of date.

The following documents are a detailed derivation of the equations of motion of *ElastoDyn*. These documents have not been compiled into a report, so they contain mostly equations and little explanatory text. A reader with a background in kinematics and dynamics may be able to comprehend the equations. The documents make the most sense if studied in the following order:

1. *FASTDOFs.xls*: Contains a listing of the DOF indices used by the equations of motion in FAST.
2. *FASTCoordinateSystems.doc*: Documents the transformation matrices relating each coordinate system in FAST. Unfortunately, there are no pictures in this document that diagram these coordinate systems. They can hopefully be visualized by means of the transformation matrices.
3. *FASTKinematics.doc*: Documents the linear position, velocity, and acceleration vectors of each “important” point in the system and documents the angular velocity and acceleration vectors of each “important” reference frame in the system. Also included is documentation of the partial velocity vectors needed by Kane’s dynamics.
4. *FASTKinetics.doc*: Documents the derivation of the equations of motion using Kane’s dynamics.
5. *FASTLoads.doc*: Documents how the output loads are computed using terms from the equations of motion.
6. *FASTMotions.doc*: Documents how the output motions are computed using variables from the equations of motion.

7. `FASTLogicFlow.doc`: Contains a listing of the subroutine names used by FAST. The names are listed in the order they are called within the program.

There are a few minor errors in the equations documented in these papers that may be clear after understanding the equations. The implemented code does not have these errors. The papers do not describe the Fortran source code and variable naming conventions, but a source code comparison is possible with careful study.

Note that the “unofficial FAST Theory Manual” applies to the structural equations of FAST v7 and the ElastoDyn module of FAST v8 and OpenFAST.

Input Files

The user configures the structural model parameters via a primary ElastoDyn input file, as well as separate input files for the tower and *other stuff that will be documented here later*.

No lines should be added or removed from the input files.

Units

ElastoDyn uses the SI system (kg, m, s, N). Angles are assumed to be in radians unless otherwise specified.

ElastoDyn Primary Input File

The primary ElastoDyn input file defines modeling options and geometries for the OpenFAST structure including the tower, nacelle, drivetrain, and blades (if BeamDyn is not used). It also sets the initial conditions for the structure.

Simulation Control

Set the **Echo** flag to TRUE if you wish to have ElastoDyn echo the contents of the ElastoDyn primary, airfoil, and blade input files (useful for debugging errors in the input files). The echo file has the naming convention of *OutRootFile.ED.ech*. **OutRootFile** is either specified in the I/O SETTINGS section of the driver input file when running ElastoDyn standalone, or by the OpenFAST program when running a coupled simulation.

Method

dT

Degrees of Freedom

FlapDOF1 - First flapwise blade mode DOF (flag)

FlapDOF2 - Second flapwise blade mode DOF (flag)

EdgeDOF - First edgewise blade mode DOF (flag)

TeetDOF - Rotor-teeter DOF (flag) [unused for 3 blades]

DrTrDOF - Drivetrain rotational-flexibility DOF (flag)

GenDOF - Generator DOF (flag)

YawDOF - Yaw DOF (flag)

TwFADOF1 - First fore-aft tower bending-mode DOF (flag)

TwFADOF2 - Second fore-aft tower bending-mode DOF (flag)

TwSSDOF1 - First side-to-side tower bending-mode DOF (flag)

TwSSDOF2 - Second side-to-side tower bending-mode DOF (flag)

PtfmSgDOF - Platform horizontal surge translation DOF (flag)

PtfmSwDOF - Platform horizontal sway translation DOF (flag)

PtfmHvDOF - Platform vertical heave translation DOF (flag)

PtfmRDOF - Platform roll tilt rotation DOF (flag)

PtfmPDOF - Platform pitch tilt rotation DOF (flag)

PtfmYDOF - Platform yaw rotation DOF (flag)

Initial Conditions

OoPDefl - Initial out-of-plane blade-tip displacement (meters)

IPDefl - Initial in-plane blade-tip deflection (meters)

BIPitch(1) - Blade 1 initial pitch (degrees)

BIPitch(2) - Blade 2 initial pitch (degrees)

BIPitch(3) - Blade 3 initial pitch (degrees) [unused for 2 blades]

TeetDefl - Initial or fixed teeter angle (degrees) [unused for 3 blades]

Azimuth - Initial azimuth angle for blade 1 (degrees)

RotSpeed - Initial or fixed rotor speed (rpm)

NacYaw - Initial or fixed nacelle-yaw angle (degrees)

TTDspFA - Initial fore-aft tower-top displacement (meters)

TTDspSS - Initial side-to-side tower-top displacement (meters)

PtfmSurge - Initial or fixed horizontal surge translational displacement of platform (meters)

PtfmSway - Initial or fixed horizontal sway translational displacement of platform (meters)

PtfmHeave - Initial or fixed vertical heave translational displacement of platform (meters)

PtfmRoll - Initial or fixed roll tilt rotational displacement of platform (degrees)

PtfmPitch - Initial or fixed pitch tilt rotational displacement of platform (degrees)

PtfmYaw - Initial or fixed yaw rotational displacement of platform (degrees)

Turbine Configuration

NumBl - Number of blades (-)

TipRad - The distance from the rotor apex to the blade tip (meters)

HubRad - The distance from the rotor apex to the blade root (meters)

PreCone(1) - Blade 1 cone angle (degrees)

PreCone(2) - Blade 2 cone angle (degrees)

PreCone(3) - Blade 3 cone angle (degrees) [unused for 2 blades]

HubCM - Distance from rotor apex to hub mass [positive downwind] (meters)

UndSling - Undersling length [distance from teeter pin to the rotor apex] (meters) [unused for 3 blades]

Delta3 - Delta-3 angle for teetering rotors (degrees) [unused for 3 blades]

AzimB1Up - Azimuth value to use for I/O when blade 1 points up (degrees)

OverHang - Distance from yaw axis to rotor apex [3 blades] or teeter pin [2 blades] (meters)

ShftGagL - Distance from rotor apex [3 blades] or teeter pin [2 blades] to shaft strain gages [positive for upwind rotors] (meters)

ShftTilt - Rotor shaft tilt angle (degrees)

NacCMxn - Downwind distance from the tower-top to the nacelle CM (meters)

NacCMyn - Lateral distance from the tower-top to the nacelle CM (meters)

NacCMzn - Vertical distance from the tower-top to the nacelle CM (meters)

NcIMUxn - Downwind distance from the tower-top to the nacelle IMU (meters)

NcIMUyn - Lateral distance from the tower-top to the nacelle IMU (meters)

NcIMUzn - Vertical distance from the tower-top to the nacelle IMU (meters)

Twr2Shft - Vertical distance from the tower-top to the rotor shaft (meters)

TowerHt - Height of tower above ground level [onshore] or MSL [offshore] (meters)

TowerBsHt - Height of tower base above ground level [onshore] or MSL [offshore] (meters)

PtfmCMxt - Downwind distance from the ground level [onshore] or MSL [offshore] to the platform CM (meters)

PtfmCMyt - Lateral distance from the ground level [onshore] or MSL [offshore] to the platform CM (meters)

PtfmCMzt - Vertical distance from the ground level [onshore] or MSL [offshore] to the platform CM (meters)

PtfmRefzt - Vertical distance from the ground level [onshore] or MSL [offshore] to the platform reference point (meters)

Mass and Inertia

TipMass(1) - Tip-brake mass, blade 1 (kg)

TipMass(2) - Tip-brake mass, blade 2 (kg)

TipMass(3) - Tip-brake mass, blade 3 (kg) [unused for 2 blades]

HubMass - Hub mass (kg)

HubIner - Hub inertia about rotor axis [3 blades] or teeter axis [2 blades] (kg m^2)

GenIner - Generator inertia about HSS (kg m^2)

NacMass - Nacelle mass (kg)

NacYIner - Nacelle inertia about yaw axis (kg m^2)

YawBrMass - Yaw bearing mass (kg)

PtfmMass - Platform mass (kg)

PtfmRIner - Platform inertia for roll tilt rotation about the platform CM (kg m^2)

PtfmPIner - Platform inertia for pitch tilt rotation about the platform CM (kg m^2)

PtfmYIner - Platform inertia for yaw rotation about the platform CM (kg m²)

Blade

BldNodes - Number of blade nodes (per blade) used for analysis (-)

BldFile(1) - Name of file containing properties for blade 1 (quoted string)

BldFile(2) - Name of file containing properties for blade 2 (quoted string)

BldFile(3) - Name of file containing properties for blade 3 (quoted string) [unused for 2 blades]

Rotor-Teeter

TeetMod - Rotor-teeter spring/damper model {0: none, 1: standard, 2: user-defined from routine UserTeet} (switch) [unused for 3 blades]

TeetDmpP - Rotor-teeter damper position (degrees) [used only for 2 blades and when TeetMod=1]

TeetDmp - Rotor-teeter damping constant (N-m/(rad/s)) [used only for 2 blades and when TeetMod=1]

TeetCDmp - Rotor-teeter rate-independent Coulomb-damping moment (N-m) [used only for 2 blades and when TeetMod=1]

TeetSSStP - Rotor-teeter soft-stop position (degrees) [used only for 2 blades and when TeetMod=1]

TeetHStP - Rotor-teeter hard-stop position (degrees) [used only for 2 blades and when TeetMod=1]

TeetSSSp - Rotor-teeter soft-stop linear-spring constant (N-m/rad) [used only for 2 blades and when TeetMod=1]

TeetHSSp - Rotor-teeter hard-stop linear-spring constant (N-m/rad) [used only for 2 blades and when TeetMod=1]

Drivetrain

GBoxEff - Gearbox efficiency (%)

GBRatio - Gearbox ratio (-)

DTTorSpr - Drivetrain torsional spring (N-m/rad)

DTTorDmp - Drivetrain torsional damper (N-m/(rad/s))

Furling

Furling - Read in additional model properties for furling turbine (flag) [must currently be FALSE]

FurlFile - Name of file containing furling properties (quoted string) [unused when Furling=False]

Tower

TwrNodes - Number of tower nodes used for analysis (-)

TwrFile - Name of file containing tower properties (quoted string)

Outputs

SumPrint [flag] Set this value to TRUE if you want ElastoDyn to generate a summary file with the name **OutFileRoot.ED.sum***. **OutFileRoot** is specified by the OpenFAST program when running a coupled simulation.

OutFile [switch] is currently unused. The eventual purpose is to allow output from ElastoDyn to be written to a module output file (option 1), or the main OpenFAST output file (option 2), or both. At present this switch is ignored.

TabDelim [flag] is currently unused. Setting this to True will set the delimiter for text files to the tab character for the ElastoDyn module **OutFile**.

OutFmt [quoted string] is currently unused. ElastoDyn will use this string as the numerical format specifier for output of floating-point values in its local output specified by **OutFile**. The length of this string must not exceed 20 characters and must be enclosed in apostrophes or double quotes. You may not specify an empty string. To ensure that fixed-width column data align properly with the column titles, you should ensure that the width of the field is 10 characters. Using an E, EN, or ES specifier will guarantee that you will never overflow the field because the number is too big, but such numbers are harder to read. Using an F specifier will give you numbers that are easier to read, but you may overflow the field. Please refer to any Fortran manual for details for format specifiers.

TStart [s] sets the start time for **OutFile**. This is currently unused.

DecFact [-] This parameter sets the decimation factor for output. ElastoDyn will output data to **OutFile** only once each DecFact integration time steps. For instance, a value of 5 will cause FAST to generate output only every fifth time step. This value must be an integer greater than zero.

NTwGages [-] The number of strain-gage locations along the tower indicates the number of input values on the next line. Valid values are integers from 0 to 5 (inclusive).

TwrGagNd [-] The virtual strain-gage locations along the tower are assigned to the tower analysis nodes specified on this line. Possible values are 1 to **TwrNodes** (inclusive), where 1 corresponds to the node closest to the tower base (but not at the base) and a value of **TwrNodes** corresponds to the node closest to the tower top. The exact elevations of each analysis node in the undeflected tower, relative to the base of the tower, are determined as follows:

$$\text{Elev. of node } J = \text{TwrRBHt} + (J - 1/2) \cdot [(\text{TowerHt} + \text{TwrDraft} - \text{TwrRBHt}) / \text{TwrNodes}]$$

(for $J = 1, 2, \dots, \text{TwrNodes}$)

You must enter at least **NTwGages** values on this line. If **NTwGages** is 0, this line will be skipped, but you must have a line taking up space in the input file. You can separate the values with combinations of tabs, spaces, and commas, but you may use only one comma between numbers.

NBlGages [-] specifies the number of strain-gage locations along the blade, and indicates the number of input values expected in **BldGagNd**. This is only used when the blade structure is modeled in ElastoDyn.

BldGagNd [-] specifies the virtual strain-gage locations along the blade that should be output. Possible values are 1 to **BldNodes** (inclusive), where 1 corresponds to the node closest to the blade root (but not at the root) and a value of **BldNodes** corresponds to the node closest to the blade tip. The node locations are specified by the ElastoDyn blade input files. You must enter at least **NBlGages** values on this line. If **NBlGages** is 0, this line will be skipped, but you must have a line taking up space in the input file. You can separate the values with combinations of tabs, spaces, and commas, but you may use only one comma between numbers. This is only used when the blade structure is modeled in ElastoDyn.

The **OutList** section controls output quantities generated by ElastoDyn. Enter one or more lines containing quoted strings that in turn contain one or more output parameter names. Separate output parameter names by any combination

of commas, semicolons, spaces, and/or tabs. If you prefix a parameter name with a minus sign, “-”, underscore, “_”, or the characters “m” or “M”, ElastoDyn will multiply the value for that channel by -1 before writing the data. The parameters are written in the order they are listed in the input file. ElastoDyn allows you to use multiple lines so that you can break your list into meaningful groups and so the lines can be shorter. You may enter comments after the closing quote on any of the lines. Entering a line with the string “END” at the beginning of the line or at the beginning of a quoted string found at the beginning of the line will cause ElastoDyn to quit scanning for more lines of channel names. Blade and tower node-related quantities are generated for the requested nodes identified through the **BldGagNd** and **TwrGagNd** lists above. If ElastoDyn encounters an unknown/invalid channel name, it warns the users but will remove the suspect channel from the output file. Please refer to the ElastoDyn tab in the Excel file `OutListParameters.xlsx` for a complete list of possible output parameters.

Nodal Outputs

In addition to the named outputs in [Section 4.2.7](#) above, ElastoDyn allows for outputting the full set blade node motions and loads (tower nodes unavailable at present). Please refer to the `ElastoDyn_Nodes` tab in the Excel file `OutListParameters.xlsx` for a complete list of possible output parameters.

This section follows the *END* statement from normal Outputs section described above, and includes a separator description line followed by the following options.

BldNd_BladesOut specifies the number of blades to output. Possible values are 0 through the number of blades ElastoDyn is modeling. If the value is set to 1, only blade 1 will be output, and if the value is 2, blades 1 and 2 will be output.

BldNd_BlOutNd specifies which nodes to output. This is currently unused.

The **OutList** section controls the nodal output quantities generated by ElastoDyn. In this section, the user specifies the name of the channel family to output. The output name for each channel is then created internally by ElastoDyn by combining the blade number, node number, and channel family name. For example, if the user specifies **TDx** as the channel family name, the output channels will be named with the convention of **B β N###TDx** where β is the blade number, and **###** is the three digit node number.

Sample Nodal Outputs section

This sample includes the *END* statement from the regular outputs section.

```

1  END of input file (the word "END" must appear in the first 3 columns of this last
   ↳ OutList line)
2  ----- NODE OUTPUTS -----
3      3    BldNd_BladesOut - Blades to output
4      99    BldNd_BlOutNd  - Blade nodes on each blade (currently unused)
5      OutList - The next line(s) contains a list of output parameters. See
   ↳ OutListParameters.xlsx, ElastoDyn_Nodes tab for a listing of available output channels,
   ↳ (-)
6  "ALx"    - local flapwise acceleration (absolute) of node
7  "ALy"    - local flapwise acceleration (absolute) of node
8  "ALz"    - local flapwise acceleration (absolute) of node
9  "TDx"    - local flapwise (translational) deflection (relative to the undeflected
   ↳ position) of node
10 "TDy"    - local edgewise (translational) deflection (relative to the undeflected
   ↳ position) of node
11 "TDz"    - local axial (translational) deflection (relative to the undeflected position)
   ↳ of node

```

(continues on next page)

(continued from previous page)

```

12 "RDx"    - Local rotational displacement about x-axis (relative to undeflected)
13 "RDy"    - Local rotational displacement about y-axis (relative to undeflected)
14 "RDz"    - Local rotational displacement about z-axis (relative to undeflected)
15 "MLx"    - local edgewise moment at node
16 "MLy"    - local flapwise moment at node
17 "MLz"    - local pitching moment at node
18 "FLx"    - local flapwise shear force at node
19 "FLy"    - local edgewise shear force at node
20 "FLz"    - local axial force at node
21 "MLxNT"  - Edgewise moment in local coordinate system (initial structural twist removed)
22 "MlyNT"  - Flapwise shear moment in local coordinate system (initial structural twist
↳ removed)
23 "FLxNT"  - Flapwise shear force in local coordinate system (initial structural twist
↳ removed)
24 "FlyNT"  - Edgewise shear force in local coordinate system (initial structural twist
↳ removed)
25 END of input file (the word "END" must appear in the first 3 columns of this last
↳ OutList line)
26 -----

```

4.2.8 HydroDyn User Guide and Theory Manual

Installation and Getting Started

HydroDyn is included in the OpenFAST software repository and consists of two major components:

- *hydrodyn_driver* is the standalone HydroDyn executable
- *hydrodynlib* is the OpenFAST module library; it is most commonly used when driven through the HydroDyn driver or the OpenFAST glue code

For installation instructions, see *Installing OpenFAST*. In sections where an installation target can be specific, use *hydrodyn_driver*.

Running the HydroDyn Driver

The HydroDyn Driver has a simple command line interface:

```
hydrodyn_driver <input_file>
```

where *input_file* is the file described in *HydroDyn Driver Input File*. Additional input files are required, including the *HydroDyn Primary Input File*. The time-series output as well as other output from HydroDyn are described in *Output Files*.

Running HydroDyn coupled to OpenFAST

To run an OpenFAST simulation with the HydroDyn module enabled, the *CompHydro* flag must be switched on and the *HydroDyn Primary Input File* path supplied in the OpenFAST primary input file:

```
# In the "Feature switches" section
1           CompHydro    - Compute hydrodynamic loads (switch) {0=None; 1=HydroDyn}

# In the "Input files" section
"HydroDyn.dat" HydroFile  - Name of file containing hydrodynamic input parameters.
↪(quoted string)
```

The time-series output as well as other output from HydroDyn are described in *Output Files*.

Input Files

The user configures the hydrodynamic model parameters as well as the substructure geometry and properties via a primary HydroDyn input file. When used in standalone mode, an additional driver input file is required. This driver file specifies initialization inputs normally provided to HydroDyn by OpenFAST, as well as the per-time-step inputs to HydroDyn.

No lines should be added or removed from the input files, except in tables where the number of rows is specified.

Units

HydroDyn uses the SI system (kg, m, s, N).

HydroDyn Driver Input File

The driver input file is only needed for the standalone version of HydroDyn and contains inputs normally generated by OpenFAST, and are necessary to control the hydrodynamic simulation for uncoupled models. A sample HydroDyn driver input file is given in Appendix B.

Set the **Echo** flag in this file to TRUE if you wish to have HydroDynDriver echo the contents of the driver input file (useful for debugging errors in the driver file). The echo file has the naming convention of `OutRootName.dvr.ech`. **OutRootName** is specified in the HYDRODYN section of the driver input file. Set the gravity constant using the **Gravity** parameter. HydroDyn expects a magnitude, so in SI units this would be set to $9.80665 \frac{m}{s^2}$. **WtrDens** specifies the water density and must be a value greater than or equal to zero; a typical value of seawater is around 1025 kg/m^3 . **WtrDpth** specifies the water depth (depth of the flat seabed), based on the reference MSL, and must be a value greater than zero. **MSL2SWL** is the offset between the MSL and SWL, positive upward. This parameter is useful when simulating the effect of tides or storm-surge sea-level variations without having to alter the substructure geometry information. This parameter is unused with **WaveMod** = 6 and must be set to zero if you are using a potential-flow model (**PotMod** = 1 or 2). **WaveMod** and **PotMod** are specified in the HydroDyn primary input file.

HDInputFile is the filename of the primary HydroDyn input file. This name should be in quotations and can contain an absolute path or a relative path. All HydroDyn-generated output files will be prefixed with **OutRootName**. If this parameter includes a file path, the output will be generated in that folder. **NSteps** specifies the number of simulation time steps, and **TimeInterval** specifies the time between steps.

Setting **WAMITInputsMod** = 0 forces all WAMIT reference point (WRP) input motions to zero for all time. If you set **WAMITInputsMod** = 1, then you must set the steady-state inputs in the WAMIT STEADY STATE INPUTS section of the file. Setting **WAMITInputsMod** = 2, requires the time-series input file whose name is specified via the **WAMITInputsFile** parameter. The WAMIT inputs file is a text-formatted file. This file has no header lines. Each

data row corresponds to a given time step, and the whitespace separated columns of floating point values represent the necessary motion inputs as shown in Table 4.9. All motions are specified in the global inertial-frame coordinate system.

Table 4.9: WAMIT Inputs Time-Series Data File Contents

Column Number	Input	Units
1	Time step value	s
2-4	Translational displacements along X , Y , and Z	m
5-7	Rotational displacements about X , Y , and Z (small angle assumptions apply)	radians
8-10	Translational velocities along X , Y , and Z	$\frac{m}{s}$
11-13	Rotational velocities about X , Y , and Z	$\frac{\text{radians}}{s}$
14-16	Translational accelerations along X , Y , and Z	$\frac{m}{s^2}$
17-19	Rotational accelerations about X , Y , and Z	$\frac{\text{radians}}{s^2}$

In a similar fashion, the input motions for the Morison members (strip-theory model) are set to zero if **MorisonInputsMod** = 0. If you select **MorsionInputsMod** = 1 then the motions at each substructure joint are set to the steady-state values given in the MORISON STEADY STATE INPUTS section. Currently, option 2 is unavailable for the Morison inputs.

The standalone HydroDyn does not check for physical consistency between motions specified for the WRP and Morison members in the driver file.

Setting **WaveElevSeriesFlag** to TRUE enables the outputting of a grid of wave elevations to a text-based file with the name `OutRootName.WaveElev.out`. The grid consists of **WaveElevNX** by **WaveElevNY** wave elevations (centered at $X = 0$, $Y = 0$ i.e., (0,0)) with a **dX** and **dY** spacing in the global inertial-frame coordinate system. These wave elevations are distinct and output separately from the wave elevations determined by **NWaveElev** in the HydroDyn primary input file, such that the total number of wave elevation outputs is **NWaveElev** + (**WaveElevNX** × **WaveElevNY**

). The wave-elevation output file `OutRootName.WaveElev.out` contains the total wave elevation, which is the sum of the first- and second-order terms (when the second-order wave kinematics are optionally enabled).

HydroDyn Primary Input File

The HydroDyn input file defines the substructure geometry, hydrodynamic coefficients, incident wave kinematics and current, potential-flow solution options, flooding/ballasting and marine growth, and auxiliary parameters. The geometry of strip-theory members is defined by joint coordinates of the undisplaced substructure in the global reference system, with the origin at the intersection of the undeflected tower centerline with MSL. A member connects two joints; multiple members can use a common joint. The hydrodynamic loads are computed at nodes, which are the resultant of member refinement into multiple (**MDivSize** input) elements (nodes are located at the ends of each element), and they are calculated by the module. Member properties include outer diameter, thickness, and dynamic-pressure, added-mass and viscous-drag coefficients. Member properties are specified at the joints; if properties change from one joint to the other, they will be linearly interpolated for the inner nodes.

The file is organized into several functional sections. Each section corresponds to an aspect of the hydrodynamics model or the submerged substructure. A sample HydroDyn primary input file is given in [Appendix A: OC4 Semi-submersible Input File](#).

If this manual refers to an ID in a table entry, this is an integer identifier for the table entry, and these IDs do not need to be consecutive or increasing, but they must be unique for a given table entry.

The input file begins with two lines of header information which is for your use, but is not used by the software. On the next line, set the **Echo** flag to TRUE if you wish to have HydroDyn echo the contents of the HydroDyn input file (useful for debugging errors in the input file). The echo file has the naming convention of **OutRootName.HD.ech**. **OutRootName** is either specified in the HYDRODYN section of the driver input file when running HydroDyn standalone, or by FAST when running a coupled simulation.

Environmental Conditions

Environmental conditions are now specified in the driver input file but are left in the primary input file for legacy compatibility. Use the keyword DEFAULT to pass in values specified by the driver input file. Otherwise, values given in the primary input file will overwrite those given in the driver input file. **WtrDens** specifies the water density and must be a value greater than or equal to zero; a typical value of seawater is around 1025 kg/m³. **WtrDpth** specifies the water depth (depth of the flat seabed), based on the reference MSL, and must be a value greater than zero. **MSL2SWL** is the offset between the MSL and SWL, positive upward. This parameter is useful when simulating the effect of tides or storm-surge sea-level variations without having to alter the substructure geometry information. This parameter is unused with **WaveMod** = 6 and must be set to zero if you are using a potential-flow model (**PotMod** = 1 or 2).

Waves

The WAVES section of the input file controls the internal generation of first-order waves or the use of externally generated waves, used by both the strip-theory and potential-flow solutions. The wave spectrum settings in this section only pertain to the first-order wave frequency components. When second-order terms are optionally enabled—see the [2nd-Order Waves](#) and [2nd-Order Floating Platform Forces](#) sections below—the second-order terms are calculated using the first-order wave-component amplitudes and extra energy is added to the wave spectrum (at the difference and sum frequencies).

WaveMod specifies the incident wave kinematics model. The options are:

- 0: none = still water
- 1: regular (periodic) waves

- 1P#: regular (periodic) waves with user-specified phase, for example 1P20.0 for regular waves with a 20 phase (without P#, the phase will be random, based on **WaveSeed**); 0 phase represents a cosine function, starting at the peak and decreasing in time
- 2: Irregular (stochastic) waves based on the JONSWAP or Pierson-Moskowitz frequency spectrum
- 3: Irregular (stochastic) waves based on a white-noise frequency spectrum
- 4: Irregular (stochastic) waves based on a user-defined frequency spectrum from routine *UserWaveSpcrm()*; see Appendix D for compiling instructions
- 5: Externally generated wave-elevation time series
- 6: Externally generated full wave-kinematics time series

Option 4 requires that the *UserWaveSpcrm()* subroutine of the *Waves.f90* source file be implemented by the user, and will require recompiling either the standalone HydroDyn program or FAST. Option 5 allows the use of externally generated wave-elevation time series, from which the hydrodynamic loads in the potential-flow solution or the wave kinematics used in the strip-theory solution are derived internally. Option 6 allows the use of full externally generated wave kinematics for use with the strip-theory solution (but not the potential-flow solution). With options 5 and 6, the externally generated wave data is provided through input files, all of which have the root name given by the **WvKinFile** parameter below.

This version does not include the ability to model stretching of internally generated incident wave kinematics to the instantaneous free surface; you must set **WaveStMod** = 0.

WaveTMax sets the length of the incident wave kinematics time series, but it also determines the frequency step used in the inverse FFT, from which the internal wave time series are derived ($= 2/\text{WaveTMax}$). If **WaveTMax** is less than the total simulation time, HydroDyn implements repeating wave kinematics that have a period of **WaveTMax**; **WaveTMax** must not be less than the total simulation time when **WaveMod** = 5. **WaveDT** determines the time step for the wave kinematics time series, but it also determines the maximum frequency in the inverse FFT ($\text{max} = 1/\text{WaveDT}$). When modeling irregular sea states, we recommend that **WaveTMax** be set to at least 1 hour (3600 s) and that **WaveDT** be a value in the range between 0.1 and 1.0 s to ensure sufficient resolution of the wave spectrum and wave kinematics. When HydroDyn is coupled to FAST, **WaveDT** may be specified arbitrarily independently from the glue code time step of FAST (the wave kinematics will be interpolated in time as necessary); **WaveDT** must equal the glue code time step of FAST when **WaveMod** = 6.

For internally generated waves, the wave height (crest-to-trough, twice the amplitude) for regular waves and the significant wave height for irregular waves is set using **WaveHs** (only used when **WaveMod** = 1, 2, or 3). The wave period for regular waves and the peak-spectral wave period for irregular waves is controlled with the **WaveTp** parameter (only used when **WaveMod** = 1 or 2). **WavePkShp** is the peak-shape parameter of JONSWAP irregular wave spectrum (only used when **WaveMod** = 2). Set **WavePkShp** to DEFAULT to obtain the value recommended in the IEC 61400-3 Annex B, derived based on the peak-spectral period and significant wave height [IEC, 2009]. Set **WavePkShp** to 1.0 for the Pierson-Moskowitz spectrum.

WvLowCoff and **WvHiCoff** control the lower and upper cut-off frequencies (in rad/s) of the first-order wave spectrum; the first-order wave-component amplitudes are zeroed below and above these cut-off frequencies, respectively. **WvLowCoff** may be set lower than the low-energy limit of the first-order wave spectrum to minimize computational expense. Setting a proper upper cut-off frequency (**WvHiCoff**) also minimizes computational expense and is important to prevent nonphysical effects when approaching of the breaking-wave limit and to avoid nonphysical wave forces at high frequencies (i.e., at short wavelengths) when using a strip-theory solution. **WvLowCoff** and **WvHiCoff** are unused when **WaveMod** = 0, 1, or 6.

WaveDir (unused when **WaveMod** = 0 or 6) is the mean wave propagation heading direction (in degrees), and must be in the range (-180,180]. A heading of 0 corresponds to wave propagation in the positive X-axis direction. And a heading of 90 corresponds to wave propagation in the positive Y-axis direction. **WaveDirMod** specifies the wave directional spreading model (only used when **WaveMod** = 2, 3, or 4). Setting **WaveDirMod** to 0 disables directional spreading, resulting in long-crested (plane-progressive) sea states propagating in the **WaveDir** direction. Setting **WaveDirMod** to 1 enables the modeling of short-crested sea states, with a mean propagation direction of **WaveDir**, through the

commonly used cosine spreading function (COS:sup:2S) to define the directional spreading spectrum, based on the spreading coefficient (S) defined via **WaveDirSpread**. The wave directional spreading spectrum is discretized with an equal-energy method using **WaveNDir** number of equal-energy bins. **WaveNDir** is an odd-valued integer greater or equal to 1 (1 or 3 or 5...), but HydroDyn may slightly increase the specified value of **WaveNDir** to ensure that there is the same number of wave components within each direction bin; setting **WaveNDir** = 1 is equivalent to setting **WaveDirMod** = 0. The range of the directional spread (in degrees) is defined via **WaveDirSpread**. The equal-energy method assumes that the directional spreading spectrum is the product of a frequency spectrum and a spreading function i.e. $S(\cdot) = S(D(\cdot))$. Directional spreading is not permitted when using Newman's approximation of the second-order difference-frequency potential-flow loads.

WaveSeed(1) and **WaveSeed(2)** (unused when **WaveMod** = 0, 5, or 6) combined determine the initial seed (starting point) for the internal pseudorandom number generator (pRNG) needed to derive the internal wave kinematics from the wave frequency and direction spectra. If both are numeric values, the Fortran intrinsic pRNG is used. If **WaveSeed(2)** is the string "RANLUX", an alternative pRNG included with the NWTC Library is used and the value of **WaveSeed(1)** is the seed. If you want to run different time-domain realizations for given boundary conditions (of significant wave height, and peak-spectral period, etc.), you should change one or both seeds between simulations. While the phase of each wave frequency and direction component of the wave spectrum is always based on a uniform distribution (except when using the 1P# **WaveMod** option), the amplitude of the wave frequency spectrum can also be randomized (following a normal distribution) by setting **WaveNDamp** to TRUE. Setting **WaveNDamp** to FALSE means that the amplitude of the wave frequency spectrum always matches the target spectrum. **WaveNDamp** is only used with **WaveMod** = 2, 3, or 4.

When using externally generated wave data (**WaveMod** = 5 or 6), input parameter **WvKinFile** should be set to the root name of the input file(s) (without extension) containing the data.

Using externally generated wave-elevation time series (**WaveMod** = 5) requires a text-formatted input data file with the extension *.Elev* containing two columns of data—the first is time (starting at zero) (in s) and the second is the wave elevation at (0,0) (in m), separated by whitespace. Header lines (identified as those not beginning with a number) are ignored. The time series must be at least **WaveTMax** in length and not less than the total simulation time and the time step must match **WaveDT**. The wave-elevation time series specified is assumed to be of first order and long-crested, but is not checked for physical correctness. When second-order terms are optionally enabled—see the 2ND-ORDER WAVES and 2ND-ORDER FLOATING PLATFORM FORCES sections below—the second-order terms are calculated using the wave-component amplitudes derived from the provided wave-elevation time series and extra energy is added to the wave spectrum (at the difference and sum frequencies).

Using full externally generated wave kinematics (**WaveMod** = 6) requires eight text-formatted input data files, all without headers. Seven files with extensions *.Vxi*, *.Vyi*, *.Vzi*, *.Axi*, *.Ayi*, *.Azi*, and *.DynP* correspond to the X , Y , and Z velocities (in m/s) and accelerations (in m/s^2) in the global inertial-frame coordinate system and the dynamic pressure (in Pa) time series. Each of these files must have exactly **WaveTMax/DT** rows and N whitespace-separated columns, where N is the total number of internal HydroDyn analysis nodes (corresponding exactly to those written to the HydroDyn summary file). Time is absent from the files, but is assumed to go from zero to **WaveTMax** – **WaveDT** in steps of **WaveDT**. To use this feature, it is the burden of the user to generate wave kinematics data at each of HydroDyn's time steps and analysis nodes. HydroDyn will not interpolate the data; as such, when HydroDyn is coupled to FAST, **WaveDT** must equal the glue code time step of FAST. A numerical value (including 0) in a file is assumed to be valid data (with 0 corresponding to 0 m/s, 0 m/s^2 , or 0 Pa); a nonnumeric string will designate that the node is outside of the water at that time step (above the instantaneous water elevation or below the seabed)—externally generated wave kinematics used with **WaveMod** = 6 are not limited to the domain between a flat seabed and SWL and may consider wave stretching, higher-order wave theories, or an uneven seabed. All seven files must have nonnumeric strings in the same locations within the file. The eighth file, with extension *.Elev*, must contain the wave elevation (in m) at each of the **NWaveElev** points on the SWL where wave elevations can be output—see below; this data is required for output purposes only and is not used by HydroDyn for other means. This file must have exactly **WaveTMax/DT** rows and **NWaveElev** whitespace-separated columns and only valid numeric data is allowed (the file will have **NWaveElev** + (**WaveElevNX** × **WaveElevNY**) columns when HydroDyn is operated in standalone mode). The data in these files is not processed (filtered, etc.) or checked for physical correctness (other than for consistency in the location of the nonnumeric strings). Full externally generated wave kinematics (**WaveMod** = 6) cannot be used in conjunction with the potential-flow solution.

You can generate up to 9 wave elevation outputs. **NWaveElev** determines the number (between 0 and 9), and the whitespace-separated lists of **WaveElevxi** and **WaveElevyi** determine the locations of these **NWaveElev** number of points on the SWL plane in the global inertial-frame coordinate system.

2nd-Order Waves

The 2ND-ORDER WAVES section (unused when **WaveMod** = 0 or 6) of the input file allows the option of adding second-order contributions to the wave kinematics used by the strip-theory solution. When second-order terms are optionally enabled, the second-order terms are calculated using the first-order wave-component amplitudes and extra energy is added to the wave spectrum (at the difference and sum frequencies). The second-order terms cannot be computed without also including the first-order terms from the WAVES section above. Enabling the second-order terms allows one to capture some of the nonlinearities of real surface waves, permitting more accurate modeling of sea states and the associated wave loads at the expense of greater computational effort (mostly at HydroDyn initialization).

While the cut-off frequencies in this section apply to both the second-order wave kinematics used by strip theory and the second-order diffraction loads in potential-flow theory, the second-order terms themselves are enabled separately. The second-order wave kinematics used by strip theory are enabled in this section while the second-order diffraction loads in potential-flow theory are enabled in the *2nd-Order Floating Platform Forces* section below. While the second-order effects are included when enabled, the wave elevations output from HydroDyn will only include the second-order terms when the second-order wave kinematics are enabled in this section.

To use second-order wave kinematics in the strip-theory solution, set **WvDiffQTF** and/or **WvSumQTF** to TRUE. When **WvDiffQTF** is set to TRUE, second-order difference-frequency terms, calculated using the full difference-frequency QTF, are incorporated in the wave kinematics. When **WvSumQTF** is set to TRUE, second-order sum-frequency terms, calculated using the full sum-frequency QTF, are incorporated in the wave kinematics. The full difference- and sum-frequency wave kinematics QTFs are implemented analytically following [Sharma and Dean, 1981], which extends Stokes second-order theory to irregular multidirectional waves. A setting of FALSE disregards the second-order contributions to the wave kinematics in the strip-theory solution.

WvLowCoffD and **WvHiCoffD** control the lower and upper cut-off frequencies (in rad/s) of the second-order difference-frequency terms; the second-order difference-frequency terms are zeroed below and above these cut-off frequencies, respectively. The cut-offs apply directly to the physical difference frequencies, not the two individual first-order frequency components of the difference frequencies. When enabling second-order potential-flow theory, a setting of **WvLowCoffD** = 0 is advised to avoid eliminating the mean-drift term (second-order wave kinematics do not have a nonzero mean). **WvHiCoffD** need not be set higher than the peak-spectral frequency of the first-order wave spectrum ($p = 2/\text{WaveTp}$) to minimize computational expense.

Likewise, **WvLowCoffS** and **WvHiCoffS** control the lower and upper cut-off frequencies (in rad/s) of the second-order sum-frequency terms; the second-order sum-frequency terms are zeroed below and above these cut-off frequencies, respectively. The cut-offs apply directly to the physical sum frequencies, not the two individual first-order frequency components of the sum frequencies. **WvLowCoffS** need not be set lower than the peak-spectral frequency of the first-order wave spectrum ($p = 2/\text{WaveTp}$) to minimize computational expense. Setting a proper upper cut-off frequency (**WvHiCoffS**) also minimizes computational expense and is important to (1) ensure convergence of the second-order summations, (2) avoid unphysical “bumps” in the wave troughs, (3) prevent nonphysical effects when approaching of the breaking-wave limit, and (4) avoid nonphysical wave forces at high frequencies (i.e., at short wavelengths) when using a strip-theory solution.

Because the second-order terms are calculated using the first-order wave-component amplitudes, the second-order cut-off frequencies (**WvLowCoffD**, **WvHiCoffD**, **WvLowCoffS**, and **WvHiCoffS**) are used in conjunction with the first-order cut-off frequencies (**WvLowCoff** and **WvHiCoff**) from the WAVES section. However, the second-order cut-off frequencies are not used by Newman’s approximation of the second-order difference-frequency potential-flow loads, which are derived solely from first-order effects.

Current

You can include water velocity due to a current model by setting **CurrMod** = 1. If **CurrMod** is set to zero, then the simulation will not include current. **CurrMod** = 2 requires that the *UserCurrent()* subroutine of the *Current.f90* source file be implemented by the user, and will require recompiling either the standalone HydroDyn program or FAST. Current induces steady hydrodynamic loads through the viscous-drag terms (both distributed and lumped) of strip-theory members. Current is not used in the potential-flow solution or when **WaveMod** = 6.

HydroDyn's standard current model includes three sub-models: near-surface, sub-surface, and depth-independent, as illustrated in Fig. 4.42. All three currents are vector summed, along with the wave particle kinematics velocity.

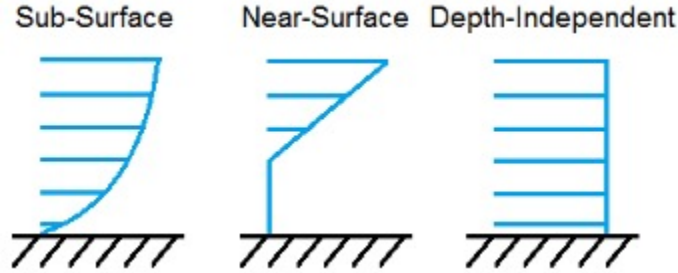


Fig. 4.42: Standard Current Sub-Models

The sub-surface current model follows a power law,

$$U_{SS}(Z) = U_{0SS} \left(\frac{Z + d}{d} \right)^{\frac{1}{7}} \quad (4.181)$$

where Z is the local depth below the SWL (negative downward), d is the water depth (equal to **WtrDpth** + **MSL2SWL**), and U_{0SS} is the current velocity at SWL, corresponding to **CurrSSV0**. The heading of the sub-surface current is defined using **CurrSSDir** following the same convention as **WaveDir**.

The near-surface current model follows a linear relationship down to a reference depth such that,

$$U_{NS}(Z) = U_{0NS} \left(\frac{Z + h_{ref}}{h_{ref}} \right), Z \in [-h_{ref}, 0] \quad (4.182)$$

otherwise,

$$U_{NS}(Z) = 0 \quad (4.183)$$

where h_{ref} is the reference depth corresponding to **CurrNSRef** and must be positive valued. U_{0NS} is the current velocity at SWL, corresponding to **CurrNSV0**. The heading of the near-surface current is defined using **CurrNSDir**, following the same convention as **WaveDir**.

The depth-independent current velocity everywhere equals **CurrDIV**. This current has a heading direction **CurrDIDir**, following the same convention as **WaveDir**.

Floating Platform

This and the next few sections of the input file have “Floating Platform” in the title, but the input parameters control the potential-flow model, regardless of whether the substructure is floating or not. The potential-flow solution cannot be used in conjunction with nonzero **MSL2SWL** or **WaveMod** = 6.

If the load contributions from potential-flow theory are to be used, set **PotMod** to 1 for the use of frequency-to-time-domain transforms based on WAMIT output or 2 for the use of FIT (FIT is not yet documented in this manual). With **PotMod** = 1, include the root name (without extensions) for the WAMIT-related output files in **PotFile**. These files consist of the *.I*, *.3*, *.hst* and second-order files. These are written by the WAMIT program and should not include any file headers. When the linear state-space model is used in place of convolution, the *.ss* file generated by **SS_Fitting** must have the same root name as the other WAMIT-related files (see **RdtnMod** below). The remaining parameters in this section are only used when **PotMod** = 1.

The output files from WAMIT are in a standard nondimensional form that HydroDyn will dimensionalize internally upon input. **WAMITULEN** is the characteristic body length scale used to redimensionalize the WAMIT output. The body motions and forces in these files are in relation to the WAMIT reference point (WRP) in HydroDyn, which for the undisplaced substructure is the same as the origin of the global inertial-frame coordinate system (0,0,0). The *.hst* file contains the 6x6 linear hydrostatic restoring (stiffness) matrix of the platform. The *.I* file contains the 6x6 frequency-dependent hydrodynamic added-mass and damping matrix of the platform from the radiation problem. The *.3* file contains the 6x1 frequency- and direction-dependent first-order wave-excitation force vector of the platform from the linear diffraction problem. While HydroDyn expects hydrodynamic coefficients derived from WAMIT, if you are not using WAMIT, it is recommended that you reformat your data according to the WAMIT format (including nondimensionalization) before inputting them to HydroDyn. Information on the WAMIT format is available from Chapter 4 of the WAMIT User’s Guide [LN06].

PtfmVol0 is the displaced volume of water when the platform is in its undisplaced position. This value should be set equal to the value computed by WAMIT as output in the WAMIT *.out* file. **PtfmCOBxt** and **PtfmCOByt** are the *X* and *Y* offsets of the center of buoyancy from the WRP.

HydroDyn has two methods for calculating the radiation memory effect. Set **RdtnMod** to 1 for the convolution method, 2 for the linear state-space model, or 0 to disable the memory effect calculation. For the convolution method, **RdtnTMax** determines how long to track the memory effect (truncating the convolutions at $t - \text{RdtnTMax}$, where t is the current simulation time), but it also determines the frequency step used in the cosine transform, from which the time-domain radiation kernel (radiation impulse-response function) is derived. A **RdtnTMax** of 60 s is usually more than sufficient because the radiation kernel decays to zero after a short amount of time; setting **RdtnTMax** much greater than this will cause HydroDyn to run significantly slower. (**RdtnTMax** does not need to match or exceed the total simulation length.) Setting **RdtnTMax** to 0 s disables the memory effect, akin to setting **RdtnMod** to 0. For the convolution method, **RdtnDT** is the time step for the radiation calculations (numerical convolutions), but also determines the maximum frequency in the cosine transform. For the state-space model, **RdtnDT** is the time step to use for time integration of the linear state-space model. In this version of HydroDyn, **RdtnDT** must match the glue code (FAST/driver program) simulation time step; the DEFAULT keyword can be used for this.

2nd-Order Floating Platform Forces

The 2ND-ORDER FLOATING PLATFORM FORCES section of the input file allows the option of adding second-order contributions to the potential-flow solution. When second-order terms are optionally enabled, the second-order terms are calculated using the first-order wave-component amplitudes and extra energy is added to the wave spectrum (at the difference and sum frequencies). The second-order terms cannot be computed without also including the first-order terms from the FLOATING PLATFORM section above (**PotMod** = 1). Enabling the second-order terms allows one to capture some of the nonlinearities of real surface waves, permitting more accurate modeling of sea states and the associated wave loads at the expense of greater computational effort (mostly at HydroDyn initialization).

While the cut-off frequencies in the *2nd-Order Waves* section above apply to both the second-order wave kinematics used by strip theory and the second-order diffraction loads in potential-flow theory, the second-order terms themselves

are enabled separately. The second-order wave kinematics used by strip theory are enabled in the [2nd-Order Waves](#) section above while the second-order diffraction loads in potential-flow theory are enabled in this section. While the second-order effects are included when enabled, the wave elevations output from HydroDyn will only include the second-order terms when the second-order wave kinematics are enabled in the [2nd-Order Waves](#) section above.

The second-order difference-frequency potential-flow terms can be enabled in one of three ways. To compute only the mean-drift term, set **MnDrift** to a nonzero value; to estimate the mean- and slow-drift terms using Standing et al.'s extension to Newman's approximation, based only on first-order effects, set **NewmanApp** to a nonzero value; or to compute the mean- and slow-drift terms using the full difference-frequency QTF set **DiffQTF** to a nonzero value. Valid values of **MnDrift** are 0, 7, 8, 9, 10, 11, or 12 corresponding to which WAMIT output file the mean-drift terms will be calculated from. Valid values of **NewmanApp** are 0, 7, 8, 9, 10, 11, or 12 corresponding to which WAMIT output file the Newman's approximation will be calculated from. Newman's approximation cannot be used in conjunction with directional spreading (**WaveDirMod** must be 0) and the second-order cut-off frequencies do not apply to Newman's approximation. Valid values of **DiffQTF** are 0, 10, 11, or 12 corresponding to which WAMIT output file the full difference-frequency potential-flow solution will be calculated from. Only one of **MnDrift**, **NewmanApp**, and **DiffQTF** can be nonzero; a setting of 0 disregards the second-order difference-frequency contributions to the potential-flow solution.

The .7 WAMIT file refers to the mean-drift loads (diagonal of the difference-frequency QTF) in all 6 DOFs derived from the control-surface integration method based on the first-order solution. The .8 WAMIT file refers to the mean-drift loads (diagonal of the difference-frequency QTF) only in surge, sway, and roll derived from the momentum conservation principle based on the first-order solution. The .9 WAMIT file refers to the mean-drift loads (diagonal of the difference-frequency QTF) in all six DOFs derived from the pressure integration method based on the first-order solution. For the difference-frequency terms, 10, 11, and 12 refer to the WAMIT .10d, .11d, and .12d files, corresponding to the full QTF of (.10d*) loads in all 6 DOFs associated with the quadratic interaction of first-order quantities, (.11d*) total (quadratic plus second-order potential) loads in all 6 DOFs derived by the indirect method, and (.12d*) total (quadratic plus second-order potential) loads in all 6 DOFs derived by the direct method, respectively.

The second-order sum-frequency potential-flow terms can only be enabled using the full sum-frequency QTF, by setting **SumQTF** to a nonzero value. Valid values of **SumQTF** are 0, 10, 11, or 12 corresponding to which WAMIT output file the full sum-frequency potential-flow solution will be calculated from; a setting of 0 disregards the second-order sum-frequency contributions to the potential-flow solution. For the sum-frequency terms, 10, 11, and 12 refer to the WAMIT .10s, .11s, and .12s files, corresponding to the full QTF of (.10s*) loads in all 6 DOFs associated with the quadratic interaction of first-order quantities, (.11s*) total (quadratic plus second-order potential) loads in all 6 DOFs derived by the indirect method, and (.12s*) total (quadratic plus second-order potential) loads in all 6 DOFs derived by the direct method, respectively.

Platform Additional Stiffness and Damping

The vectors and matrices of this section are used to generate additional loads on the platform (in addition to other hydrodynamic terms calculated by HydroDyn), per the following equation.

$$\vec{F}_{Add} = \vec{F}_0 - [C]\vec{q} - [B]\dot{\vec{q}} - [B_{quad}]ABS\left(\dot{\vec{q}}\right)\dot{\vec{q}} \quad (4.184)$$

where \vec{F}_0 corresponds to the **AddF0** 6x1 static load (preload) vector, $[C]$ corresponds to the **AddCLin** 6x6 linear restoring (stiffness) matrix, $[B]$ corresponds to the **AddBLin** 6x6 linear damping matrix, $[B_{quad}]$ corresponds to the **AddBQuad** 6x6 quadratic drag matrix, and \vec{q} corresponds to the WRP 6x1 (six-DOF) displacement vector (three translations and three rotations), where the overdot refers to the first time-derivative.

These terms can be used, e.g., to model a linearized mooring system, to augment strip-theory members with a linear hydrostatic restoring matrix (see [Section 4.2.8](#)), or to “tune” HydroDyn to match damping to experimental results, such as free-decay tests. While likely most useful for floating systems, these matrices can also be used for fixed-bottom systems; in both cases, the resulting load is applied at the WRP, which when HydroDyn is coupled to FAST, get applied to the platform in ElastoDyn (bypassing SubDyn for fixed-bottom systems). See [Modeling Considerations](#) for addition guidance for where these terms are necessary.

Axial Coefficients

This and the next several sections of the input file control the strip-theory model for both fixed-bottom and floating substructures.

HydroDyn computes lumped viscous-drag, added-mass, fluid-inertia, and static pressure loads at member ends (joints). The hydrodynamic coefficients for the lumped the lumped loads at joints are referred to as “axial coefficients” and include viscous-drag coefficients, **AxCd**, added-mass coefficients, **AxCa**, and dynamic-pressure coefficients, **AxCp**. **AxCa** influences both the added-mass loads and the scattering component of the fluid-inertia loads. Any number of separate axial coefficient sets, distinguished by **AXCoefID**, may be specified by setting **NAXCoef** > 1.

Axial viscous-drag loads will be calculated for all specified member joints. Axial added-mass, fluid-inertia, and static-pressure loads will only be calculated for member joints of members not modeled with potential flow (**PropPot** = FALSE). Axial loads are only calculated at user-specified joints. Axial loads are not calculated at joints HydroDyn may automatically create as part its solution process. For example, if you want axial effects at a marine-growth boundary (where HydroDyn automatically adds a joint), you must explicitly set a joint at that location.

Member Joints

The strip-theory model is based on a substructure composed of joints interconnected by members. **NJoints** is the user-specified number of joints and determines the number of rows in the subsequent table. Because a member connects two nodes, **NJoints** must be exactly zero or greater than or equal to two. Each joint listed in the table is identified by a unique integer, **JointID**. The (X,Y,Z) coordinate of each joint is specified in the global inertial-frame coordinate system via **Jointxi**, **Jointyi**, and **Jointzi**, respectively. **JointAxID** corresponds to an entry in the AXIAL COEFFICIENTS table and sets the axial coefficients for a joint. This version of HydroDyn cannot calculate joint overlap when multiple members meet at a common joint; therefore **JointOvrIp** must be set to 0. Future releases will enable joint overlap calculations.

Modeling a fixed-bottom substructure embedded into the seabed (e.g., through piles or suction buckets) requires that the lowest member joint(s) lie below the water depth. Placing a joint at or above the water depth results in static pressure loads being applied.

Member Cross-Sections

Members in HydroDyn are assumed to be straight circular (and possibly tapered) cylinders. Apart from the hydrodynamic coefficients, the circular cross-section properties needed for the hydrodynamic load calculations are member outer diameter, **PropD**, and member thickness, **PropThck**. You will need to create an entry in this table, distinguished by **PropSetID**, for each unique combination of these two properties. The member property-set table contains **NPropSets** rows. The member property sets are referred to by their **PropSetID** in the MEMBERS table, as described in [Section 4.2.8](#) below. **PropD** determines the static buoyancy loads exterior to a member, as well as the area used in the viscous-drag calculation and the volume used in the added-mass and fluid-inertia calculations. **PropThck** determines the interior volume for fluid-filled (flooded/ballasted) members.

Hydrodynamic Coefficients

HydroDyn computes distributed viscous-drag, added-mass, fluid-inertia, and static buoyancy loads along members.

The hydrodynamic coefficients for the distributed strip-theory loads are specified using any of three models, which we refer to as the simple model, a depth-based model, and a member-based model. All of these models require the specification of both transverse and axial hydrodynamic coefficients for viscous drag, added mass, and dynamic pressure (axial viscous drag is not yet available). The added-mass coefficient influences both the added-mass loads and the scattering component of the fluid-inertia loads. There are separate set of hydrodynamic coefficients both with and without marine growth. A given element will either use the marine growth or the standard version of a coefficient, but never both. Note that input members are split into elements, one of the splitting rules guarantees the previous statement is true. Which members have marine growth is defined by the MARINE GROWTH table of [Section 4.2.8](#). You can specify only one model type, **MCoeffMod**, for any given member in the MEMBERS table. However, different members can specify different coefficient models.

In the hydrodynamic coefficient input parameters, **Cd**, **Ca**, and **Cp** refer to the viscous-drag, added-mass, and dynamic-pressure coefficients, respectively, **MG** identifies the coefficients to be applied for members with marine growth (the standard values are identified without **MG**), and **Ax** identifies the axial coefficients to be applied for tapered members (the transverse coefficients are identified without **Ax**). It is noted that for the transverse coefficients, , the inertia coefficient.

While the strip-theory solution assumes circular cross sections, the hydrodynamic coefficients can include shape corrections; however, there is no distinction made in HydroDyn between different transverse directions.

Simple Model

This table consists of a single complete set of hydrodynamic coefficients as follows: **SimplCd**, **SimplCdMG**, **SimplCa**, **SimplCaMG**, **SimplCp**, **SimplCpMG**, **SimplAxCa**, **SimplAxCaMG**, **SimplAxCp**, and **SimplAxCpMG**. These hydrodynamic coefficients are referenced in the members table of [Section 4.2.8](#) by selecting **MCoeffMod** = 1.

Depth-Based Model

The depth-based coefficient model allows you to specify a series of depth-dependent coefficients. **NCoeffDpth** is the user-specified number of depths and determines the number of rows in the subsequent table. Currently, this table requires that the rows are ordered by increasing depth, **Dpth**; this is equivalent to a decreasing global Z-coordinate. The hydrodynamic coefficients at each depth are as follows: **DpthCd**, **DpthCdMG**, **DpthCa**, **DpthCaMG**, **DpthCp**, **DpthCpMG**, **DpthAxCa**, **DpthAxCaMG**, **DpthAxCp**, and **DpthAxCpMG**. Members use these hydrodynamic coefficients by setting **MCoeffMod** = 2. The HydroDyn module will interpolate coefficients for a node whose Z-coordinate lies between table Z-coordinates.

Member-Based Model

The member-based coefficient model allows you to specify a hydrodynamic coefficients for each particular member. **NCoeffMembers** is the user-specified number of members with member-based coefficients and determines the number of rows in the subsequent table. The hydrodynamic coefficients for a member distinguished by **MemberID** are as follows: **MemberCd1**, **MemberCd2**, **MemberCdMG1**, **MemberCdMG2**, **MemberCa1**, **MemberCa2**, **MemberCaMG1**, **MemberCaMG2**, **MemberCp1**, **MemberCp2**, **MemberCpMG1**, **MemberCpMG2**, **MemberAxCa1**, **MemberAxCa2**, **MemberAxCaMG1**, **MemberAxCaMG2**, **MemberAxCp1**, **MemberAxCp2**, **MemberAxCpMG1**, and **MemberAxCpMG2**, where 1 and 2 identify the starting and ending joint of the member, respectively. Members use these hydrodynamic coefficients by setting **MCoeffMod** = 3.

Members

NMembers is the user-specified number of members and determines the number of rows in the subsequent table. For each member distinguished by **MemberID**, **MJointID1** specifies the starting joint and **MJointID2** specifies the ending joint, corresponding to an identifier (**JointID**) from the MEMBER JOINTS table. Likewise, **MPropSetID1** corresponds to the starting cross-section properties and **MProSetID2** specify the ending cross-section properties, allowing for tapered members. **MDivSize** determines the maximum spacing (in meters) between simulation nodes where the distributed loads are actually computed; the smaller the number, the finer the resolution and longer the computational time. Each member in your model will have hydrodynamic coefficients, which are specified using one of the three models (**MCoefMod**). Model 1 uses a single set of coefficients found in the SIMPLE HYDRODYNAMIC COEFFICIENTS section. Model 2 is depth-based, and is determined via the table found in the DEPTH-BASED HYDRODYNAMIC COEFFICIENTS section. Model 3 specifies coefficients for a particular member, by referring to the MEMBER-BASED HYDRODYNAMIC COEFFICIENTS section. The **PropPot** flag indicates whether the corresponding member coincides with the body represented by the potential-flow solution. When **PropPot** = TRUE, only viscous-drag loads, and ballasting loads will be computed for that member.

Filled Members

Members—whether they are also modeled with potential-flow or not—may be fluid-filled, meaning that they are flooded and/or ballasted. Fluid-filled members introduce interior buoyancy that subtracts from the exterior buoyancy and a mass. Both distributed loads along a member and lumped loads at joints are applied. The volume of fluid in the member is derived from the outer diameter and thickness of the member and a fluid-filled free-surface level. The fluid in the member is assumed to be compartmentalized such that it does not slosh. Rotational inertia of the fluid in the member is ignored. A member's filled configuration is defined by the filled-fluid density and the free-surface level. Filled members that have the same configuration are collected into fill groups.

NFillGroups specifies the number of fluid-filled member groups and determines the number of rows in the subsequent table. **FillNumM** specifies the number of members in the fill group. **FillMList** is a list of **FillNumM** whitespace-separated **MemberIDs**. **FillFSLoc** specifies the Z-height of the free-surface (0 for MSL). **FillDens** is the density of the fluid. If **FillDens** = DEFAULT, then **FillDens** = **WtrDens**.

Marine Growth

Members not also modeled with potential-flow theory may be modeled with marine growth. Marine growth causes three effects. First, marine growth introduces a static weight and mass to a member, applied as distributed loads along the member. Second, marine growth increases the outer diameter of a member, which impacts the diameter used in the viscous-drag, added-mass, fluid-inertia, and static buoyancy load calculations. Third, the hydrodynamic coefficients for viscous drag, added mass, and dynamic pressure are specified distinctly for marine growth. Rotational inertia of the marine growth is ignored and marine growth is not added to member ends.

Marine growth is specified using a depth-based table with **NMGDepths** rows. This table must have exactly zero or at least 2 rows. The columns in the table include the local depth, **MGDpth**, the marine growth thickness, **MGThck**, and marine growth density, **MGDens**. Marine growth for a particular location in the substructure geometry is added by linearly interpolating between the marine-growth table entries. The smallest and largest values of **MGDpth** define the marine growth region. Outside this region the marine growth thickness is set to zero. If you want sub-regions of zero marine growth thickness within these bounds, you must generate depth entries which explicitly set **MGThck** to zero. The hydrodynamic coefficient tables contain coefficients with and without marine growth. If **MGThck** = 0 for a particular node, the coefficients not associated with marine growth are used.

Member Output List

HydroDyn can output distributed load and wave kinematic quantities at up to 9 locations on up to 9 different members, for a total of 81 possible local member output locations. **NMOutputs** specifies the number of members. You must create a table entry for each requested member. Within a table entry, **MemberID** is the ID specified in the MEMBERS table, and **NOutLoc** specifies how many output locations are generated for this member. **NodeLocs** specifies those locations as a normalized distance from the starting joint (0.0) to the ending joint (1.0) of the member. If the chosen location does not align with a calculation node, the results at the two surrounding nodes will be linearly interpolated. The outputs specified in [Appendix C. List of Output Channels](#) determines which quantities are actually output at these locations.

Joint Output List

HydroDyn can output lumped load and wave kinematic quantities at up to 9 different joints. **JOutLst** contains a list of **NJOutputs** number of **JointIDs**. The outputs specified in [Appendix C. List of Output Channels](#) determines which quantities are actually output at these joints.

Output

Specifying **HDSum** = TRUE causes HydroDyn to generate a summary file with name **OutRootname.HD.sum**. **OutRootName** is either specified in the HYDRODYN section of the driver input file when running HydroDyn standalone, or by the FAST program when running a coupled simulation. See [Section 4.2.8](#) for summary file details.

For this version, **OutAll** must be set to FALSE. In future versions, setting **OutAll** = TRUE will cause HydroDyn to auto-generate outputs for every joint and member in the input file.

If **OutSwitch** is set to 1, outputs are sent to a file with the name **OutRootname.HD.out**. If **OutSwitch** is set to 2, outputs are sent to the calling program (FAST) for writing. If **OutSwitch** is set to 3, both file outputs occur. In standalone mode, setting **OutSwitch** to 2 results in no output file being produced.

The **OutFmt** and **OutSFmt** parameters control the formatting for the output data and the channel headers, respectively. HydroDyn currently does not check the validity of these format strings. They need to be valid Fortran format strings. Since the **OutSFmt** is used for the column header and **OutFmt** is for the channel data, in order for the headers and channel data to align properly, the width specification should match. For example,

```
"E11.4" OutFmt
"A11" OutSFmt
```

Output Channels

This section controls output quantities generated by HydroDyn. Enter one or more lines containing quoted strings that in turn contain one or more output parameter names. Separate output parameter names by any combination of commas, semicolons, spaces, and/or tabs. If you prefix a parameter name with a minus sign, “-”, underscore, “_”, or the characters “m” or “M”, HydroDyn will multiply the value for that channel by -1 before writing the data. The parameters are not necessarily written in the order they are listed in the input file. HydroDyn allows you to use multiple lines so that you can break your list into meaningful groups and so the lines can be shorter. You may enter comments after the closing quote on any of the lines. Entering a line with the string “END” at the beginning of the line or at the beginning of a quoted string found at the beginning of the line will cause HydroDyn to quit scanning for more lines of channel names. Member- and joint-related quantities are generated for the requested [Member Output List](#) and [Joint Output List](#). If HydroDyn encounters an unknown/invalid channel name, it warns the users but will remove the suspect channel from the output file. Please refer to Appendix C for a complete list of possible output parameters.

Output Files

HydroDyn produces four types of output files: an echo file, a wave-elevations file, a summary file, and a time-series results file. The following sections detail the purpose and contents of these files.

Echo Files

If you set the **Echo** flag to TRUE in the HydroDyn driver file or the HydroDyn primary input file, the contents of those files will be echoed to a file with the naming conventions, **OutRootName.dvr.ech** for the driver input file and **OutRootName.HD.ech** for the HydroDyn primary input file. **OutRootName** is either specified in the HYDRODYN section of the driver input file, or by the FAST program. The echo files are helpful for debugging your input files. The contents of an echo file will be truncated if HydroDyn encounters an error while parsing an input file. The error usually corresponds to the line after the last successfully echoed line.

Wave-Elevations File

Setting **WaveElevSeriesFlag** in the driver file to TRUE enables the outputting of a grid of wave elevations to a text-based file with the name **OutRootName.WaveElev.out**. The grid consists of **WaveElevNX** by **WaveElevNY** wave elevations (centered at $X = 0$, $Y = 0$) with a **dX** and **dY** spacing in the global inertial-frame coordinate system. These wave elevations are distinct and output separately from the wave elevations determined by **NWaveElev** in the HydroDyn primary input file, such that the total number of wave elevation outputs is **NWaveElev** + (**WaveElevNX** × **WaveElevNY**). The wave-elevation output file **OutRootName.WaveElev.out** contains the total wave elevation, which is the sum of the first- and second-order terms (when the second-order wave kinematics are optionally enabled).

Summary File

HydroDyn generates a summary file with the naming convention, **OutRootName.HD.sum** if the **HDSum** parameter is set to TRUE. This file summarizes key information about your hydrodynamics model, including buoyancy, substructure volumes, marine growth weight, the simulation mesh and its properties, first-order wave frequency components, and the radiation kernel.

When the text refers to an index, it is referring to a given row in a table. The indexing starts at 1 and increases consecutively down the rows.

WAMIT-model volume and buoyancy information

This section summarizes the buoyancy of the potential-flow-model platform in its undisplaced configuration. For a hybrid potential-flow/strip-theory model, these buoyancy values must be added to any strip-theory member buoyancy reported in the subsequent sections to obtain the total buoyancy of the platform.

Substructure Volume Calculations

This section contains a summary of the total substructure volume, the submerged volume, volume of any marine growth, and fluid-filled (flooded/ballasted) volume for the substructure in its undisplaced configuration. Except for the fluid-filled volume value, the reported volumes are only for members that have the **PropPot** flag set to FALSE. The flooded/ballasted volume applies to any fluid-filled member, regardless of its **PropPot** flag.

Integrated Buoyancy Loads

This section details the buoyancy loads of the undisplaced substructure when summed about the WRP (0,0,0). The external buoyancy includes the effects of marine growth, and only applies to members whose **PropPot** flag is set to FALSE. The internal buoyancy is the negative effect on buoyancy due to flooding or ballasting and is independent of the **PropPot** flag.

Integrated Marine Growth Weights

This section details the marine growth weight loads of the undisplaced substructure when summed about the WRP (0,0,0).

Simulation Node Table

This table details the undisplaced nodal information and properties for all internal analysis nodes used by the HydroDyn model. The node index is provided in the first column. The second column maps the node to the input joint index (not to be confused with the **JointID**). If a value of -1 is found in this column, the node is an interior node and results from an input member being split somewhere along its length due to the requirements of the **MDivSize** parameter in the primary input file members table. The third column indicates if this node is part of a Super Member (**JointOvrtp** = 1). The next column tells you the corresponding input member index (not to be confused with the **MemberID**). **Nxi**, **Nyi**, and **Nzi**, provide the (X,Y,Z) coordinates in the global inertial-frame coordinate system. **InpMbrDist** provides the normalized distance to the node from the start of the input member. **R** is the outer radius of the member at the node (excluding marine growth), and **t** is the member wall thickness at the node. **dRdZ** is the taper of the member at the node, **tMG** is the marine growth thickness, and **MGDens** is the marine growth density. **PropPot** indicates whether the element attached to this node is modeled using potential-flow theory. If **FilledFlag** is TRUE, then **FillDens** gives the filled fluid density and **FillFSLoc** indicates the free-surface height (Z-coordinate). **Cd**, **Ca**, **Cp**, **AxCa**, **AxCp**, **JAXCd**, **JAXCa**, and **JAXCp** are the viscous-drag, added-mass, dynamic-pressure, axial added-mass, axial dynamic-pressure, end-effect axial viscous-drag, end-effect axial added-mass, and end-effect axial dynamic-pressure coefficients, respectively. **NConn** gives the number of elements connected to node, and **Connection List** is the list of element indexes attached to the node.

Simulation Element Table

This section details the undisplaced simulation elements and their associated properties. A suffix of 1 or 2 in a column heading refers to the element's starting or ending node, respectively. The first column is the element index. **node1** and **node2** refer to the node index found in the node table of the previous section. Next are the element **Length** and exterior **Volume**. This exterior volume calculation includes any effects of marine growth. **MGVolume** provides the volume contribution due to marine growth. The cross-sectional properties of outer radius (excluding marine growth), marine growth thickness, and wall thickness for each node are given by **R1**, **tMG1**, **t1**, **R2**, **tMG2**, and **t2**, respectively. **MGDens1** and **MGDens2** are the marine growth density at node 1 and 2. **PropPot** indicates if the element is modeled using potential-flow theory. If the element is fluid-filled (has flooding or ballasting), **FilledFlag** is set to **T** for TRUE. **FillDensity** and **FillFSLoc** are the filled fluid density and the free-surface location's Z-coordinate in the global inertial-frame coordinate system. **FillMass** is calculated by multiplying the **FillDensity** value by the element's interior volume. Finally, the element hydrodynamic coefficients are listed. These are the same coefficients listed in the node table (above).

Summary of User-Requested Outputs

The summary file includes information about all requested member and joint output channels.

Member Outputs

The first column lists the data channel's string label, as entered in the OUTPUT CHANNELS section of the HydroDyn input file. **Xi**, **Yi**, **Zi**, provide the output's undisplaced spatial location in the global inertial-frame coordinate system. The next column, **InpMbrIndx**, tells you the corresponding input member index (not to be confused with the **MemberID**). Next are the coordinates of the starting (**StartXi**, **StartYi**, **StartZi**) and ending (**EndXi**, **EndYi**, **EndZi**) nodes of the element containing this output location. **Loc** is the normalized distance from the starting node of this element.

Joint Outputs

The first column lists the data channel's string label, as entered in the OUTPUT CHANNELS section of the HydroDyn input file. **Xi**, **Yi**, **Zi**, provide the output's undisplaced spatial location in the global inertial-frame coordinate system. **InpJointID** specifies the **JointID** for the output as given in the MEMBER JOINTS table of the HydroDyn input file.

The Wave Number and Complex Values of the Wave Elevations as a Function of Frequency

This section provides the frequency-domain description (in terms of a Discrete Fourier Transform or DFT) of the first-order wave elevation at (0,0) on the free surface, but is not written when **WaveMod** = 0 or 6. The first column, **m**, identifies the index of each wave frequency component. The finite-depth wave number, frequency, and direction of the wave component are given by **k**, **Omega**, and **Direction**, respectively. The last two columns provide the real (**REAL(DFT{WaveElev})**) and imaginary (**IMAG(DFT{WaveElev})**) components of the DFT of the first-order wave elevation. The DFT produces includes both the negative- and positive-frequency components. The negative-frequency components are complex conjugates of the positive frequency components because the time-domain wave elevation is real-valued. The relationships between the negative- and positive-frequency components of the DFT are given by $k(-\omega) = -k(\omega)$ and $H(-\omega) = H(\omega)^*$, where H is the DFT of the wave elevation and $*$ denotes the complex conjugate.

Radiation Memory Effect Convolution Kernel

In the potential-flow solution based on frequency-to-time-domain transforms, HydroDyn computes the radiation kernel used by the convolution method for calculating the radiation memory effect through the cosine transform of the 6x6 frequency-dependent hydrodynamic damping matrix from the radiation problem. The resulting time-domain radiation kernel (radiation impulse-response function)—which is a 6x6 time-dependent matrix—is provided in this section. **n** and **t** give the time-step index and time, which are followed by the elements (**K11**, **K12**, etc.) of the radiation kernel associated with that time. Because the frequency-dependent hydrodynamic damping matrix is symmetric, so is the radiation kernel; thus, only the diagonal and upper-triangular portion of the matrix are provided. The radiation kernel should decay to zero after a short amount of time, which should aid in selecting an appropriate value of **RdtnTMax**.

Results File

The HydroDyn time-series results are written to a text-based file with the naming convention `OutRootName.HD.out` when **OutSwitch** is set to either 1 or 3. If HydroDyn is coupled to FAST and **OutSwitch** is set to 2 or 3, then FAST will generate a master results file that includes the HydroDyn results. The results are in table format, where each column is a data channel (the first column always being the simulation time), and each row corresponds to a simulation output time step. The data channels are specified in the OUTPUT CHANNELS section of the HydroDyn primary input file. The column format of the HydroDyn-generated file is specified using the **OutFmt** and **OutSFmt** parameter of the primary input file.

Modeling Considerations

HydroDyn was designed as an extremely flexible tool for modeling a wide-range of hydrodynamic conditions and substructures. This section provides some general guidance to help you construct models that are compatible with HydroDyn.

Waves

Waves generated internally within HydroDyn can be regular (periodic) or irregular (stochastic) and long-crested (unidirectional) or short-crested (with wave energy spread across a range of directions). Internally, HydroDyn generates waves analytically for finite depth using first-order (linear Airy) or first- plus second-order wave theory [Sharma and Dean, 1981] with the option to include directional spreading, but wave kinematics are only computed in the domain between the flat seabed and SWL and no wave stretching or higher order wave theories are included. Modeling unidirectional sea states is often overly conservative in engineering design. Enabling the second-order terms allows one to capture some of the nonlinearities of real surface waves, permitting more accurate modeling of sea states and the associated wave loads at the expense of greater computational effort (mostly at HydroDyn initialization). The magnitude and frequency content of second-order hydrodynamic loads can excite structural natural frequencies, leading to greater ultimate and fatigue loads than can be predicted solely using first-order theory. Sum-frequency effects are important to the loading of stiff fixed-bottom structures and for the springing and ringing analysis of TLPs. Difference-frequency (mean-drift and slow-drift) effects are important to the analysis of compliant structures, including the motion analysis and mooring loads of catenary-moored floating platforms (spar buoys and semi-submersibles).

When modeling irregular sea states, we recommend that **WaveTMax** be set to at least 1 hour (3600 s) and that **WaveDT** be a value in the range between 0.1 and 1.0 s to ensure sufficient resolution of the wave spectrum and wave kinematics. When HydroDyn is coupled to FAST, **WaveDT** may be specified arbitrarily independently from the glue code time step of FAST. (The wave kinematics and hydrodynamic loads will be interpolated in time as necessary.)

Wave directional spreading is implemented in HydroDyn via the equal-energy method, which assumes that the directional spreading spectrum is the product of a frequency spectrum and a spreading function i.e. $S(\cdot) = S(D)$. Directional spreading is not permitted when using Newman's approximation of the second-order difference-frequency potential-flow loads.

When second-order terms are optionally enabled, the second-order terms are calculated using the first-order wave-component amplitudes and extra energy is added to the wave spectrum (at the difference and sum frequencies). The second-order terms cannot be computed without also including the first-order terms.

It is important to set proper wave cut-off frequencies to minimize computational expense and to ensure that the wave kinematics and hydrodynamic loads are realistic. HydroDyn gives the user six user-defined cut-off frequencies—**WvLowCoff** and **WvHiCoff** for the low- and high-frequency cut-offs of first-order wave components, **WvLowCoffD** and **WvHiCoffD** for the low- and high-frequency cut-offs of second-order difference-frequency wave components, and **WvLowCoffS** and **WvHiCoffS** for low- and high-frequency cut-offs of second-order sum-frequency wave components—none of which have default settings. The second-order cut-offs apply directly to the physical difference and sum frequencies, not the two individual first-order frequency components of the difference and sum frequencies. Because the second-order terms are calculated using the first-order wave-component amplitudes, the second-order

cut-off frequencies are used in conjunction with the first-order cut-off frequencies. However, the second-order cut-off frequencies are not used by Newman's approximation of the second-order difference-frequency potential-flow loads, which are derived solely from first-order effects.

For the first-order wave-component cut-off frequencies, **WvLowCoff** may be set lower than the low-energy limit of the first-order wave spectrum to minimize computational expense. Setting a proper upper cut-off frequency (**WvHiCoff**) also minimizes computational expense and is important to prevent nonphysical effects when approaching of the breaking-wave limit and to avoid nonphysical wave forces at high frequencies (i.e., at short wavelengths) when using a strip-theory solution.

When enabling second-order potential-flow theory, a setting of **WvLowCoffD** = 0 is advised to avoid eliminating the mean-drift term (second-order wave kinematics do not have a nonzero mean). **WvHiCoffD** need not be set higher than the peak-spectral frequency of the first-order wave spectrum ($p = 2/\text{WaveTp}$) to minimize computational expense. **WvLowCoffS** need not be set lower than the peak-spectral frequency of the first-order wave spectrum ($p = 2/\text{WaveTp}$) to minimize computational expense. Setting a proper upper cut-off frequency (**WvHiCoffS**) also minimizes computational expense and is important to (1) ensure convergence of the second-order summations, (2) avoid unphysical "bumps" in the wave troughs, (3) prevent nonphysical effects when approaching of the breaking-wave limit, and (4) avoid nonphysical wave forces at high frequencies (i.e., at short wavelengths) when using a strip-theory solution.

For all models with internally generated wave data, if you want to run different time-domain incident wave realizations for given boundary conditions (of significant wave height, and peak-spectral period, etc.), you should change one or both wave seeds (**WaveSeed(1)** and **WavedSeed(2)**) between simulations.

Wave elevations or full wave kinematics can also be generated externally and used within HydroDyn.

WaveMod = 5 allows the use of externally generated wave-elevation time series, which is useful if you want HydroDyn to simulate specific wave transient events where the wave-elevation time series is known a priori e.g. to match wave-elevation measurements taken from a wave tank or open-ocean test. Internally, HydroDyn will compute an FFT of the provided wave-elevation time series to store the amplitudes and phases of each frequency component, and use those in place of a wave energy spectrum and random seeds to internally derive the hydrodynamic loads in the potential-flow solution or the wave kinematics used in the strip-theory solution. The wave-elevation time series specified is assumed to be of first order and long-crested, but is not checked for physical correctness. The time series must be at least **WaveTMax** in length and not less than the total simulation time and the time step must match **WaveDT**. When second-order terms are optionally enabled, the second-order terms are calculated using the wave-component amplitudes derived from the provided wave-elevation time series and extra energy is added to the wave energy spectrum (at the difference and sum frequencies). Using higher order wave data may produce erroneous results; alternatively, **WvLowCoff** and **WvHiCoff** can be used to filter out energy outside of the first-order wave energy range. The wave-elevation time series output by HydroDyn will only match the specified time series identically if the second-order terms are disabled and the cut-off frequencies are outside the range of wave energy.

WaveMod = 6 allows the use of full externally generated wave kinematics for use with the strip-theory solution (but not the potential-flow solution), completely bypassing HydroDyn's internal wave models. This feature is useful if you want HydroDyn to make use of wave kinematics data derived outside of HydroDyn a priori e.g. from a separate numerical tool, perhaps bypassing some of HydroDyn's internal wave modeling limitations. To use this feature, it is the burden of the user to generate wave kinematics data at each of HydroDyn's time steps and analysis nodes. HydroDyn will not interpolate the data; as such, when HydroDyn is coupled to FAST, **WaveDT** must equal the glue code time step of FAST. Before generating the wave kinematics data externally, users should identify all of the internal analysis nodes by running HydroDyn and generating the summary file—see [Section 4.2.8](#). The fluid domain at each time step are specified by the use of numeric values and nonnumeric strings in the wave data input files. The wave kinematics data specified are not limited to the domain between a flat seabed and SWL and may consider wave stretching, higher-order wave theories, or an uneven seabed. The specified wave kinematics data are not processed (filtered, etc.) or checked for physical correctness. The wave kinematics output by HydroDyn should match the specified data identically.

You can generate up to 9 wave elevation outputs (at different points on the SWL plane) when HydroDyn is coupled to FAST or a large grid of wave elevations when running HydroDyn standalone. While the second-order effects are included when enabled, the wave elevations output from HydroDyn will only include the second-order terms when the second-order wave kinematics are enabled.

Strip-Theory Model Discretization

A user will define the geometry of a structure modeled with strip theory in HydroDyn using joints and members. Members in HydroDyn are assumed to be straight circular (and possibly tapered) cylinders. Members can be further subdivided using **MDivSize**, which HydroDyn will internally use to subdivide members into multiple elements (and nodes). HydroDyn may further refine the geometry at the free surface, flat seabed, marine-growth region, and filled-fluid free surface.

Due to the exponential decay of hydrodynamic loads with depth, a higher resolution near the water free surface is required to capture hydrodynamic loading as waves oscillate about SWL. It is recommended, for instance, that the HydroDyn discretization not exceed element lengths of 0.5 m in the region of the free surface (5 to 10 m above and below SWL), 1.0 m between 25 and 50 m depth, and 2.0 m in deeper waters. When HydroDyn is coupled to SubDyn through FAST for the analysis of fixed-bottom systems, it is recommended that the length ratio between elements of HydroDyn and SubDyn not exceed 10 to 1.

Domain for Strip-Theory Hydrodynamic Load Calculations

Part of the automated geometry refinement mentioned in the above section deals with splitting of input members into sub-elements such that both of the resulting nodes at the element ends lie within the discrete domains described in the following sections.

Distributed Loads

Inertia, Added Mass, Buoyancy, Marine-Growth Weight, Marine-Growth Mass Inertia

These loads are generated at a node as long as **PropPot** = FALSE, the Z-coordinate is in the range $[-\mathbf{WtrDpth}, \mathbf{MSL2SWL}]$, and the element the node is connected to is in the water. When **WaveMod** = 6, the domain is determined by the use of numeric values and nonnumeric strings in the wave data input files.

Viscous Drag

These loads are generated at a node as long as the Z-coordinate is in the range $[-\mathbf{WtrDpth}, \mathbf{MSL2SWL}]$ and the element the node is connected to is in the water. When **WaveMod** = 6, the domain is determined by the use of numeric values and nonnumeric strings in the wave data input files.

Filled Buoyancy, Filled Mass Inertia

These loads are generated at a node as long as the Z-coordinate is in the range $[-\mathbf{WtrDpth}, \mathbf{FillFSLoc}]$ and the element the node is connected to is in the filled fluid.

Lumped Loads

Lumped loads at member ends (axial effects) are only calculated at user-specified joints, and not at joints HydroDyn may automatically create as part its solution process. For example, if you want axial effects at a marine-growth boundary, you must explicitly set a joint at that location.

Added Mass, Inertia, Buoyancy

These loads are generated at a node as long as **PropPot** = FALSE and the Z-coordinate is in the range $[-\mathbf{WtrDpth}, \mathbf{MSL2SWL}]$. When **WaveMod** = 6, the domain is determined by the use of numeric values and non-numeric strings in the wave data input files.

Axial Drag

These loads are generated at a node as long as the Z-coordinate is in the range $[-\mathbf{WtrDpth}, \mathbf{MSL2SWL}]$. When **WaveMod** = 6, the domain is determined by the use of numeric values and nonnumeric strings in the wave data input files.

Filled Buoyancy

These loads are generated at a node as long as the Z-coordinate is in the range $[-\mathbf{WtrDpth}, \mathbf{FillFSLoc}]$

Strip-Theory Hydrodynamic Coefficients

The strip-theory solution of HydroDyn is dependent, among other factors, on user-specified hydrodynamic coefficients, including viscous-drag coefficients, **Cd**, added-mass coefficients, **Ca**, and dynamic-pressure coefficients, **Cp**, for transverse and axial (**Ax**) loads distributed along members and for axial lumped loads at member ends (joints). There are no default settings for these coefficients in HydroDyn. In general, these coefficients are dependent on many factors, including Reynold's number (Re), Keulegan-Carpenter number (KC), surface roughness, substructure geometry, and location relative to the free surface, among others. In practice, the coefficients are (1) selected from tables derived from measurements of flow past cylinders, (2) calculated through high-fidelity computational fluid dynamics (CFD) solutions, or (3) tuned to match experimental results. A value of 1.0 is a plausible guess for all coefficients in the absence of any other information.

While the strip-theory solution assumes circular cross sections, the hydrodynamic coefficients can include shape corrections; however, there is no distinction made in HydroDyn between different transverse directions.

Please note that added-mass coefficients in HydroDyn influence both the added-mass loads and the scattering component of the fluid-inertia loads. For the coefficients associated with transverse loads distributed along members, note that $C_P + C_A = C_M$, the inertia coefficient. For the distributed loads along members, there are separate set of hydrodynamic coefficients both with and without marine growth (**MG**).

Impact of Substructure Motions on Loads

In general, HydroDyn assumes that structural motions of the substructure are small, such that (1) small-angle assumptions apply to structural rotations, (2) the frequency-to-time-domain-based potential-flow solution can be split into uncoupled hydrostatic, radiation, and diffraction solutions, and (3) the hydrodynamic loads dependent on wave kinematics (both from diffraction loads in the potential-flow solution and from the fluid-inertia and viscous-drag loads in the strip-theory solution) can be computed using wave kinematics solved at the undisplaced position of the substructure (the wave kinematics are not recomputed at the displaced position). Nevertheless, HydroDyn uses the substructure motions in the following calculations:

- The structural displacements of the WRP are used in the calculation of the hydrostatic loads (i.e., the change in buoyancy with substructure displacement) in the potential-flow solution.
- The structural velocities and accelerations of the WRP are used in the calculation of the wave-radiation loads (i.e., the radiation memory effect and added mass) in the potential-flow solution.
- The structural displacements and velocities of the WRP are used in the calculation of the additional platform loads (via the Platform Additional Stiffness and Damping).
- The structural velocities of the substructure nodes are used in the calculation of the viscous-drag loads in the strip-theory solution (e.g., the relative form of Morison's equation is applied).
- The structural accelerations of the substructure nodes are used in the calculation of the added-mass, marine-growth mass inertia, and filled-fluid mass inertia loads in the strip-theory solution.
- When coupled to FAST, the hydrodynamic loads computed by HydroDyn are applied to the displaced position of the substructure (i.e., the displaced platform in ElastoDyn and/or the displaced substructure in SubDyn), but are based on wave kinematics at the undisplaced position.

Platform Additional Stiffness and Damping

HydroDyn allows the user to apply additional loads to the platform (in addition to other hydrodynamic terms calculated by HydroDyn), by including a 6x1 static load vector (preload) (**AddF0**), a 6x6 linear restoring matrix (**AddCLin**), a 6x6 linear damping matrix (**AddBLin**), and a 6x6 quadratic drag matrix (**AddBQuad**). These terms can be used, e.g., to model a linearized mooring system, to augment strip-theory members with a linear hydrostatic restoring matrix (see [Section 4.2.8](#)), or to “tune” HydroDyn to match damping to experimental results, such as free-decay tests. While likely most useful for floating systems, these matrices can also be used for fixed-bottom systems; in both cases, the resulting load is applied at the WRP, which when HydroDyn is coupled to FAST, get applied to the platform in ElastoDyn (bypassing SubDyn for fixed-bottom systems).

Fixed-Bottom Substructures

When modeling a fixed-bottom system, the use of a strip-theory (Morison) only model is recommended. When HydroDyn is coupled to FAST, SubDyn is used for the substructure structural dynamics.

All members that are embedded into the seabed (e.g., through piles or suction buckets) must have a joint that is located below the water depth. For example, if the water depth is set to 20 m, and you are modeling a fixed-bottom monopile, then the bottom-most joint needs to have a Z-coordinate such that m . This configuration avoids having HydroDyn apply static pressure loads on the bottom of the structure.

Gravity-based foundations should be modeled such that the lowest joint(s) are located exactly at the prescribed water depth. In other words, the lowest Z-coordinate should be set to m if the water depth is set to 20 m. This configuration allows for static pressure loads to be applied at the bottom of the gravity-base structure.

Floating Platforms

When modeling a floating system, you may use potential-flow theory only, strip-theory (Morison) only, or a hybrid model containing both.

Potential-flow theory based on frequency-to-time-domain transforms is enabled when **PotMod** is set to 1. In this case, you must run WAMIT (or equivalent) in a pre-processing step and HydroDyn will use the WAMIT output files—see [Section 4.2.8](#) for guidance. For a potential-flow-only model, do not create any strip-theory joints or members in the input file. The WAMIT model should account for all of the members in the floating substructure, and Morison's equation is neglected in this case.

For a strip-theory-only model, set **PotMod** to FALSE and create one or more strip-theory members in the input file. Marine growth and nonzero **MSL2SWL** (the offset between still-water and mean-sea level) may only be included in strip-theory-only models.

A hybrid model is formed when both **PotMod** is TRUE and you have defined one or more strip-theory members. The potential-flow model created can consider all of the Morison members in the floating substructure, or just some. Specify whether certain members of the structure are considered in the potential-flow model by setting the **PropPot** flag for each member. The state of the **PropPot** flag for a given member determines which components of the strip-theory equations are applied.

When using either the strip-theory-only or hybrid approaches, filled fluid (flooding or ballasting) may be added to the strip-theory members. Also, the hydrostatic restoring matrix must be entered manually for the strip-theory members—see [Section 4.2.8](#) for guidance.

Please note that current-induced water velocity only induces hydrodynamic loads in HydroDyn through the viscous-drag terms (both distributed and lumped) of strip-theory members. Current is not used in the potential-flow solution. Thus, modeling the effects of current requires the use of a strip-theory-only or hybrid approach.

Undisplaced Position for Floating Systems

The HydroDyn model (geometry, etc.) is defined about the undisplaced position of the substructure. For floating systems, it is important for solution accuracy for the undisplaced position to coincide with the static-equilibrium position in the platform-heave (vertical) direction in the absence of loading from wind, waves, and current. As such, the undisplaced position of the substructure should be defined such that the external buoyancy from displaced water balances with the weight of the system (including the weight of the rotor-nacelle assembly, tower and substructure) and mooring system pretension following the equation below. In this equation, ρ is the water density, g is gravity, V_0 is the undisplaced volume of the floating platform (found in the HydroDyn summary file), m_{Total} is the total mass of the system (found in the ElastoDyn summary), and $T_{Mooring}$ is the mooring system pretension (found in e.g. the MAP summary file). The effects of marine growth, filled fluid (flooding and/or ballasting), and the additional static force (**AddFX0**) should also be taken into consideration in this force balance, where appropriate.

$$\rho g V_0 - m_{Total} g - T_{Mooring} = 0 \quad (4.185)$$

Initial Conditions for Floating Systems

Because the initial conditions used for dynamic simulations typically have an effect on the response statistics during the beginning of the simulation period, an appropriate amount of initial data should be eliminated from consideration in any post-processing analysis. This initial condition solution is more important for floating offshore wind turbines because floating systems typically have long natural periods of the floating substructure and low damping. The appropriate time to eliminate should be chosen such that initial numeric transient effects have sufficiently decayed and the floating substructure has reached a quasi-stationary position. To decrease this initial time in each simulation, it is suggested that the initial conditions of the model (especially blade-pitch angle, rotor speed, substructure surge, and substructure pitch in ElastoDyn) be initialized according to the specific prevalent wind, wave, current, and operational conditions.

Hydrostatic Restoring for Strip-Theory Members of Floating Systems

One notable absence from the list calculations in HydroDyn that make use of substructure motions—see [Section 4.2.8](#)—is that the substructure buoyancy in the strip-theory solution is not recomputed based on the displaced position of the substructure. While the change in buoyancy is likely negligible for fixed-bottom systems, for floating systems modeled using a strip-theory solution, the change in buoyancy with displacement is likely important and should not be neglected. In this latter case, the user should manually calculate the 6x6 linear hydrostatic restoring matrix associated with the strip-theory members and enter this as the additional linear restoring (stiffness) matrix, **AddCLin**. (The static buoyancy of the strip-theory members is automatically calculated and applied within HydroDyn.)

In its most general form, the 6x6 linear hydrostatic restoring matrix of a floating platform is given by the equation below.

$$\text{AddCLin} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \rho g A_0 & \rho g \iint_{A_0} y dA & -\rho g \iint_{A_0} x dA & 0 \\ 0 & 0 & \rho g \iint_{A_0} y dA & \rho g \iint_{A_0} y^2 dA + \rho g V_0 z_b - m_{mg} g z_{mg} - m_f g z_f & -\rho g \iint_{A_0} x y dA & 0 \\ 0 & 0 & -\rho g \iint_{A_0} x dA & -\rho g \iint_{A_0} x y dA & \rho g \iint_{A_0} x^2 dA + \rho g V_0 z_b - m_{mg} g z_{mg} - m_f g z_f & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (4.186)$$

where:

- ρ is water density (kg/m³)
- g is gravity (m/s²)
- A_0 is undisplaced waterplane area of platform (m²)
- V_0 is undisplaced volume of platform (m³)
- (x_b, y_b, z_b) is coordinates of the center of buoyancy of the undisplaced platform (m)
- m_{mg} is total mass of marine growth (kg)
- (x_{mg}, y_{mg}, z_{mg}) is coordinates of the center of mass of the undisplaced marine growth mass (m)
- m_f is total mass of ballasting/flooding (kg)
- (x_f, y_f, z_f) is coordinates of the center of mass of the undisplaced filled fluid (flooding or ballasting) mass (m)

The equation above can be simplified when the floating platform has one or more planes of symmetry. That is, $\iint_{A_0} y dA = 0$, $\iint_{A_0} x y dA = 0$, $y_b = 0$, $y_{mg} = 0$, $y_f = 0$, and if the $x - z$ plane of the platform is a symmetry plane. Likewise, $\iint_{A_0} x dA = 0$, $\iint_{A_0} x y dA = 0$, $x_b = 0$, $x_{mg} = 0$, $x_f = 0$, and if the $y - z$ plane of the platform is a symmetry plane.

The undisplaced coordinates of the center of buoyancy, (x_b, y_b, z_b) , center of marine-growth mass, (x_{mg}, y_{mg}, z_{mg}) , and center of filled-fluid mass, (x_f, y_f, z_f) , are in the global inertial-frame coordinate system. Most of these parameters can be derived from data found in the HydroDyn summary file. While the equation above makes use of several area integrals, the integrals can often be easily estimated by hand for platforms composed of one or more circular members piercing the waterplane (still-water free surface).

The waterplane area of the undisplaced platform, A_0 , affects the hydrostatic load because the displaced volume of the fluid changes with changes in the platform displacement. Similarly, the location of the center of buoyancy of the platform affects the hydrostatic load because its vector position changes with platform displacement and because the cross product of the buoyancy force with the vector position produces hydrostatic moments about the WRP. A_0 , V_0 , and (x_b, y_b, z_b) should be based on the external volume of the platform, including marine-growth thickness. The marine-growth mass and filled-fluid mass also have a direct effect of the hydrostatic restoring because of the moments produced about the WRP.

In classical marine hydrostatics, the effects of body weight are often lumped with the effects of hydrostatics when defining the hydrostatic-restoring matrix; for example, when it is defined in terms of metacentric heights. However, when HydroDyn is coupled to FAST, the body-weight terms (other than the marine-growth and filled-fluid mass within HydroDyn) are automatically accounted for by ElastoDyn, and so, are not included here.

Floating Systems Modeled with Potential Flow

Frequency-dependent hydrodynamic coefficients are needed before running the potential-flow solution in HydroDyn using **PotMod** = 1. An external pre-processing tool should be used to generate the appropriate frequency-dependent hydrodynamic coefficients. The naming in this manual has focused on WAMIT [LN06], but other frequency-domain wave-body interaction panel codes can be used that produce similar data. However, in the end, the WAMIT format is what is expected by HydroDyn.

For the first-order potential-flow solution, HydroDyn requires data from the WAMIT files with *.1*, *.3*, and *.hst* extensions. When creating these files, one should keep in mind:

- The *.1* file must contain the 6×6 added-mass matrix at infinite frequency (period = zero). Additionally, the *.1* file must contain the 6×6 damping matrix over a large range from low frequency to high frequency (the damping should approach zero at both ends of the range). A range of 0.0 to 5.0 rad/s with a discretization of 0.05 rad/s is suggested.
- The *.3* file must contain the first-order wave-excitation (diffraction) loads (3 forces and 3 moments) per unit wave amplitude across frequencies and directions where there is wave energy. A range of 0.0 to 5.0 rad/s with a discretization of 0.05 rad/s is suggested and the direction should be specified across the desired range—the full direction range of (-180 to 180] degrees with a discretization of 10 degrees is suggested. While the *.3* file contains both the magnitude/phase and real/imaginary components of the first-order wave-excitation loads, only the latter are used by HydroDyn.
- The *.hst* file should account for the restoring provided by buoyancy, but not the restoring provided by body mass or moorings. (The hydrostatic file is not frequency dependent.) An important thing to keep in mind is that the pitch and roll restoring of a floating body depends on the vertical distance between the center of buoyancy and center of mass of the body. In WAMIT, the vertical center of gravity (VCG) is used to determine the pitch and roll restoring associated with platform weight, and WAMIT will include these effects in the restoring matrix that it outputs (the *.hst* file). However, the ElastoDyn module of FAST intrinsically accounts for the platform weight's influence on the pitch and roll restoring if the platform weight and center-of-mass location are defined appropriately. To avoid double booking these terms, it is important to neglect these terms in WAMIT. This can be achieved by setting VCG to zero when solving the first-order problem in WAMIT.

The second-order WAMIT files only need to be pre-calculated if a second-order potential-flow option is enabled in HydroDyn. For the second-order mean-drift solution, or for Standing et al.'s extension to Newman's approximation to the mean- and slow-drift solution, HydroDyn requires WAMIT files with *.7*, *.8*, *.9*, *.10d*, *.11d*, or *.12d* extensions. For the second-order full difference-frequency solution of the mean- and slow-drift terms, HydroDyn requires WAMIT files with *.10d*, *.11d*, or *.12d* extension. For the second-order full sum-frequency solution, HydroDyn requires WAMIT files with *.10s*, *.11s*, or *.12s* extensions. When creating any of these files, one should keep in mind:

- The second-order frequency-domain solution is dependent on first-order body motions, whose accuracy is impacted by properly setting the 6×6 rigid-body mass matrix and center of gravity of the complete floating wind system and the 6×6 mooring system restoring matrix. So, while the body center of gravity and mooring stiffness should be zeroed when creating the first-order WAMIT files, they should not be zeroed when creating the second-order WAMIT files. (Thus, obtaining the first-order and second-order WAMIT files requires distinct WAMIT runs.)
- The *.7*, *.8*, and *.9* files contain the diagonal of the difference-frequency QTF, based on the first-order potential-flow solution. The files contain the second-order mean-drift loads (3 forces and 3 moments) per unit wave amplitude squared at each first-order wave frequency and pair of wave directions, across a range of frequencies and a range

of direction pairs. While the .7, .8, and .9 files contains both the magnitude/phase and real/imaginary components of the second-order wave-excitation loads, only the latter are used by HydroDyn.

- The *10d*, *11d*, and *12d*, or *10s*, *11s*, and *12s* files contain the full difference- and sum-frequency QTFs, respectively, based on the first-order or first- plus second-order potential-flow solutions. The files contain the second-order wave-excitation (diffraction) loads (3 forces and 3 moments) per unit wave amplitude squared at each pair of first-order wave frequencies and directions, across a range of frequency and direction pairs. While the *10d*, *11d*, *12d*, *10s*, *11s*, and *12s* files contains both the magnitude/phase and real/imaginary components of the second-order wave-excitation loads, only the latter are used by HydroDyn.
- The frequencies and directions in the WAMIT files do not need to be evenly spaced.
- The discretization of the first set of directions does not need to be the same as the discretization of the second set of directions; however, the matrix of direction pairs must be fully populated (not sparse). Both sets of directions should span across the desired range—the full direction range of $(-180 \text{ to } 180]$ degrees with a discretization of 10 degrees is suggested.
- The frequencies should span the range where there is first-order wave energy and the frequency discretization should be such that the differences and sums between pairs of frequencies span the range where there is second-order wave energy. A range of 0.25 to 2.75 rad/s with a discretization of 0.05 rad/s is suggested.
- Second-order hydrodynamic theory dictates that difference-frequency QTFs are conjugate symmetric between frequency pairs and sum-frequency QTFs are symmetric between frequency pairs. Due to this symmetry, the QTFs (the *10d*, *11d*, or *12d*, *10s*, *11s*, and *12s* files) may be upper triangular, lower triangular, a mix of upper and lower triangular terms, or full; however, after applying the symmetry, the matrix of frequency pairs must be fully populated (not sparse). When an element of the QTF is supplied together with its symmetric pairing, HydroDyn will warn the user if the QTF is not properly symmetric.

Future Work

This list contains features that could be implemented in future releases:

- Enable times-series input motions for Morison members in the standalone HydroDyn (**MorisonInputsMod** = 2).
- Enable tight-coupling to FAST, including linearization.
- Enable wave stretching (**WaveStMod** > 0).
- Enable full support for floating platform force flags.
- Enable joint overlap calculations (**JointOvrtp** = 1).
- Enable auto-generation of all possible output channels (**OutAll** = TRUE).
- Add outputs pertaining to the total hydrodynamic applied loads at nodes along members and at joints.
- Ensure that the output channels are written in the order they are entered.
- Allow for a WAMIT reference point location other than (0,0,0).
- Allow **RdtnDT** to be independent from the FAST simulation time step.
- Add distributed axial viscous-drag loads on tapered members.
- Add rotational inertia terms for fluid-filled members and marine growth.
- Calculate the effective 6x6 added-mass matrix from strip-theory members and place in the HydroDyn summary file.
- Add graphics/animation capability to visualize the substructure geometry and motion, wave elevation, and hydrodynamic loads.
- Add convective fluid acceleration terms.

- Allow for wave directional spreading to include energy spectra that varies with direction (requires changing from the equal-energy method).
- Add higher order regular wave kinematics models for fixed-bottom substructures.
- Add breaking wave-impact loads for fixed-bottom substructures.
- Add floating platform hydro-elastics.
- Add pressure mapping for floating platforms.
- Added automated computation and use of hydrostatic restoring matrix for strip-theory members.

Appendix A: OC4 Semi-submersible Input File

The following is a HydroDyn primary input file for OC4 semi-submersible structure:

```

----- HydroDyn Input File -----
NREL 5.0 MW offshore baseline floating platform HydroDyn input properties for the OC4
↳ Semi-submersible.
False          Echo          - Echo the input file data (flag)
----- ENVIRONMENTAL CONDITIONS -----
"DEFAULT"      WtrDens        - Water density (kg/m^3)
"DEFAULT"      WtrDpth        - Water depth (meters)
"DEFAULT"      MSL2SWL        - Offset between still-water level and mean sea level
↳ (meters) [positive upward; unused when WaveMod = 6; must be zero if PotMod=1 or 2]
----- WAVES -----
          3   WaveMod         - Incident wave kinematics model {0: none=still water, 1:
↳ regular (periodic), 1P#: regular with user-specified phase, 2: JONSWAP/Pierson-
↳ Moskowitz spectrum (irregular), 3: White noise spectrum (irregular), 4: user-defined
↳ spectrum from routine UserWaveSpcrm (irregular), 5: Externally generated wave-
↳ elevation time series, 6: Externally generated full wave-kinematics time series
↳ [option 6 is invalid for PotMod/=0]} (switch)
          0   WaveStMod        - Model for stretching incident wave kinematics to
↳ instantaneous free surface {0: none=no stretching, 1: vertical stretching, 2:
↳ extrapolation stretching, 3: Wheeler stretching} (switch) [unused when WaveMod=0 or
↳ when PotMod/=0]
          4600  WaveTMax        - Analysis time for incident wave calculations (sec) [unused
↳ when WaveMod=0; determines WaveDOmega=2Pi/WaveTMax in the IFFT]
          0.2   WaveDT          - Time step for incident wave calculations (sec) [unused
↳ when WaveMod=0; 0.1<=WaveDT<=1.0 recommended; determines WaveOmegaMax=Pi/WaveDT in the
↳ IFFT]
          1.2646 WaveHs         - Significant wave height of incident waves (meters) [used
↳ only when WaveMod=1, 2, or 3]
          10    WaveTp          - Peak-spectral period of incident waves (sec)
↳ [used only when WaveMod=1 or 2]
"DEFAULT"      WavePkShp        - Peak-shape parameter of incident wave spectrum (-) or
↳ DEFAULT (string) [used only when WaveMod=2; use 1.0 for Pierson-Moskowitz]
          0.314159 WvLowCOff     - Low cut-off frequency or lower frequency limit of the
↳ wave spectrum beyond which the wave spectrum is zeroed (rad/s) [unused when WaveMod=0,
↳ 1, or 6]
          1.570796 WvHiCOff     - High cut-off frequency or upper frequency limit of the
↳ wave spectrum beyond which the wave spectrum is zeroed (rad/s) [unused when WaveMod=0,
↳ 1, or 6]
          0     WaveDir         - Incident wave propagation heading direction

```

(continues on next page)

(continued from previous page)

```

→      (degrees) [unused when WaveMod=0 or 6]
0      WaveDirMod      - Directional spreading function {0: none, 1: COS2S}
→      (-)             [only used when WaveMod=2,3, or 4]
1      WaveDirSpread   - Wave direction spreading coefficient ( > 0 )
→      (-)             [only used when WaveMod=2,3, or 4 and WaveDirMod=1]
1      WaveNDir        - Number of wave directions
→      (-)             [only used when WaveMod=2,3, or 4 and WaveDirMod=1; odd number
→only]
0      WaveDirRange    - Range of wave directions (full range: WaveDir +/- 1/
→2*WaveDirRange) (degrees) [only used when WaveMod=2,3,or 4 and WaveDirMod=1]
123456789 WaveSeed(1)  - First random seed of incident waves [-2147483648 to
→2147483647] (-)       [unused when WaveMod=0, 5, or 6]
1011121314 WaveSeed(2) - Second random seed of incident waves [-2147483648 to
→2147483647] (-)       [unused when WaveMod=0, 5, or 6]
FALSE      WaveNDamp    - Flag for normally distributed amplitudes
→      (flag)          [only used when WaveMod=2, 3, or 4]
""         WvKinFile    - Root name of externally generated wave data file(s)
→      (quoted string) [used only when WaveMod=5 or 6]
1      NWaveElev       - Number of points where the incident wave elevations can
→be computed (-)       [maximum of 9 output locations]
0      WaveElevxi      - List of xi-coordinates for points where the incident
→wave elevations can be output (meters) [NWaveElev points, separated by commas or white
→space; unused if NWaveElev = 0]
0      WaveElevyi      - List of yi-coordinates for points where the incident
→wave elevations can be output (meters) [NWaveElev points, separated by commas or white
→space; unused if NWaveElev = 0]
----- 2ND-ORDER WAVES ----- [unused
→with WaveMod=0 or 6]
FALSE      WvDiffQTF   - Full difference-frequency 2nd-order wave kinematics
→(flag)
FALSE      WvSumQTF    - Full summation-frequency 2nd-order wave kinematics
→(flag)
0      WvLowCoffD      - Low frequency cutoff used in the difference-
→frequencies (rad/s) [Only used with a difference-frequency method]
1.256637 WvHiCoffD     - High frequency cutoff used in the difference-
→frequencies (rad/s) [Only used with a difference-frequency method]
0.618319 WvLowCoffS    - Low frequency cutoff used in the summation-
→frequencies (rad/s) [Only used with a summation-frequency method]
3.141593 WvHiCoffS     - High frequency cutoff used in the summation-
→frequencies (rad/s) [Only used with a summation-frequency method]
----- CURRENT ----- [unused
→with WaveMod=6]
0      CurrMod         - Current profile model {0: none=no current, 1: standard,
→2: user-defined from routine UserCurrent} (switch)
0      CurrSSV0        - Sub-surface current velocity at still water level (m/
→s) [used only when CurrMod=1]
"DEFAULT" CurrSSDir    - Sub-surface current heading direction (degrees) or
→DEFAULT (string) [used only when CurrMod=1]
20     CurrNSRef       - Near-surface current reference depth
→(meters) [used only when CurrMod=1]
0      CurrNSV0        - Near-surface current velocity at still water level (m/
→s) [used only when CurrMod=1]

```

(continues on next page)

(continued from previous page)

```

0 CurrNSDir - Near-surface current heading direction
↪(degrees) [used only when CurrMod=1]
0 CurrDIV - Depth-independent current velocity (m/
↪s) [used only when CurrMod=1]
0 CurrDIDir - Depth-independent current heading direction
↪(degrees) [used only when CurrMod=1]
----- FLOATING PLATFORM ----- [unused]
↪with WaveMod=6]
1 PotMod - Potential-flow model {0: none=no potential flow, 1:
↪frequency-to-time-domain transforms based on WAMIT output, 2: fluid-impulse theory
↪(FIT)} (switch)
"HydroData/marin_semi" PotFile - Root name of potential-flow model data; WAMIT
↪output files containing the linear, nondimensionalized, hydrostatic restoring matrix (.
↪hst), frequency-dependent hydrodynamic added mass matrix and damping matrix (.1), and
↪frequency- and direction-dependent wave excitation force vector per unit wave
↪amplitude (.3) (quoted string) [MAKE SURE THE FREQUENCIES INHERENT IN THESE WAMIT
↪FILES SPAN THE PHYSICALLY-SIGNIFICANT RANGE OF FREQUENCIES FOR THE GIVEN PLATFORM;
↪THEY MUST CONTAIN THE ZERO- AND INFINITE-FREQUENCY LIMITS!]
1 WAMITULEN - Characteristic body length scale used to
↪redimensionalize WAMIT output (meters) [only used when PotMod=1]
13917 PtfmVol0 - Displaced volume of water when the platform is in its
↪undisplaced position (m^3) [only used when PotMod=1; USE THE SAME VALUE COMPUTED BY
↪WAMIT AS OUTPUT IN THE .OUT FILE!]
0 PtfmCOBxt - The xt offset of the center of buoyancy (COB) from the
↪platform reference point (meters) [only used when PotMod=1]
0 PtfmCOByt - The yt offset of the center of buoyancy (COB) from the
↪platform reference point (meters) [only used when PotMod=1]
1 ExctnMod - Wave Excitation model {0: None, 1: DFT, 2: state-space}
↪(switch) [only used when PotMod=1; STATE-SPACE REQUIRES *.ssxctn INPUT FILE]
1 RdtnMod - Radiation memory-effect model {0: no memory-effect
↪calculation, 1: convolution, 2: state-space} (switch) [only used when PotMod=1; STATE-
↪SPACE REQUIRES *.ss INPUT FILE]
60 RdtnTMax - Analysis time for wave radiation kernel calculations
↪(sec) [only used when PotMod=1 and RdtnMod>0; determines RdtnDOmega=Pi/RdtnTMax in the
↪cosine transform; MAKE SURE THIS IS LONG ENOUGH FOR THE RADIATION IMPULSE RESPONSE
↪FUNCTIONS TO DECAY TO NEAR-ZERO FOR THE GIVEN PLATFORM!]
"DEFAULT" RdtnDT - Time step for wave radiation kernel calculations (sec)
↪[only used when PotMod=1 and RdtnMod=1; DT<=RdtnDT<=0.1 recommended; determines
↪RdtnOmegaMax=Pi/RdtnDT in the cosine transform]
----- 2ND-ORDER FLOATING PLATFORM FORCES ----- [unused]
↪with WaveMod=0 or 6, or PotMod=0 or 2]
0 MnDrift - Mean-drift 2nd-order forces computed
↪{0: None; [7, 8, 9, 10, 11, or 12]: WAMIT file to use} [Only one of
↪MnDrift, NewmanApp, or DiffQTF can be non-zero]
0 NewmanApp - Mean- and slow-drift 2nd-order forces computed with
↪Newman's approximation {0: None; [7, 8, 9, 10, 11, or 12]: WAMIT file to use} [Only
↪one of MnDrift, NewmanApp, or DiffQTF can be non-zero. Used only when WaveDirMod=0]
0 DiffQTF - Full difference-frequency 2nd-order forces computed
↪with full QTF {0: None; [10, 11, or 12]: WAMIT file to use} [Only
↪one of MnDrift, NewmanApp, or DiffQTF can be non-zero]
0 SumQTF - Full summation -frequency 2nd-order forces computed
↪with full QTF {0: None; [10, 11, or 12]: WAMIT file to use}

```

(continues on next page)

(continued from previous page)

```

----- FLOATING PLATFORM FORCE FLAGS ----- [unused]
↳with WaveMod=6]
True          PtfmSgF      - Platform horizontal surge translation force (flag) or
↳DEFAULT
True          PtfmSwF      - Platform horizontal sway translation force (flag) or
↳DEFAULT
True          PtfmHvF      - Platform vertical heave translation force (flag) or
↳DEFAULT
True          PtfmRF        - Platform roll tilt rotation force (flag) or DEFAULT
True          PtfmPF        - Platform pitch tilt rotation force (flag) or DEFAULT
True          PtfmYF        - Platform yaw rotation force (flag) or DEFAULT
----- PLATFORM ADDITIONAL STIFFNESS AND DAMPING -----
0          0          0          0          0          0
↳AddF0      - Additional preload (N, N-m)
0          0          0          0          0          0
↳AddCLin    - Additional linear stiffness (N/m, N/rad, N-m/m, N-m/rad)
0          0          0          0          0          0
0          0          0          0          0          0
0          0          0          1451298897  0          0
0          0          0          0          1451298897  0
0          0          0          0          0          0
0          0          0          0          0          0
↳AddBLin    - Additional linear damping(N/(m/s), N/(rad/s), N-m/(m/s), N-m/(rad/s))
0          0          0          0          0          0
0          0          0          0          0          0
0          0          0          0          0          0
0          0          0          0          0          0
0          0          0          0          0          0
0          0          0          0          0          0
↳AddBQuad   - Additional quadratic drag(N/(m/s)^2, N/(rad/s)^2, N-m(m/s)^2, N-m/(rad/s)^2)
0          0          0          0          0          0
0          0          0          0          0          0
0          0          0          0          0          0
0          0          0          0          0          0
0          0          0          0          0          0
----- AXIAL COEFFICIENTS -----
2  NACoef      - Number of axial coefficients (-)
AxCoefID  AxCd    AxCa    AxCp
(-)      (-)      (-)      (-)
1      0.00    0.00    1.00
2      9.60    0.00    1.00
----- MEMBER JOINTS -----
44  NJoints      - Number of joints (-) [must be exactly 0 or at least
↳2]
JointID  Jointxi  Jointyi  Jointzi  JointAxID  JointOvrldp [JointOvrldp= 0: do
↳nothing at joint, 1: eliminate overlaps by calculating super member]
(-)      (m)      (m)      (m)      (-)      (switch)
1      0.00000    0.00000   -20.00000    1          0
2      0.00000    0.00000    10.00000    1          0
3     14.43376    25.00000   -14.00000    1          0
4     14.43376    25.00000    12.00000    1          0
5    -28.86751     0.00000   -14.00000    1          0

```

(continues on next page)

(continued from previous page)

6	-28.86751	0.00000	12.00000	1	0	
7	14.43376	-25.00000	-14.00000	1	0	
8	14.43376	-25.00000	12.00000	1	0	
9	14.43375	25.00000	-20.00000	2	0	
10	-28.86750	0.00000	-20.00000	2	0	
11	14.43375	-25.00000	-20.00000	2	0	
12	9.23760	22.00000	10.00000	1	0	
13	-23.67130	3.00000	10.00000	1	0	
14	-23.67130	-3.00000	10.00000	1	0	
15	9.23760	-22.00000	10.00000	1	0	
16	14.43375	-19.00000	10.00000	1	0	
17	14.43375	19.00000	10.00000	1	0	
18	4.04145	19.00000	-17.00000	1	0	
19	-18.47520	6.00000	-17.00000	1	0	
20	-18.47520	-6.00000	-17.00000	1	0	
21	4.04145	-19.00000	-17.00000	1	0	
22	14.43375	-13.00000	-17.00000	1	0	
23	14.43375	13.00000	-17.00000	1	0	
24	1.62500	2.81500	10.00000	1	0	
25	11.43376	19.80385	10.00000	1	0	
26	-3.25000	0.00000	10.00000	1	0	
27	-22.87000	0.00000	10.00000	1	0	
28	1.62500	-2.81500	10.00000	1	0	
29	11.43376	-19.80385	10.00000	1	0	
30	1.62500	2.81500	-17.00000	1	0	
31	8.43376	14.60770	-17.00000	1	0	
32	-3.25000	0.00000	-17.00000	1	0	
33	-16.87000	0.00000	-17.00000	1	0	
34	1.62500	-2.81500	-17.00000	1	0	
35	8.43376	-14.60770	-17.00000	1	0	
36	1.62500	2.81500	-16.20000	1	0	
37	11.43376	19.80385	9.13000	1	0	
38	-3.25000	0.00000	-16.20000	1	0	
39	-22.87000	0.00000	9.13000	1	0	
40	1.62500	-2.81500	-16.20000	1	0	
41	11.43376	-19.80385	9.13000	1	0	
42	14.43376	25.00000	-19.94000	1	0	
43	-28.86751	0.00000	-19.94000	1	0	
44	14.43376	-25.00000	-19.94000	1	0	
----- MEMBER CROSS-SECTION PROPERTIES -----						
4 NPropSets - Number of member property sets (-)						
PropSetID	PropD	PropThck				
(-)	(m)	(m)				
1	6.50000	0.03000	! Main Column			
2	12.00000	0.06000	! Upper Columns			
3	24.00000	0.06000	! Base Columns			
4	1.60000	0.01750	! Pontoons			
----- SIMPLE HYDRODYNAMIC COEFFICIENTS (model 1) -----						
SimplCd	SimplCdMG	SimplCa	SimplCaMG	SimplCp	SimplCpMG	SimplAxCa
↪SimplAxCaMG	SimplAxCp	SimplAxCpMG				
(-)	(-)	(-)	(-)	(-)	(-)	(-)
↪ (-)	(-)	(-)				

(continues on next page)

(continued from previous page)

```

0.00      0.00      0.00      0.00      1.00      1.00      0.00
→ 0.00      1.00      1.00
----- DEPTH-BASED HYDRODYNAMIC COEFFICIENTS (model 2) -----
      0 NCoefDpth      - Number of depth-dependent coefficients (-)
Dpth      DpthCd      DpthCdMG      DpthCa      DpthCaMG      DpthCp      DpthCpMG      DpthAxCa
→ DpthAxCaMG      DpthAxCp      DpthAxCpMG
(m)      (-)      (-)      (-)      (-)      (-)      (-)      (-)      (-)
→      (-)      (-)
----- MEMBER-BASED HYDRODYNAMIC COEFFICIENTS (model 3) -----
      25 NCoefMembers      - Number of member-based coefficients (-)
MemberID      MemberCd1      MemberCd2      MemberCdMG1      MemberCdMG2      MemberCa1
→ MemberCa2      MemberCaMG1      MemberCaMG2      MemberCp1      MemberCp2      MemberCpMG1
→ MemberCpMG2      MemberAxCa1      MemberAxCa2      MemberAxCaMG1      MemberAxCaMG2      MemberAxCp1
→ MemberAxCp2      MemberAxCpMG1      MemberAxCpMG2
(-)      (-)      (-)      (-)      (-)      (-)      (-)
→      (-)      (-)      (-)      (-)      (-)      (-)      (-)
→      (-)      (-)      (-)      (-)      (-)      (-)      (-)
→      (-)      (-)      ! Main Column
1      0.56      0.56      0.00      0.00      0.00      0.00
→      0.00      0.00      0.00      0.00      0.00      0.00
→      0.00      0.00      0.00      0.00      0.00      0.00
→ 0.00      0.00      ! Upper Column 1
2      0.61      0.61      0.00      0.00      0.00      0.00
→      0.00      0.00      0.00      0.00      0.00      0.00
→      0.00      0.00      0.00      0.00      0.00      0.00
→ 0.00      0.00      ! Upper Column 2
3      0.61      0.61      0.00      0.00      0.00      0.00
→      0.00      0.00      0.00      0.00      0.00      0.00
→      0.00      0.00      0.00      0.00      0.00      0.00
→ 0.00      0.00      ! Upper Column 3
4      0.61      0.61      0.00      0.00      0.00      0.00
→      0.00      0.00      0.00      0.00      0.00      0.00
→      0.00      0.00      0.00      0.00      0.00      0.00
→ 0.00      0.00      ! Base Column 1
5      0.68      0.68      0.00      0.00      0.00      0.00
→      0.00      0.00      0.00      0.00      0.00      0.00
→      0.00      0.00      0.00      0.00      0.00      0.00
→ 0.00      0.00      ! Base Column 2
6      0.68      0.68      0.00      0.00      0.00      0.00
→      0.00      0.00      0.00      0.00      0.00      0.00
→      0.00      0.00      0.00      0.00      0.00      0.00
→ 0.00      0.00      ! Base Column 3
7      0.68      0.68      0.00      0.00      0.00      0.00
→      0.00      0.00      0.00      0.00      0.00      0.00
→      0.00      0.00      0.00      0.00      0.00      0.00
→ 0.00      0.00      ! Base column cap 1
23      0.68      0.68      0.00      0.00      0.00      0.00
→      0.00      0.00      0.00      0.00      0.00      0.00
→      0.00      0.00      0.00      0.00      0.00      0.00
→ 0.00      0.00      ! Base column cap 2
24      0.68      0.68      0.00      0.00      0.00      0.00
→      0.00      0.00      0.00      0.00      0.00      0.00

```

(continues on next page)

(continued from previous page)

```

→ 0.00 0.00 0.00 0.00 0.00 0.00
→ 0.00 0.00 ! Base column cap 3
25 0.68 0.68 0.00 0.00 0.00 0.00
→ 0.00 0.00 0.00 0.00 0.00 0.00
→ 0.00 0.00 0.00 0.00 0.00 0.00
→ 0.00 0.00 ! Delta Pontoon, Upper 1
8 0.63 0.63 0.00 0.00 0.00 0.00
→ 0.00 0.00 0.00 0.00 0.00 0.00
→ 0.00 0.00 0.00 0.00 0.00 0.00
→ 0.00 0.00 ! Delta Pontoon, Upper 2
9 0.63 0.63 0.00 0.00 0.00 0.00
→ 0.00 0.00 0.00 0.00 0.00 0.00
→ 0.00 0.00 0.00 0.00 0.00 0.00
→ 0.00 0.00 ! Delta Pontoon, Upper 3
10 0.63 0.63 0.00 0.00 0.00 0.00
→ 0.00 0.00 0.00 0.00 0.00 0.00
→ 0.00 0.00 0.00 0.00 0.00 0.00
→ 0.00 0.00 ! Delta Pontoon, Lower 1
11 0.63 0.63 0.00 0.00 0.00 0.00
→ 0.00 0.00 0.00 0.00 0.00 0.00
→ 0.00 0.00 0.00 0.00 0.00 0.00
→ 0.00 0.00 ! Delta Pontoon, Lower 2
12 0.63 0.63 0.00 0.00 0.00 0.00
→ 0.00 0.00 0.00 0.00 0.00 0.00
→ 0.00 0.00 0.00 0.00 0.00 0.00
→ 0.00 0.00 ! Delta Pontoon, Lower 3
13 0.63 0.63 0.00 0.00 0.00 0.00
→ 0.00 0.00 0.00 0.00 0.00 0.00
→ 0.00 0.00 0.00 0.00 0.00 0.00
→ 0.00 0.00 ! Y Pontoon, Upper 1
14 0.63 0.63 0.00 0.00 0.00 0.00
→ 0.00 0.00 0.00 0.00 0.00 0.00
→ 0.00 0.00 0.00 0.00 0.00 0.00
→ 0.00 0.00 ! Y Pontoon, Upper 2
15 0.63 0.63 0.00 0.00 0.00 0.00
→ 0.00 0.00 0.00 0.00 0.00 0.00
→ 0.00 0.00 0.00 0.00 0.00 0.00
→ 0.00 0.00 ! Y Pontoon, Upper 3
16 0.63 0.63 0.00 0.00 0.00 0.00
→ 0.00 0.00 0.00 0.00 0.00 0.00
→ 0.00 0.00 0.00 0.00 0.00 0.00
→ 0.00 0.00 ! Y Pontoon, Lower 1
17 0.63 0.63 0.00 0.00 0.00 0.00
→ 0.00 0.00 0.00 0.00 0.00 0.00
→ 0.00 0.00 0.00 0.00 0.00 0.00
→ 0.00 0.00 ! Y Pontoon, Lower 2
18 0.63 0.63 0.00 0.00 0.00 0.00
→ 0.00 0.00 0.00 0.00 0.00 0.00
→ 0.00 0.00 0.00 0.00 0.00 0.00
→ 0.00 0.00 ! Y Pontoon, Lower 3
19 0.63 0.63 0.00 0.00 0.00 0.00
→ 0.00 0.00 0.00 0.00 0.00 0.00

```

(continues on next page)

(continued from previous page)

```

→ 0.00      0.00      0.00      0.00      0.00      0.00
→ 0.00      0.00      ! Cross Brace 1
20 0.63      0.63      0.00      0.00      0.00      0.00
→ 0.00      0.00      0.00      0.00      0.00      0.00
→ 0.00      0.00      0.00      0.00      0.00      0.00
→ 0.00      0.00      ! Cross Brace 2
21 0.63      0.63      0.00      0.00      0.00      0.00
→ 0.00      0.00      0.00      0.00      0.00      0.00
→ 0.00      0.00      0.00      0.00      0.00      0.00
→ 0.00      0.00      ! Cross Brace 3
22 0.63      0.63      0.00      0.00      0.00      0.00
→ 0.00      0.00      0.00      0.00      0.00      0.00
→ 0.00      0.00      0.00      0.00      0.00      0.00
→ 0.00      0.00
----- MEMBERS -----
      25 NMembers      - Number of members (-)
MemberID MJointID1 MJointID2 MPropSetID1 MPropSetID2 MDivSize MCoefMod PropPot
→ [MCoefMod=1: use simple coeff table, 2: use depth-based coeff table, 3: use member-
→ based coeff table] [ PropPot/=0 if member is modeled with potential-flow theory]
(-)      (-)      (-)      (-)      (-)      (m)      (switch)      (flag)
1        1        2        1        1        1.0000    3        TRUE
→ ! Main Column
2        3        4        2        2        1.0000    3        TRUE
→ ! Upper Column 1
3        5        6        2        2        1.0000    3        TRUE
→ ! Upper Column 2
4        7        8        2        2        1.0000    3        TRUE
→ ! Upper Column 3
5        42       3        3        3        1.0000    3        TRUE
→ ! Base Column 1
6        43       5        3        3        1.0000    3        TRUE
→ ! Base Column 2
7        44       7        3        3        1.0000    3        TRUE
→ ! Base Column 3
23       9        42       3        3        1.0000    3        TRUE
→ ! Base column cap 1
24       10       43       3        3        1.0000    3        TRUE
→ ! Base column cap 2
25       11       44       3        3        1.0000    3        TRUE
→ ! Base column cap 3
8        12       13       4        4        1.0000    3        TRUE
→ ! Delta Pontoon, Upper 1
9        14       15       4        4        1.0000    3        TRUE
→ ! Delta Pontoon, Upper 2
10       16       17       4        4        1.0000    3        TRUE
→ ! Delta Pontoon, Upper 3
11       18       19       4        4        1.0000    3        TRUE
→ ! Delta Pontoon, Lower 1
12       20       21       4        4        1.0000    3        TRUE
→ ! Delta Pontoon, Lower 2
13       22       23       4        4        1.0000    3        TRUE
→ ! Delta Pontoon, Lower 3

```

(continues on next page)

(continued from previous page)

```

14      24      25      4      4      1.0000      3      TRUE      ␣
↪      ! Y Pontoon, Upper 1
15      26      27      4      4      1.0000      3      TRUE      ␣
↪      ! Y Pontoon, Upper 2
16      28      29      4      4      1.0000      3      TRUE      ␣
↪      ! Y Pontoon, Upper 3
17      30      31      4      4      1.0000      3      TRUE      ␣
↪      ! Y Pontoon, Lower 1
18      32      33      4      4      1.0000      3      TRUE      ␣
↪      ! Y Pontoon, Lower 2
19      34      35      4      4      1.0000      3      TRUE      ␣
↪      ! Y Pontoon, Lower 3
20      36      37      4      4      1.0000      3      TRUE      ␣
↪      ! Cross Brace 1
21      38      39      4      4      1.0000      3      TRUE      ␣
↪      ! Cross Brace 2
22      40      41      4      4      1.0000      3      TRUE      ␣
↪      ! Cross Brace 3
----- FILLED MEMBERS -----
      2      NFillGroups      - Number of filled member groups (-) [If FillDens =
↪DEFAULT, then FillDens = WtrDens; FillFSLoc is related to MSL2SWL]
FillNumM FillMList      FillFSLoc      FillDens
(-)      (-)      (m)      (kg/m^3)
3      2      3      4      -6.17      1025
3      5      6      7      -14.89      1025
----- MARINE GROWTH -----
      0      NMGDepths      - Number of marine-growth depths specified (-)
MGDpth      MGDthck      MGDens
(m)      (m)      (kg/m^3)
----- MEMBER OUTPUT LIST -----
      0      NMOutputs      - Number of member outputs (-) [must be < 10]
MemberID      NOutLoc      NodeLocs [NOutLoc < 10; node locations are normalized distance
↪from the start of the member, and must be >=0 and <= 1] [unused if NMOutputs=0]
(-)      (-)      (-)
----- JOINT OUTPUT LIST -----
      0      NJOutputs      - Number of joint outputs [Must be < 10]
0      JOutLst      - List of JointIDs which are to be output (-)[unused if
↪NJOutputs=0]
----- OUTPUT -----
True      HDSum      - Output a summary file [flag]
False      OutAll      - Output all user-specified member and joint loads (only
↪at each member end, not interior locations) [flag]
      1      OutSwitch      - Output requested channels to: [1=Hydrodyn.out,
↪2=GlueCode.out, 3=both files]
"ES11.4e2"      OutFmt      - Output format for numerical results (quoted string)
↪[not checked for validity!]
"A11"      OutSFmt      - Output format for header strings (quoted string) [not
↪checked for validity!]
----- OUTPUT CHANNELS -----
"Wave1Elev"      - Wave elevation at the platform reference point (0, 0)
END of output channels and end of file. (the word "END" must appear in the first 3
↪columns of this line)

```

Appendix B: OC4 Semi-submersible Input File

The following is a HydroDyn driver input file for OC4 semi-submersible structure:

```

HydroDyn Driver file for OC4 Semi-submersible.
Compatible with HydroDyn v2.03.*
TRUE          Echo          - Echo the input file data (flag)
----- ENVIRONMENTAL CONDITIONS -----
9.80665       Gravity       - Gravity (m/s^2)
1025          WtrDens       - Water density (kg/m^3)
200           WtrDpth       - Water depth (meters)
0             MSL2SWL       - Offset between still-water level and mean sea_
↪level (meters) [positive upward; unused when WaveMod = 6; must be zero if PotMod=1 or_
↪2]
----- HYDRODYN -----
"./OC4Semi.dat" HDInputFile - Primary HydroDyn input file name (quoted string)
"./OC4Semi"     OutRootName - The name which prefixes all HydroDyn generated_
↪files (quoted string)
1              NSteps       - Number of time steps in the simulations (-)
0.025         TimeInterval - TimeInterval for the simulation (sec)
----- WAMIT INPUTS -----
1             WAMITInputsMod - Inputs model {0: all inputs are zero for every_
↪timestep, 1: steadystate inputs, 2: read inputs from a file (InputsFile)} (switch)
""           WAMITInputsFile - Name of the inputs file if InputsMod = 2 (quoted_
↪string)
----- WAMIT STEADY STATE INPUTS -----
1.0  2.0  3.0  4.0  5.0  6.0  uWAMITInSteady - input displacements and_
↪rotations at the platform reference point (m, rads)
7.0  8.0  9.0  10.0  11.0  12.0  uDotWAMITInSteady - input translational and_
↪rotational velocities at the platform reference point (m/s, rads/s)
13.0  14.0  15.0  16.0  17.0  18.0  uDotDotWAMITInSteady - input translational and_
↪rotational acccelerations at the platform reference point (m/s^2, rads/s^2)
----- MORISON INPUTS -----
0             MorisonInputsMod - Inputs model {0: all inputs are zero for every_
↪timestep, 1: steadystate inputs, 2: read inputs from a file (InputsFile)} (switch)
" "          MorisonInputsFile - Name of the inputs file if InputsMod = 2 (quoted_
↪string)
----- MORISON STEADY STATE INPUTS -----
1.0  2.0  3.0  4.0  5.0  6.0  uMorisonInSteady - input displacements and_
↪rotations for the morison elements (m, rads)
7.0  8.0  9.0  10.0  11.0  12.0  uDotMorisonInSteady - input translational and_
↪rotational velocities for the morison elements (m/s, rads/s)
13.0  14.0  15.0  16.0  17.0  18.0  uDotDotMorisonInSteady - input translational and_
↪rotational acccelerations for the morison elements (m/s^2, rads/s^2)
----- Waves multipoint elevation output -----
TRUE          WaveElevSeriesFlag - T/F flag to calculate the wave elevation field_
↪(for movies)
5.0  5.0      WaveElevDX WaveElevDY - WaveElevSeries spacing -- WaveElevDX WaveElevDY
3  3          WaveElevNX WaveElevNY - WaveElevSeries points -- WaveElevNX WaveElevNY
END of driver input file

```

Appendix C. List of Output Channels

This is a list of all possible output parameters for the HydroDyn module. The names are grouped by meaning, but can be ordered in the OUTPUT CHANNELS section of the HydroDyn input file as you see fit. MN, refers to output node of output member , where is a number in the range [1,9] and corresponds to row in the MEMBER OUTPUT LIST table and is a number in the range [1,9] and corresponds to location in the **NodeLocs** list of that table entry. J refers to output joint , where is a number in the range [1,9] and corresponds to row in the JOINT OUTPUT LIST table. B refers to body , where is a number in the range [1,9]. Setting > NBody yields invalid output; if NBody > 9, only the first 9 bodies can be output. Wave refers to point where wave elevations can be output, where is a number in the range [1,9]. Setting > NWaveElev yields invalid output. All outputs are in the global inertial-frame coordinate.

Channel Name(s)	Units	Description
Wave and Current Kinematics		
WaveElev	(m)	Total (first- plus second-order) wave elevations (up to 9 designated locations)
WaveElv1	(m)	First-order wave elevations (up to 9 designated locations)
WaveElv2	(m)	Second-order wave elevations (up to 9 designated locations)
MNVxi, MNVyi, MNVzi	(m/s), (m/s), (m/s)	Total (first- plus second-order) fluid particle velocities at MN
MNAxi, MNAyi, MNAzi	(m/s ²), (m/s ²), (m/s ²)	Total (first- plus second-order) fluid particle accelerations at MN
MNDynP	(Pa)	Total (first- plus second-order) fluid particle dynamic pressure at MN
JVxi, JVyi, JVzi	(m/s), (m/s), (m/s)	Total (first- plus second-order) fluid particle velocities at J
JAXi, JAYi, JAZi	(m/s ²), (m/s ²), (m/s ²)	Total (first- plus second-order) fluid particle accelerations at J
JDynP	(Pa)	Total (first- plus second-order) fluid particle dynamic pressure at J
Total and Additional Loads		
BAddFxi, BAddFyi, BAddFzi, BAddMxi, BAddMyi, BAddMzi	(N), (N), (N), (N·m), (N·m), (N·m)	Loads due to additional preload, stiffness, and damping at B
HydroFxi, HydroFyi, HydroFzi, HydroMxi, HydroMyi, HydroMzi	(N), (N), (N), (N·m), (N·m), (N·m)	Total integrated hydrodynamic loads from both potential flow and strip theory at (0,0,0)
Loads from Potential-Flow Solution		
BWvsFxi, BWvsFyi, BWvsFzi, BWvsMxi, BWvsMyi, BWvsMzi	(N), (N), (N), (N·m), (N·m), (N·m)	Total (first- plus second-order) wave-excitation loads from diffraction at B
BWvsF1xi, BWvsF1yi, BWvsF1zi, BWvsM1xi, BWvsM1yi, BWvsM1zi	(N), (N), (N), (N·m), (N·m), (N·m)	First-order wave-excitation loads from diffraction at B
BWvsF2xi, BWvsF2yi, BWvsF2zi, BWvsM2xi, BWvsM2yi, BWvsM2zi	(N), (N), (N), (N·m), (N·m), (N·m)	Second-order wave-excitation loads from diffraction at B

continues on next page

Table 4.10 – continued from previous page

Channel Name(s)	Units	Description
BHdSFxi, BHdSFyi, BHdSFzi, BHdSMxi, BHdSMyi, BHdSMzi	(N), (N), (N), (N·m), (N·m), (N·m)	Hydrostatic loads at B
BRdtFxi, BRdtFyi, BRdtFzi, BRdt-Mxi, BRdtMyi, BRdtMzi	(N), (N), (N), (N·m), (N·m), (N·m)	Wave-radiation loads at B
Structural Motions		
BSurge, BSway, BHeave, BRoll, BPitch, BYaw	(m), (m), (m), (rad), (rad), (rad)	Displacements and rotations at B
BTVxi, BTVyi, BTVzi, BRVxi, BRVyi, BRVzi	(m/s), (m/s), (m/s), (rad/s), (rad/s), (rad/s)	Translational and rotational velocities at B
BTAXi, BTAYi, BTAZi, BRAxi, BRAyi, BRAzi	(m/s ²), (m/s ²), (m/s ²), (rad/s ²), (rad/s ²), (rad/s ²)	Translational and rotational accelerations at B
MNSTVxi, MNSTVyi, MNSTVzi	(m/s), (m/s), (m/s)	Structural translational velocities at MN
MNSTAXi, MNSTAYi, MNSTAZi	(m/s ²), (m/s ²), (m/s ²)	Structural translational accelerations at MN
JSTVxi, JSTVyi, JSTVzi	(m/s), (m/s), (m/s)	Structural translational velocities at J
JSTAXi, JSTAYi, JSTAZi	(m/s ²), (m/s ²), (m/s ²)	Structural translational accelerations at J
Distributed Loads (Per Unit Length) on Members		
MNFDxi, MNFDyi, MNFDzi	(N/m), (N/m), (N/m)	Viscous-drag forces at MN
MNFIxi, MNFIyi, MNFIzi	(N/m), (N/m), (N/m)	Fluid-inertia forces at MN
MNFBxi, MNFByi, MNFBzi, MNMBxi, MNMByi, MNMBzi	(N/m), (N/m), (N/m), (N·m/m), (N·m/m), (N·m/m)	Buoyancy loads at MN
MNFBFxi, MNFBFyi, MNFBFzi, MNMBFxi, MNMBFyi, MNMBFzi	(N/m), (N/m), (N/m), (N·m/m), (N·m/m), (N·m/m)	Negative buoyancy loads due to flooding/ballasting at MN
MNFMGxi, MNFMGyi, MNFMGzi, MNMMGxi, MNMMGyi, MNMMGzi	(N/m), (N/m), (N/m), (N·m/m), (N·m/m), (N·m/m)	Loads due to marine growth weight at MN
MNFAMxi, MNFAMyi, MNFAMzi	(N/m), (N/m), (N/m)	Hydrodynamic added-mass forces at MN
MNFAGxi, MNFAGyi, MNFAGzi, MNMAGxi, MNMAGyi, MNMAGzi	(N/m), (N/m), (N/m), (N·m/m), (N·m/m), (N·m/m)	Marine growth mass inertia loads at MN
MNFAFxi, MNFAFyi, MNFAFzi, MNMAFxi, MNMAFyi, MNMAFzi	(N/m), (N/m), (N/m), (N·m/m), (N·m/m), (N·m/m)	Flooding/ballasting mass inertia loads at MN
Lumped Loads at Joints		
JFDxi, JFDyi, JFDzi	(N), (N), (N)	Viscous-drag forces at J
JFIxi, JFIyi, JFIzi	(N), (N), (N)	Fluid-inertia forces at J
JFBxi, JFByi, JFBzi, JMBxi, JMByi, JMBzi	(N), (N), (N), (N·m), (N·m), (N·m)	Buoyancy loads at J
JFBFxi, JFBFyi, JFBFzi, JMBFxi, JMBFyi, JMBFzi	(N), (N), (N), (N·m), (N·m), (N·m)	Negative buoyancy loads due to flooding/ballasting at J
JFMGxi, JFMGyi, JFMGzi	(N), (N), (N)	Forces due to marine growth weight at J
JFAMxi, JFAMyi, JFAMzi	(N), (N), (N)	Hydrodynamic added-mass forces at J

continues on next page

Table 4.10 – continued from previous page

Channel Name(s)	Units	Description
JFAGxi, JFAGyi, JFAGzi, JMAGxi, JMAGyi, JMAGzi	(N), (N), (N), (N·m), (N·m), (N·m)	Marine growth mass inertia loads at J

HydroDyn is a time-domain hydrodynamics module that has been coupled into the OpenFAST wind turbine multi-physics engineering tool to enable aero-hydro-servo-elastic simulation of offshore wind turbines. HydroDyn is applicable to both fixed-bottom and floating offshore substructures. The current release of HydroDyn integrates with OpenFAST through the FAST modularization framework. HydroDyn can also be driven as a standalone code to compute hydrodynamic loading uncoupled from OpenFAST.

In addition to this documentation, the following materials including presentation slides, development plans, and publications are made available for reference. Note that some of these may be outdated and pertain to older versions of HydroDyn.

- Computation of Wave Loads Under Multidirectional Sea States for Floating Offshore Wind Turbines
- Effects of Second-Order Hydrodynamic Forces on Floating Offshore Wind Turbines
- State-Space Realization of the Wave-Radiation Force within FAST
- Dynamics of Offshore Floating Wind Turbines-Model Development and Verification
- Dynamics Modeling and Loads Analysis of an Offshore Floating Wind Turbine
- Draft Implementation Plan - Changes in HydroDyn to Support Time-Varying Buoyancy Loads on Morison Members
- Implementation Plan - Modifications to State-Space Modules in HydroDyn to Support Multiple WAMIT Bodies
- Implementation Plan (Revised) - Changes in HydroDyn to Support Multiple WAMIT Bodies
- Implementation Plan - 2nd-order Forces Within HydroDyn
- Implementation Plan - 2nd-order Wave Kinematics Within HydroDyn
- Plan for Adding Wave Stretching to HydroDyn
- Breaking Wave Modeling Approach for FAST

HydroDyn allows for multiple approaches for calculating the hydrodynamic loads on a structure:

- Potential-flow theory solution
- Strip-theory solution
- Hybrid combination of the tower

Waves generated internally within HydroDyn can be regular (periodic) or irregular (stochastic) and long-crested (unidirectional) or short-crested (wave energy spread across a range of directions). Wave elevations or full wave kinematics can also be generated externally and used within HydroDyn. Internally, HydroDyn generates waves analytically for finite depth using first-order (linear Airy) or first plus second-order wave theory [SD81] with the option to include directional spreading, but wave kinematics are only computed in the domain between the flat seabed and still-water level (SWL) and no wave stretching or higher order wave theories are included. The second-order hydrodynamic implementations include time-domain calculations of difference- (mean- and slow-drift-) and sum-frequency terms. To minimize computational expense, Fast Fourier Transforms (FFTs) are applied in the summation of all wave frequency components.

The potential-flow solution is applicable to substructures or members of substructures that are large relative to a typical wavelength. The potential-flow solution involves either frequency-to-time-domain transforms or fluid-impulse theory (FIT). In the former, potential-flow hydrodynamic loads include linear hydrostatic restoring, the added mass and

damping contributions from linear wave radiation (including free-surface memory effects), and the incident-wave excitation from first- and second-order diffraction (Froude-Kriloff and scattering). The hydrodynamic coefficients (first and second order) required for the potential-flow solution are frequency dependent and must be supplied by a separate frequency-domain panel code (e.g., WAMIT) from a pre-computation step. The radiation memory effect can be calculated either through direct time-domain convolution or through a linear state-space approach, with a state-space model derived through the [SS_Fitting](#) preprocessor. The second-order terms can be derived from the full difference- and sum-frequency quadratic transfer functions (QTFs) or the difference-frequency terms can be estimated via Standing et al.'s [\[RGSW87\]](#) extension to Newman's approximation, based only on first-order coefficients. The use of FIT is not yet documented in this manual.

The strip-theory solution may be preferable for substructures or members of substructures that are small in diameter relative to a typical wavelength. Strip-theory hydrodynamic loads can be applied across multiple interconnected members, each with possible incline and taper, and are derived directly from the undisturbed wave and current kinematics at the undisplaced position of the substructure. The strip-theory loads include the relative form of Morison's equation for the distributed fluid-inertia, added-mass, and viscous-drag components. Additional distributed load components include axial loads from tapered members and static buoyancy loads. Hydrodynamic loads are also applied as lumped loads on member endpoints (called joints). It is also possible to include flooding or ballasting of members, and the effects of marine growth. The hydrodynamic coefficients required for this solution come through user-specified dynamic-pressure, added-mass, and viscous-drag coefficients.

For some substructures and sea conditions, the hydrodynamic loads from a potential-flow theory should be augmented with the loads brought about by flow separation. For this, the viscous-drag component of the strip-theory solution may be included with the potential-flow theory solution. Another option available is to supply a global damping matrix (linear or quadratic) to the system to represent this effect.

When HydroDyn is coupled to OpenFAST, HydroDyn receives the position, orientation, velocities, and accelerations of the (rigid or flexible) substructure at each coupling time step and then computes the hydrodynamic loads and returns them back to OpenFAST. At this time, OpenFAST's ElastoDyn structural-dynamics module assumes for a floating platform that the substructure (floating platform) is a six degree-of-freedom (DOF) rigid body. For fixed-bottom offshore wind turbines, OpenFAST's SubDyn module allows for structural flexibility of multi-member substructures and the coupling to HydroDyn includes hydro-elastic effects.

The primary HydroDyn input file defines the substructure geometry, hydrodynamic coefficients, incident wave kinematics and current, potential-flow solution options, flooding/ballasting and marine growth, and auxiliary parameters. The geometry of strip-theory members is defined by joint coordinates of the undisplaced substructure in the global reference system, with the origin at the intersection of the undeflected tower centerline with mean sea level (MSL). A member connects two joints; multiple members can use a common joint. The hydrodynamic loads are computed at nodes, which are the resultant of member refinement into multiple (**MDivSize** input) elements (nodes are located at the ends of each element), and they are calculated by the module. Member properties include outer diameter, thickness, and dynamic-pressure, added-mass and viscous-drag coefficients. Member properties are specified at the joints; if properties change from one joint to the other, they will be linearly interpolated for the inner nodes.

See [Installation and Getting Started](#) for details on how to download or compile the HydroDyn and OpenFAST software executables, as well as instructions for running HydroDyn standalone and coupled to OpenFAST.

4.2.9 InflowWind Users Guide and Theory Manual

InflowWind Driver

Example input files are included in [Section 4.2.9](#).

Command-line syntax for InflowWind driver:

```
InflowWind_Driver <filename> [options]
```

(continues on next page)

(continued from previous page)

```

    where: <filename>    -- Name of driver input file to use
    options: /ifw        -- treat <filename> as name of InflowWind input file (no.
↪driver input file)

    The following options will override values in the driver input file:
        /DT[#]           -- timestep
        /TStart[#]       -- start time
        /TSteps[#]       -- number of timesteps
        /xrange[#:#]     -- range of x (#'s are reals)
        /yrange[#:#]     -- range of y
        /zrange[#:#]     -- range in z (ground = 0.0)
        /Dx[#]           -- spacing in x
        /Dy[#]           -- spacing in y
        /Dz[#]           -- spacing in z
        /points[FILE]    -- calculates at x,y,z coordinates specified in a white.
↪space delimited FILE
        /v               -- verbose output
        /vv              -- very verbose output
        /hawc            -- convert wind file specified in InflowWind to HAWC.
↪format
        /bladed          -- convert wind file specified in InflowWind to Bladed.
↪format
        /vtk             -- convert wind file specified in InflowWind to VTK format
        /help            -- print this help menu and exit

```

Notes:

- Unspecified ranges and resolutions default to what is in the file.
- If no XRange is specified, assumed to be only at X=0
- Options are not case sensitive.

The [InflowWind Manual](#) contains a description of file formats that it can read.

Specifying the InflowWind Input File

The InflowWind driver input file requires that an InflowWind input file be specified within it. See an example InflowWind input file in [Section 4.2.9](#).

Within the InflowWind input file, if the wind file being specified is Bladed native format (WindType = 7), please also see [Section 4.2.9](#).

Wind-file output formats

The InflowWind driver is capable of writing the wind data read from the input wind file into wind files of various formats.

HAWC2

This format generates the following files:

- three binary files, one for each component: `<RootName>-HAWC.u`, `<RootName>-HAWC.v`, and `<RootName>-HAWC.w`
- a text summary file in the style of HAWC2 input files: `<RootName>-HAWC.sum`

In the conversion script, the u component will have the (approximate) mean removed at each height. The mean value that was removed is displayed as comments in the text summary file. The turbulence is not scaled, so it will have the same scaling as the original file.

Bladed

This format generates a packed binary file and a text summary file.

This output format is in the Bladed-style format that TurbSim generates. That means that **the shear is included** in the file.

VTK

This format creates files in a subdirectory called `vtk`. There is one `vtk` file for each time in the full-field data structure, and the entire Y-Z grid is printed in each file. This format can be used to visualize the wind field using a viewer such as ParaView.

Converting uniform wind to full-field wind format

When converting from a uniform wind file to a full-field wind format, the following assumptions are used: - The advection speed is the time-averaged horizontal wind speed in the uniform wind file (it does not include the gust speed). - The constant time-step used in the output file is the smallest difference between any two entries in the hub-height file. - The maximum time in the uniform wind file will be used as the maximum time in the FF binary file. - The grid is generated with 5-m resolution in the lateral (Y) and horizontal (Z) directions. - The size of the grid is based on the `RefLength` parameter in the `InflowWind` input file. The converter adds approximately 10% to the grid width, with the exact size determined by achieving the desired grid resolution. The grid is centered in the lateral direction; it extends vertically above `RefHt` by the same distance as the grid width, and extends below `RefHt` to the ground (or within one grid point of the ground).

Note that there is a potential time shift between the uniform and full-field wind files, equal to the time it takes to travel the distance of half the grid width. When using the resulting full-field files, care must be taken that the aeroelastic code does not treat it as periodic.

InflowWind Input Files

Native Bladed wind file support in InflowWind

The ability to read native Bladed wind files (without scaling) has been added to `InflowWind`. To use this feature, the `WindType` must be set to 7 on line 5 of the primary `InflowWind` input file. An example of this file is given in [An example of this Native Bladed scaling file is included in Section 4.2.9.](#)

```

7                               WindType      - switch for wind file type (1=steady; 2=uniform;
↪ 3=binary TurbSim FF; 4=binary Bladed-style FF; 5=HAWC format; 6=User defined; 7=Bladed
↪ native)

```

In the section for WindType = 4, the name of an intermediate Bladed wind file should be given (including the file extension). The tower file flag is ignored.

```

===== Parameters for Binary Bladed-style Full-Field files [used only for
↪ WindType = 4] =====
"tw06_80hh_s200.BladedWind.ipt"  FilenameRoot  - Name of the Full field wind file to
↪ use (.wnd, .sum)
F                                TowerFile      - Have tower file (.twr) (flag)

```

The intermediate Bladed wind scaling file must contain the following information, which can be retrieved directly from the Bladed project simulation file from the MSTART WINDSEL and MSTART WINDV sections. Additionally, the file may include an XOFFSET line, which allows the wind to be shifted by a given distance. If not included, XOFFSET is assumed to be 0. An example of this Native Bladed scaling file is included in [Section 4.2.9](#).

```

UBAR  12
REFHT  90
TI  0.033333
TI_V  0.026667
TI_W  0.016667
WDIR  0
FLINC  .139626222222222
WINDF  "../tw06_80hh_s200.wnd"
WSHEAR  .2
XOFFSET  0

```

In the above file, the names correspond to the following:

Line	Variable Name	Units	Description
1	UBAR	(m/s)	Mean wind speed
2	REFHT	(m)	Reference height (turbine hub height)
3	TI	(-)	Turbulence intensity in longitudinal (mean wind flow) direction
4	TI_V	(-)	Turbulence intensity in horizontal direction (orthogonal to mean flow direction)
5	TI_W	(-)	Turbulence intensity in vertical direction (orthogonal to mean flow direction)
6	WDIR	(rad)	Wind direction (meteorological rotation direction)
7	FLINC	(rad)	Upflow angle (positive is up)
8	WINDF	(-)	Name of native Bladed wind file (absolute or relative path, 200 character limit)
9	WSHEAR	(-)	Power law wind shear exponent
10	XOFFSET	(m)	Turbulence box offset in the X direction (how far ahead of the turbine the turbulence box starts). If missing, this value is assumed to be 0.

Limitations: - Wind file is centered on hub height (“Best fit for rotor and tower” not implemented) - Always allow wind file to wrap around (unchecked box not implemented) - Only power-law wind profile is implemented (not logarithmic, none, or user-defined)

Angles Specified in InflowWind

Wind direction and upflow angles can be specified in the InflowWind input file. When using Native Bladed wind file support in InflowWind, the angles from the InflowWind input are overwritten with the values specified in the Native Bladed Input Files. InflowWind rotates the wind box about the hub-height tower center line by these wind direction and upflow angles.

The uniform wind files also specify wind direction and upflow angles. The angles specified in uniform wind files DO NOT rotate the wind box, but just convert the local wind speed into global coordinates.

When converting from local $[u \ v \ w]$ to global $[U \ V \ W]$ reference systems, the upflow rotation, $R(\text{upflow})$ occurs before the wind direction rotation, $R(\text{wind direction})$:

$$[U \ V \ W] = R(\text{wind direction}) * R(\text{upflow}) * [u \ v \ w]$$

When using a combination of angles in InflowWind and UniformWind files, the UniformWind angles are applied first.

Note: This means that if you have upflow specified in InflowWind and wind direction specified in UniformWind, the rotation will be performed in a different order than if both angles are specified in the same file.

$$[U \ V \ W] = R(\text{wind direction: InflowWind}) * R(\text{upflow: InflowWind}) * R(\text{wind direction: } \rightarrow \text{UniformWind}) * R(\text{upflow: UniformWind}) * [u \ v \ w]$$

Appendix

InflowWind Input Files

In this appendix we describe the InflowWind input-file structure and provide examples.

1) InflowWind Driver Input File (driver input file example):

The driver input file is needed only for the standalone version of InflowWind. It contains inputs regarding the InflowWind file, interpolation parameters, and the desired output files. The InflowWind driver can also be run without this file by using command-line arguments instead.

2) InflowWind Primary Input File (primary input file example):

The primary InflowWind input file defines the inflow that is generated or read from other files. The InflowWind file contains sections for each type of wind-file format.

3) Native Bladed Scaling File (primary input file example):

This file includes lines that determine how to scale the non-dimensional full-field turbulence files from Bladed.

4) Uniform Wind Data File (uniform wind input file example):

This file includes lines that define uniform (deterministic) wind data files.

InflowWind List of Output Channels

This is a list of all possible output parameters for the InflowWind module. See the InflowWind tab of the (OutListParameters.xlsx file):

4.2.10 ServoDyn Users Guide

This document offers a quick reference guide for the ServoDyn software program. It is intended to be used by the general user in combination with other OpenFAST manuals. The manual will be updated as new releases are issued and as needed to provide further information on advancements or modifications to the software.

The Structural control sub-module of ServoDyn is documented in [Section 4.2.11](#).

The documentation here is incomplete. Much more documentation for this module is available in the FAST v6 User's Guide as well as small updates in the FAST v8 README. These references can be downloaded in section [Section 4.1](#).

Input Files

The user configures the servodynamics model parameters via a primary ServoDyn input file, as well as separate input files for Structural control, and a controller DLL. *This information is incomplete and will be documented here at a later date.*

Units

ServoDyn uses the SI system (kg, m, s, N). Angles are assumed to be in radians unless otherwise specified.

ServoDyn Primary Input File

The primary ServoDyn input file defines the modeling options for the controller. This includes some DLL options, and Structural control options (typically a tuned mass damper system).

Simulation Control

Echo [flag]

Echo input data to <RootName>.ech

DT [sec]

Communication interval for controllers (or “default”)

Pitch Control

PCMode [switch]

Pitch control mode {0: none, 3: user-defined from routine PitchCntrl, 4: user-defined from Simulink/Labview, 5: user-defined from Bladed-style DLL}

TPCOn [sec]

Time to enable active pitch control *[unused when PCMode==0]*

TPitManS(1) [sec]

Time to start override pitch maneuver for blade 1 and end standard pitch control

TPitManS(2) [sec]

Time to start override pitch maneuver for blade 2 and end standard pitch control

TPitManS(3) [sec]

Time to start override pitch maneuver for blade 3 and end standard pitch control *[unused for 2 blades]*

PitManRat(1) [deg/s]

Pitch rate at which override pitch maneuver heads toward final pitch angle for blade 1

PitManRat(2) [deg/s]

Pitch rate at which override pitch maneuver heads toward final pitch angle for blade 2

PitManRat(3) [deg/s]

Pitch rate at which override pitch maneuver heads toward final pitch angle for blade 3 *[unused for 2 blades]*

BIPitchF(1) [deg]

Blade 1 final pitch for pitch maneuvers

BIPitchF(2) [deg]

Blade 2 final pitch for pitch maneuvers

BIPitchF(3) [deg]

Blade 3 final pitch for pitch maneuvers *[unused for 2 blades]*

Generator and Torque Control

VSContrl [switch]

Variable-speed control mode {0: none, 1: simple VS, 3: user-defined from routine UserVSCont, 4: user-defined from Simulink/Labview, 5: user-defined from Bladed-style DLL}

GenModel [switch]

Generator model {1: simple, 2: Thevenin, 3: user-defined from routine UserGen} *[used only when VSContrl==0]*

GenEff [%]

Generator efficiency *[ignored by the Thevenin and user-defined generator models]*

GenTiStr [flag]

Method to start the generator {T: timed using TimGenOn, F: generator speed using SpdGenOn}

GenTiStp [Flag]

Method to stop the generator {T: timed using TimGenOf, F: when generator power = 0}

SpdGenOn [rpm]

Generator speed to turn on the generator for a startup (HSS speed) *[used only when GenTiStri==False]*

TimGenOn [sec]

Time to turn on the generator for a startup *[used only when GenTiStr==True]*

TimGenOf [sec]

Time to turn off the generator *[used only when GenTiStp==True]*

Simple Variable-speed Torque Control

VS_RtGnSp [rpm]

Rated generator speed for simple variable-speed generator control (HSS side) *[used only when VSContrl==1]*

VS_RtTq [N-m]

Rated generator torque/constant generator torque in Region 3 for simple variable-speed generator control (HSS side) *[used only when VSContrl==1]*

VS_Rgn2K [N-m/rpm²]

Generator torque constant in Region 2 for simple variable-speed generator control (HSS side) *[used only when VSContrl==1]*

VS_SlPc [%]

Rated generator slip percentage in Region 2 1/2 for simple variable-speed generator control *[used only when VSContrl==1]*

Simple Induction Generator**SIG_SlPc [%]**

Rated generator slip percentage *[used only when VSContrl==0 and GenModel==1]*

SIG_SySp [rpm]

Synchronous (zero-torque) generator speed *[used only when VSContrl==0 and GenModel==1]*

SIG_RtTq [N-m]

Rated torque *[used only when VSContrl==0 and GenModel==1]*

SIG_PORT [-]

Pull-out ratio (Tpullout/Trated) *[used only when VSContrl==0 and GenModel==1]*

Thevenin-Equivalent Induction Generator**TEC_Freq [Hz]**

Line frequency [50 or 60] *[used only when VSContrl==0 and GenModel==2]*

TEC_NPol [-]

Number of poles [even integer > 0] *[used only when VSContrl==0 and GenModel==2]*

TEC_SRes [ohms]

Stator resistance *[used only when VSContrl==0 and GenModel==2]*

TEC_RRes [ohms]

Rotor resistance *[used only when VSContrl==0 and GenModel==2]*

TEC_VLL [volts]

Line-to-line RMS voltage *[used only when VSContrl==0 and GenModel==2]*

TEC_SLR [ohms]

Stator leakage reactance *[used only when VSContrl==0 and GenModel==2]*

TEC_RLR [ohms]

Rotor leakage reactance *[used only when VSContrl==0 and GenModel==2]*

TEC_MR [ohms]

Magnetizing reactance *[used only when VSContrl==0 and GenModel==2]*

High-speed Shaft Brake

HSSBrMode [switch]

HSS brake model {0: none, 1: simple, 3: user-defined from routine UserHSSBr, 4: user-defined from Simulink/Labview, 5: user-defined from Bladed-style DLL}

THSSBrDp [sec]

Time to initiate deployment of the HSS brake

HSSBrDT [sec]

Time for HSS-brake to reach full deployment once initiated *[used only when **HSSBrMode**==1]*

HSSBrTqF [N-m]

Fully deployed HSS-brake torque

Nacelle-yaw Control

YCMode [switch]

Yaw control mode {0: none, 3: user-defined from routine UserYawCont, 4: user-defined from Simulink/Labview, 5: user-defined from Bladed-style DLL}

TYCon [sec]

Time to enable active yaw control *[unused when **YCMode**==0]*

YawNeut [deg]

Neutral yaw position—yaw spring force is zero at this yaw

YawSpr [N-m/rad]

Nacelle-yaw spring constant

YawDamp [N-m/(rad/s)]

Nacelle-yaw damping constant

TYawManS [sec]

Time to start override yaw maneuver and end standard yaw control

YawManRat [deg/s]

Yaw maneuver rate (in absolute value)

NacYawF [deg]

Final yaw angle for override yaw maneuvers

Aerodynamic Flow Control

AfCmode [switch]

Airfoil control mode {0: none, 1: sine wave cycle, 4: user-defined from Simulink/Labview, 5: user-defined from Bladed-style DLL}

AfC_Mean [-]

Mean level for cosine cycling or steady value *[used only with AfCmode==1]*

AfC_Amp [-]

Amplitude for cosine cycling of flap signal *[used only with AfCmode==1]*

AfC_Phase [deg]

Phase relative to the blade azimuth (0 is vertical) for cosine cycling of flap signal *[used only with AfCmode==1]*

When **AfCmode==1**, the signal for the airfoil flow control is set by the expression $AfC_Mean + p\%AfC_Amp * \cos(Azimuth + AfC_phase)$ where the azimuth is the azimuth of that particular blade (azimuth=0 is considered vertical).

Cable Control

Control of cable elements specified in either the MoorDyn or SubDyn modules can be controlled through ServoDyn by a Bladed-style controller. Each cable receives a pair of controller channels, one for the requested cable length change (DeltaL), and one for the cable length rate of change (DeltaLdot). The channel assignments are requested by the modules with the cable elements (MoorDyn and/or SubDyn at present), and mapped to the appropriate control channel. A summary of which module requested the channels is available in the summary file output from ServoDyn. Up to 100 channel groups may be requested when linking to a DLL, or 20 channel groups when linking to Simulink.

CCmode [switch]

Cable control mode {0: none, 4: user-defined from Simulink/Labview, 5: user-defined from Bladed-style DLL}.

Each cable control channel group consists of a channel for DeltaL (requested cable length change) and a channel for DeltaLdot (cable length change rate) from the controller/Simulink interface.

Structural Control

See [Section 4.2.11](#) for descriptions of the mounting locations for each of the following options.

NumBStC [integer]

Number of blade structural controllers

BStCfiles [-]

Name of the files for blade structural controllers (quoted strings on one line) *[unused when NumBStC==0]*

NumNStC [integer]

Number of nacelle structural controllers

NStCfiles [-]

Name of the files for nacelle structural controllers (quoted strings on one line) *[unused when NumNStC==0]*

NumTStC [integer]

Number of tower structural controllers

TStCfiles [-]

Names of the file for tower structural control damping (quoted strings on one line) *[unused when NumTStC==0]*

NumSStC [integer]

Number of substructure structural controllers

SStCfiles [-]

Name of the files for substructure structural controllers (quoted strings on one line) *[unused when NumSStC==0]*

Bladed Controller Interface

DLL_FileName [-]

Name/location of the dynamic library {.dll [Windows] or .so [Linux]} in the Bladed-DLL format *[used only with Bladed Interface]*

DLL_InFile [-]

Name of input file sent to the DLL *[used only with Bladed Interface]*

DLL_ProcName [-]

Name of procedure in DLL to be called *[case sensitive; used only with DLL Interface]*

DLL_DT [sec]

Communication interval for dynamic library (or “default”) *[used only with Bladed Interface]*

DLL_Ramp [flag]

Whether a linear ramp should be used between DLL_DT time steps [introduces time shift when true] *[used only with Bladed Interface]*

BPCutoff [Hz]

Cutoff frequency for low-pass filter on blade pitch from DLL *[used only with Bladed Interface]*

NacYaw_North [deg]

Reference yaw angle of the nacelle when the upwind end points due North *[used only with Bladed Interface]*

Ptch_Cntrl [switch]

Record 28: Use individual pitch control {0: collective pitch; 1: individual pitch control} *[used only with Bladed Interface]*

Ptch_SetPnt [deg]

Record 5: Below-rated pitch angle set-point *[used only with Bladed Interface]*

Ptch_Min [deg]

Record 6: Minimum pitch angle *[used only with Bladed Interface]*

Ptch_Max [deg]

Record 7: Maximum pitch angle *[used only with Bladed Interface]*

PtchRate_Min [deg/s]

Record 8: Minimum pitch rate (most negative value allowed) *[used only with Bladed Interface]*

PtchRate_Max [deg/s]

Record 9: Maximum pitch rate *[used only with Bladed Interface]*

Gain_OM [N-m/(rad/s)²]

Record 16: Optimal mode gain *[used only with Bladed Interface]*

GenSpd_MinOM [rpm]

Record 17: Minimum generator speed *[used only with Bladed Interface]*

GenSpd_MaxOM [rpm]

Record 18: Optimal mode maximum speed *[used only with Bladed Interface]*

GenSpd_Dem [rpm]

Record 19: Demanded generator speed above rated *[used only with Bladed Interface]*

GenTrq_Dem [N-m]

Record 22: Demanded generator torque above rated *[used only with Bladed Interface]*

GenPwr_Dem [W]

Record 13: Demanded power *[used only with Bladed Interface]*

Bladed Interface Torque-Speed Look-up table**DLL_NumTrq** [-]

Record 26: No. of points in torque-speed look-up table {0 = none and use the optimal mode parameters; nonzero = ignore the optimal mode PARAMETERS by setting Record 16 to 0.0} *[used only with Bladed Interface]* The following 2 column table format is expected:

GenSpd_TLU (rpm)	GenTrq_TLU (N-m)
----------------------------	----------------------------

Output**SumPrint** [flag]

Print summary data to <RootName>.sum. This file contains a summary of the inputs, and will give a detailed list of the communication channels with a Bladed-style controller when used. This information may be helpful in debugging a controller, or verifying how ServoDyn is configured.

OutFile [-]

Switch to determine where output will be placed: {1: in module output file only; 2: in glue code output file only; 3: both} *(currently unused)*

TabDelim [flag]

Use tab delimiters in text tabular output file? *(currently unused)*

OutFmt [-]

Format used for text tabular output (except time). Resulting field should be 10 characters. (quoted string)
(*currently unused*)

TStart [sec]

Time to begin tabular output (*currently unused*)

OutList section controls output quantities generated by ServoDyn. Enter one or more lines containing quoted strings that in turn contain one or more output parameter names. Separate output parameter names by any combination of commas, semicolons, spaces, and/or tabs. If you prefix a parameter name with a minus sign, “-”, underscore, “_”, or the characters “m” or “M”, ServoDyn will multiply the value for that channel by -1 before writing the data. The parameters are written in the order they are listed in the input file. ServoDyn allows you to use multiple lines so that you can break your list into meaningful groups and so the lines can be shorter. You may enter comments after the closing quote on any of the lines. Entering a line with the string “END” at the beginning of the line or at the beginning of a quoted string found at the beginning of the line will cause ServoDyn to quit scanning for more lines of channel names. If ServoDyn encounters an unknown/invalid channel name, it warns the users but will remove the suspect channel from the output file. Please refer to the ServoDyn tab in the Excel file `OutListParameters.xlsx` for a complete list of possible output parameters.

Extended Bladed Interface

The Bladed style DLL controller interface was extended to allow for a significant number of new channels arranged in channel groups in reserved ranges. This is shown in Fig. 4.43 below.

Bladed interface: extended channels

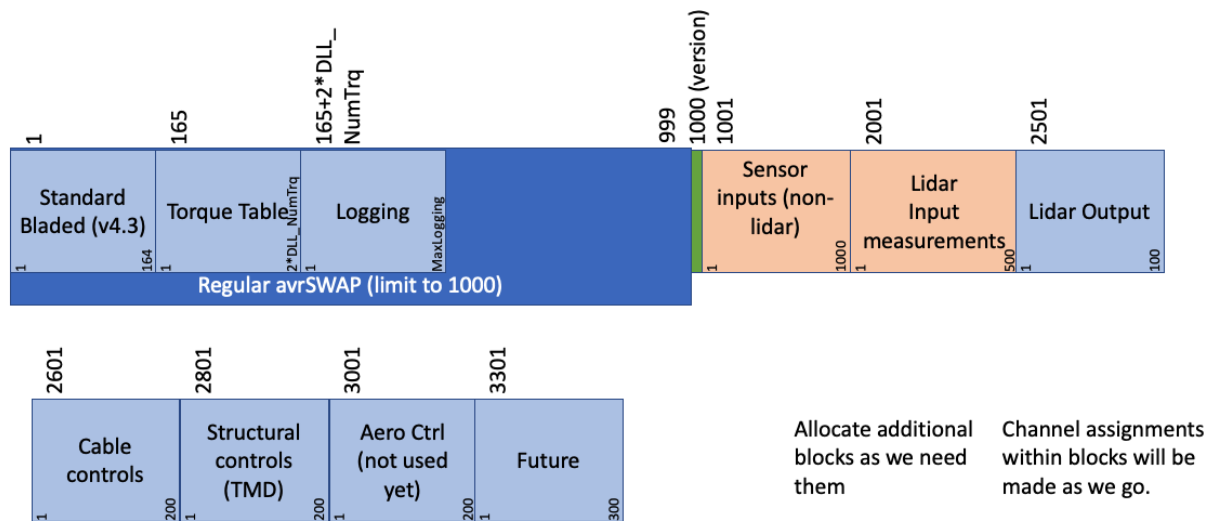


Fig. 4.43: Channel scheme for extension to the Bladed DLL interface.

The ServoDyn summary file contains a summary of all DLL interface channels in use, as well as blocks of channels that are reserved.

Legacy Bladed DLL interface channel usage by SrvD:

```
--> indicates from SrvD to DLL
<-- indicates from DLL to SrvD
```

(continues on next page)

(continued from previous page)

Record #	<->	indicates from bidirectional Description
-----		-----
1	-->	Status flag set as follows: 0 if this is the first call, 1 for all subsequent time steps, -1 if this is the final call at the end of the simulation (-)
2	-->	Current time (sec) [t in single precision]
3	-->	Communication interval (sec)
4	-->	Blade 1 pitch angle (rad) [SrvD input]
5	-->	Below-rated pitch angle set-point (rad) [SrvD Ptkh_SetPnt_ parameter]
6	-->	Minimum pitch angle (rad) [SrvD Ptkh_Min parameter]
7	-->	Maximum pitch angle (rad) [SrvD Ptkh_Max parameter]
8	-->	Minimum pitch rate (most negative value allowed) (rad/s) [SrvD_ PtkhRate_Min parameter]
9	-->	Maximum pitch rate (rad/s) [SrvD_ PtkhRate_Max parameter]
10	-->	0 = pitch position actuator, 1 = pitch rate actuator (-) [must be_ 0 for ServoDyn]
11	-->	Current demanded pitch angle (rad) [I am sending the previous_ value for blade 1 from the DLL, in the absence of any more information provided in_ Bladed documentation]
12	-->	Current demanded pitch rate (rad/s) [always zero for ServoDyn]
13	-->	Demanded power (W) [SrvD GenPwr_Dem parameter from input file]
14	-->	Measured shaft power (W) [SrvD input]
15	-->	Measured electrical power output (W) [SrvD calculation from_ previous step; should technically be a state]
16	-->	Optimal mode gain (Nm/(rad/s)^2) [if torque-speed table look-up_ not selected in input file, use SrvD Gain_OM parameter, otherwise use 0 (already_ overwritten in Init routine)]
17	-->	Minimum generator speed (rad/s) [SrvD GenSpd_MinOM parameter]
18	-->	Optimal mode maximum speed (rad/s) [SrvD GenSpd_MaxOMp parameter]
19	-->	Demanded generator speed above rated (rad/s) [SrvD GenSpd_Dem_ parameter]
20	-->	Measured generator speed (rad/s) [SrvD input]
21	-->	Measured rotor speed (rad/s) [SrvD input]
22	-->	Demanded generator torque above rated (Nm) [SrvD GenTrq_Dem_ parameter from input file]
23	-->	Measured generator torque (Nm) [SrvD calculation from previous_ step; should technically be a state]
24	-->	Measured yaw error (rad) [SrvD input]
25	-->	Start of below-rated torque-speed look-up table (Lookup table not_ in use)
26	-->	No. of points in torque-speed look-up table (-) [SrvD DLL_NumTrq_ parameter]:
27	-->	Hub wind speed (m/s) [SrvD input]
28	-->	Pitch control: 0 = collective, 1 = individual (-) [SrvD Ptkh_ Cntrl parameter]
29	-->	Yaw control: 0 = yaw rate control, 1 = yaw torque control (-) [must be 0 for ServoDyn]
30	-->	Blade 1 root out-of-plane bending moment (Nm) [SrvD input]
31	-->	Blade 2 root out-of-plane bending moment (Nm) [SrvD input]

(continues on next page)

(continued from previous page)

```

32      --> Blade 3 root out-of-plane bending moment (Nm) [SrvD input]
33      --> Blade 2 pitch angle (rad) [SrvD input]
34      --> Blade 3 pitch angle (rad) [SrvD input]
35      <-- Generator contactor (-) [sent to DLL at the next call]
36      <-- Shaft brake status (-) [sent to DLL at the next call; anything
↳ other than 0 or 1 is an error]
37      --> Nacelle yaw angle from North (rad)
41      <-- demanded yaw actuator torque [this output is ignored since record
↳ 29 is set to 0 by ServoDyn indicating yaw rate control]
45      <-- Demanded pitch angle (Collective pitch) (rad)
47      <-- Demanded generator torque (Nm)
48      <-- Demanded nacelle yaw rate (rad/s)
49      --> Maximum number of characters in the "MESSAGE" argument (-) [size
↳ of ErrMsg argument plus 1 (we add one for the C NULL CHARACTER)]
50      --> Number of characters in the "INFILE" argument (-) [trimmed
↳ length of DLL_InFile parameter plus 1 (we add one for the C NULL CHARACTER)]
51      --> Number of characters in the "OUTNAME" argument (-) [trimmed
↳ length of RootName parameter plus 1 (we add one for the C NULL CHARACTER)]
53      --> Tower top fore-aft acceleration (m/s^2) [SrvD input]
54      --> Tower top side-to-side acceleration (m/s^2) [SrvD input]
55      <-- UNUSED: Pitch override [anything other than 0 is an error in
↳ ServoDyn]
56      <-- UNUSED: Torque override [anything other than 0 is an error in
↳ ServoDyn]
60      --> Rotor azimuth angle (rad) [SrvD input]
61      --> Number of blades (-) [SrvD NumBl parameter]
62      --> Maximum number of values which can be returned for logging (-)
↳ [set to 300]
63      <-- Number logging channels
64      --> Maximum number of characters which can be returned in "OUTNAME" (-
↳ ) [set to 12601 (including the C NULL CHARACTER)]
65      <-- Number of variables returned for logging [anything greater than
↳ MaxLoggingChannels is an error]
69      --> Blade 1 root in-plane bending moment (Nm) [SrvD input]
70      --> Blade 2 root in-plane bending moment (Nm) [SrvD input]
71      --> Blade 3 root in-plane bending moment (Nm) [SrvD input]
73      --> Rotating hub My (GL co-ords) (Nm) [SrvD input]
74      --> Rotating hub Mz (GL co-ords) (Nm) [SrvD input]
75      --> Fixed hub My (GL co-ords) (Nm) [SrvD input]
76      --> Fixed hub Mz (GL co-ords) (Nm) [SrvD input]
77      --> Yaw bearing My (GL co-ords) (Nm) [SrvD input]
78      --> Yaw bearing Mz (GL co-ords) (Nm) [SrvD input]
82      --> Nacelle roll acceleration (rad/s^2) [SrvD input] -- this is in
↳ the shaft (tilted) coordinate system, instead of the nacelle (nontilted) coordinate
↳ system
83      --> Nacelle nodding acceleration (rad/s^2) [SrvD input]
84      --> Nacelle yaw acceleration (rad/s^2) [SrvD input] -- this is in
↳ the shaft (tilted) coordinate system, instead of the nacelle (nontilted) coordinate
↳ system
95      --> Reserved (SrvD customization: set to SrvD AirDens parameter)
96      --> Reserved (SrvD customization: set to SrvD AvgWindSpeed parameter)
109     --> Shaft torque (=hub Mx for clockwise rotor) (Nm) [SrvD input]

```

(continues on next page)

(continued from previous page)

```

110      --> Thrust - Rotating low-speed shaft force x (GL co-ords) (N) [SrvD_
↳input]
111      --> Nonrotating low-speed shaft force y (GL co-ords) (N) [SrvD input]
112      --> Nonrotating low-speed shaft force z (GL co-ords) (N) [SrvD input]
117      --> Controller state [always set to 0]
120      <-- Airfoil command, blade 1
121      <-- Airfoil command, blade 2
122      <-- Airfoil command, blade 3
129      --> Maximum extent of the avrSWAP array: 3300

```

Legacy Bladed DLL interface with Extended avrSWAP
channel usage by SrvD:

```

                                --> indicates from SrvD to DLL
                                <-- indicates from DLL to SrvD
Record #                       Requested by           Description
-----
1000      -->                               Version of extended avrSWAP: 1
1001      -->                               General channel group -- Platform motion -
↳- Displacement TDX (m)
1002      -->                               General channel group -- Platform motion -
↳- Displacement TDY (m)
1003      -->                               General channel group -- Platform motion -
↳- Displacement TDZ (m)
1004      -->                               General channel group -- Platform motion -
↳- Displacement RDX (rad)
1005      -->                               General channel group -- Platform motion -
↳- Displacement RDY (rad)
1006      -->                               General channel group -- Platform motion -
↳- Displacement RDZ (rad)
1007      -->                               General channel group -- Platform motion -
↳- Velocity TVX (m/s)
1008      -->                               General channel group -- Platform motion -
↳- Velocity TVY (m/s)
1009      -->                               General channel group -- Platform motion -
↳- Velocity TVZ (m/s)
1010      -->                               General channel group -- Platform motion -
↳- Velocity RVX (rad/s)
1011      -->                               General channel group -- Platform motion -
↳- Velocity RVY (rad/s)
1012      -->                               General channel group -- Platform motion -
↳- Velocity RVZ (rad/s)
1013      -->                               General channel group -- Platform motion -
↳- Acceleration TAX (m/s^2)
1014      -->                               General channel group -- Platform motion -
↳- Acceleration TAY (m/s^2)
1015      -->                               General channel group -- Platform motion -
↳- Acceleration TAZ (m/s^2)
1016      -->                               General channel group -- Platform motion -
↳- Acceleration RAX (rad/s^2)
1017      -->                               General channel group -- Platform motion -
↳- Acceleration RAY (rad/s^2)

```

(continues on next page)

(continued from previous page)

1018	-->		General channel group -- Platform motion -
↳ Acceleration RAZ (rad/s^2)			
2000	-->		Ending index for the non-lidar_
↳ measurements channel block			
2001	-->		Starting index for the lidar measurements_
↳ channel block			
2500	-->		Ending index for the lidar measurements_
↳ channel block			
2501	<--		Starting index for the lidar control_
↳ channel block			
2600	<--		Ending index for the lidar control_
↳ channel block			
2601	<--	MoorDyn	Cable control channel group 1 -- DeltaL
2602	<--	MoorDyn	Cable control channel group 1 -- DeltaLdot
2603	<--		Cable control channel group 2 -- DeltaL
2604	<--		Cable control channel group 2 -- DeltaLdot
2605	<--	MoorDyn	Cable control channel group 3 -- DeltaL
2606	<--	MoorDyn	Cable control channel group 3 -- DeltaLdot
2800	<--		Ending index for the cable control_
↳ channel block			
2801	-->		StC control channel group 1 -- StC_Disp_X
2802	-->		StC control channel group 1 -- StC_Disp_Y
2803	-->		StC control channel group 1 -- StC_Disp_Z
2804	-->		StC control channel group 1 -- StC_Vel_X
2805	-->		StC control channel group 1 -- StC_Vel_Y
2806	-->		StC control channel group 1 -- StC_Vel_Z
2807	<--		StC control channel group 1 -- StC_Stiff_
↳ X (override spring constant)			
2808	<--		StC control channel group 1 -- StC_Stiff_
↳ Y (override spring constant)			
2809	<--		StC control channel group 1 -- StC_Stiff_
↳ Z (override spring constant)			
2810	<--		StC control channel group 1 -- StC_Damp_X_
↳ (override damping constant)			
2811	<--		StC control channel group 1 -- StC_Damp_Y_
↳ (override damping constant)			
2812	<--		StC control channel group 1 -- StC_Damp_Z_
↳ (override damping constant)			
2813	<--		StC control channel group 1 -- StC_Brake_
↳ X (braking force)			
2814	<--		StC control channel group 1 -- StC_Brake_
↳ Y (braking force)			
2815	<--		StC control channel group 1 -- StC_Brake_
↳ Z (braking force)			
2816	<--		StC control channel group 1 -- StC_Force_
↳ X (additional force)			
2817	<--		StC control channel group 1 -- StC_Force_
↳ Y (additional force)			
2818	<--		StC control channel group 1 -- StC_Force_
↳ Z (additional force)			
2819	<--		StC control channel group 1 -- Reserved_
↳ for future			

(continues on next page)

(continued from previous page)

2820	<--		StC control channel group 1 -- Reserved_
↪for future			
2821	-->	SStC2	StC control channel group 2 -- StC_Displ_X
2822	-->	SStC2	StC control channel group 2 -- StC_Displ_Y
2823	-->	SStC2	StC control channel group 2 -- StC_Displ_Z
2824	-->	SStC2	StC control channel group 2 -- StC_Vel_X
2825	-->	SStC2	StC control channel group 2 -- StC_Vel_Y
2826	-->	SStC2	StC control channel group 2 -- StC_Vel_Z
2827	<--	SStC2	StC control channel group 2 -- StC_Stiff_
↪X (override spring constant)			
2828	<--	SStC2	StC control channel group 2 -- StC_Stiff_
↪Y (override spring constant)			
2829	<--	SStC2	StC control channel group 2 -- StC_Stiff_
↪Z (override spring constant)			
2830	<--	SStC2	StC control channel group 2 -- StC_Damp_X_
↪ (override damping constant)			
2831	<--	SStC2	StC control channel group 2 -- StC_Damp_Y_
↪ (override damping constant)			
2832	<--	SStC2	StC control channel group 2 -- StC_Damp_Z_
↪ (override damping constant)			
2833	<--	SStC2	StC control channel group 2 -- StC_Brake_
↪X (braking force)			
2834	<--	SStC2	StC control channel group 2 -- StC_Brake_
↪Y (braking force)			
2835	<--	SStC2	StC control channel group 2 -- StC_Brake_
↪Z (braking force)			
2836	<--	SStC2	StC control channel group 2 -- StC_Force_
↪X (additional force)			
2837	<--	SStC2	StC control channel group 2 -- StC_Force_
↪Y (additional force)			
2838	<--	SStC2	StC control channel group 2 -- StC_Force_
↪Z (additional force)			
2839	<--	SStC2	StC control channel group 2 -- Reserved_
↪for future			
2840	<--	SStC2	StC control channel group 2 -- Reserved_
↪for future			
3000	<--		Ending index for the StC control_
↪channel block			

4.2.11 Structural Control (SrvD)

Input Files

The user configures each StC instance with a separate input file. This input file defines the location of the StC relative to its mounting location, and defines the properties. It can also be used with an external forces file to apply a timeseries load at the location (primarily used for diagnostic purposes).

Units

Structural Control uses the SI system (kg, m, s, N). Angles are assumed to be in radians unless otherwise specified.

Structural Control Locations

The Structural Control input file defines the location and properties of the StC instance. The location is relative to the type of StC given in the main ServoDyn input file (see [Section 4.2.10](#)). The four location types are Nacelle, Tower, Blade, and Platform.

The mapping information for the StC will be given in the main OpenFAST summary file.

Nacelle StC

This StC mounting location is attached relative to the nacelle reference point. It will track with all nacelle motions (including motions due to yaw, tower flex, and platform motions).

Tower StC

This StC mounting location is attached to the tower mesh at the height specified above the tower base. This StC attachment will move with the line mesh at that height. For example, an StC mounted at 85 m on a 90 m tower will move with the mesh line corresponding to the 85 m height position on the tower center line.

Blade StC

This StC mounting location is attached to the blade structural center at the specified distance from the blade root along the z-axis of the blade (IEC blade coordinate system). This location will follow all blade deformations and motions (including blade twist when used with BeamDyn). This option is available with both the BeamDyn and ElastoDyn blade representations.

When this option is used, identical StCs will be attached at each of the blades. The response of each blade mounted StC is tracked separately and is available in the output channels given in the ServoDyn tab of the `OutListParameters.xlsx`.

Platform StC

This StC mounting location is located relative to the platform reference point. When a rigid body platform is modeled (such as a rigid semi-submersible), it is attached to the platform reference point. When a flexible floating body is modeled, the StC is attached to the SubDyn mesh.

Structural Control Input File

The input file may have an arbitrary number of commented header lines, and commented lines at any location in the input file. (Example Structural Control input file for tuned mass damper on tower for NREL 5 MW TLP):

Simulation Control

Echo [flag]

Echo input data to <RootName>.ech

StC Degrees of Freedom

StC_DOF_MODE [switch]

DOF mode {0: No StC or TLCD DOF; 1: StC_X_DOF, StC_Y_DOF, and/or StC_Z_DOF (three independent StC DOFs); 2: StC_XY_DOF (Omni-Directional StC); 3: TLCD; 4: Prescribed force/moment time series}

StC_X_DOF [flag]

DOF on or off for StC X [*Used only when StC_DOF_MODE==1*]

StC_Y_DOF [flag]

DOF on or off for StC Y [*Used only when StC_DOF_MODE==1*]

StC_Z_DOF [flag]

DOF on or off for StC Z [*Used only when StC_DOF_MODE==1*]

StC Location

The location of the StC is relative to the component it is attached to. This is specified in the main ServoDyn input file. See description above.

StC_P_X [m]

At rest X position of StC

StC_P_Y [m]

At rest Y position of StC

StC_P_Z [m]

At rest Z position of StC

StC Initial Conditions

used only when StC_DOF_MODE==1 or 2

StC_X_DSP [m]

StC X initial displacement *[relative to at rest position]*

StC_Y_DSP [m]

StC Y initial displacement *[relative to at rest position]*

StC_Z_DSP [m]

StC Z initial displacement *[relative to at rest position; used only when StC_DOF_MODE==1 and StC_Z_DOF==TRUE]*

StC_Z_PreLd [N]

StC Z spring preload. Either a direct value for the spring preload in Newtons, or “**gravity**” for pre-loading spring to shift the at rest position of the StC Z mass when gravity is acting on it using $F_{Z_{PreLoad}} = M_Z * G$, or “**none**” to disable spring pre-load. See [Section 4.2.11](#) for details of implementation. *[used only when StC_DOF_MODE=1 and StC_Z_DOF=TRUE]*

StC Configuration

used only when StC_DOF_MODE==1 or 2

StC_X_PSP [m]

Positive stop position – maximum X mass displacement

StC_X_NSP [m]

Negative stop position – minimum X mass displacement

StC_Y_PSP [m]

Positive stop position – maximum Y mass displacement

StC_Y_NSP [m]

Negative stop position – minimum Y mass displacement

StC_Z_PSP [m]

Positive stop position – maximum Z mass displacement *[used only when StC_DOF_MODE==1 and StC_Z_DOF==TRUE]*

StC_Z_NSP [m]

Negative stop position – minimum Z mass displacement *[used only when StC_DOF_MODE==1 and StC_Z_DOF==TRUE]*

StC Mass, Stiffness, & Damping

used only when StC_DOF_MODE==1 or 2

StC_X_M [kg]

StC X mass *[used only when StC_DOF_MODE==1 and StC_X_DOF==TRUE]*

StC_Y_M [kg]

StC Y mass *[used only when StC_DOF_MODE==1 and StC_Y_DOF==TRUE]*

StC_Z_M [kg]

StC Z mass *[used only when StC_DOF_MODE==1 and StC_Z_DOF==TRUE]*

StC_XY_M [kg]

StC XY mass *[used only when StC_DOF_MODE==2]*

StC_X_K [N/m]

StC X stiffness

StC_Y_K [N/m]

StC Y stiffness

StC_Z_K [N/m]

StC Z stiffness *[used only when StC_DOF_MODE==1 and StC_Z_DOF==TRUE]*

StC_X_C [N/(m/s)]

StC X damping

StC_Y_C [N/(m/s)]

StC Y damping

StC_Z_C [N/(m/s)]

StC Z damping *[used only when StC_DOF_MODE==1 and StC_Z_DOF==TRUE]*

StC_X_KS [N/m]

Stop spring X stiffness

StC_Y_KS [N/m]

Stop spring Y stiffness

StC_Z_KS [N/m]

Stop spring Z stiffness *[used only when StC_DOF_MODE==1 and StC_Z_DOF==TRUE]*

StC_X_CS [N/(m/s)]

Stop spring X damping

StC_Y_CS [N/(m/s)]

Stop spring Y damping

StC_Z_CS [N/(m/s)]

Stop spring Z damping *[used only when StC_DOF_MODE==1 and StC_Z_DOF==TRUE]*

StC User-Defined Spring Forces

used only when **StC_DOF_MODE**==1 or 2

Use_F_TBL [flag]

Use spring force from user-defined table

NKInpSt [-]

Number of spring force input stations

The table is expected to contain 6 columns for displacements and equivalent spring forces: **X**, **F_X**, **Y**, **F_Y**, **Z**, and **F_Z**. Displacements are in meters (m) and forces in Newtons (N).

Example spring forces table:

X (m)	F_X (N)	Y (m)	F_Y (N)	Z (m)	F_Z (N)
-6.000000E+00 ↪8000000E+06	-4.800000E+06	-6.000000E+00	-4.800000E+06	-6.000000E+00	-4.800000E+06
-5.000000E+00 ↪4000000E+06	-2.400000E+06	-5.000000E+00	-2.400000E+06	-5.000000E+00	-2.400000E+06
-4.500000E+00 ↪2000000E+06	-1.200000E+06	-4.500000E+00	-1.200000E+06	-4.500000E+00	-1.200000E+06
-4.000000E+00 ↪0000000E+05	-6.000000E+05	-4.000000E+00	-6.000000E+05	-4.000000E+00	-6.000000E+05
-3.500000E+00 ↪0000000E+05	-3.000000E+05	-3.500000E+00	-3.000000E+05	-3.500000E+00	-3.000000E+05
-3.000000E+00 ↪5000000E+05	-1.500000E+05	-3.000000E+00	-1.500000E+05	-3.000000E+00	-1.500000E+05
-2.500000E+00 ↪0000000E+05	-1.000000E+05	-2.500000E+00	-1.000000E+05	-2.500000E+00	-1.000000E+05
-2.000000E+00 ↪5000000E+04	-6.500000E+04	-2.000000E+00	-6.500000E+04	-2.000000E+00	-6.500000E+04
0.000000E+00 ↪0000000E+00	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
2.000000E+00 ↪5000000E+04	6.500000E+04	2.000000E+00	6.500000E+04	2.000000E+00	6.500000E+04
2.500000E+00 ↪0000000E+05	1.000000E+05	2.500000E+00	1.000000E+05	2.500000E+00	1.000000E+05
3.000000E+00 ↪5000000E+05	1.500000E+05	3.000000E+00	1.500000E+05	3.000000E+00	1.500000E+05
3.500000E+00 ↪0000000E+05	3.000000E+05	3.500000E+00	3.000000E+05	3.500000E+00	3.000000E+05
4.000000E+00 ↪0000000E+05	6.000000E+05	4.000000E+00	6.000000E+05	4.000000E+00	6.000000E+05
4.500000E+00 ↪2000000E+06	1.200000E+06	4.500000E+00	1.200000E+06	4.500000E+00	1.200000E+06
5.000000E+00 ↪4000000E+06	2.400000E+06	5.000000E+00	2.400000E+06	5.000000E+00	2.400000E+06
6.000000E+00 ↪8000000E+06	4.800000E+06	6.000000E+00	4.800000E+06	6.000000E+00	4.800000E+06

StructCtrl Control

used only when StC_DOF_MODE==1 or 2

StC_CMODE [switch]

Control mode {0:none; 1: Semi-Active Control Mode; 2: Active Control Mode}

StC_SA_MODE [-]

Semi-Active control mode { 1: velocity-based ground hook control; 2: Inverse velocity-based ground hook control; 3: displacement-based ground hook control 4: Phase difference Algorithm with Friction Force 5: Phase difference Algorithm with Damping Force }

StC_X_C_HIGH [-]

StC X high damping for ground hook control

StC_X_C_LOW [-]

StC X low damping for ground hook control

StC_Y_C_HIGH [-]

StC Y high damping for ground hook control

StC_Y_C_LOW [-]

StC Y low damping for ground hook control

StC_Z_C_HIGH [-]

StC Z high damping for ground hook control *[used only when StC_DOF_MODE==1 and StC_Z_DOF==TRUE]*

StC_Z_C_LOW [-]

StC Z low damping for ground hook control *[used only when StC_DOF_MODE==1 and StC_Z_DOF==TRUE]*

StC_X_C_BRAKE [-]

StC X high damping for braking the StC *[currently unused. set to zero]*

StC_Y_C_BRAKE [-]

StC Y high damping for braking the StC *[currently unused. set to zero]*

StC_Z_C_BRAKE [-]

StC Z high damping for braking the StC *[used only when StC_DOF_MODE==1 and StC_Z_DOF==TRUE]* *[currently unused. set to zero]*

TLCD – Tuned Liquid Column Damper

used only when StC_DOF_MODE==3

L_X [m]

X TLCD total length

B_X [m]

X TLCD horizontal length

area_X [m²]

X TLCD cross-sectional area of vertical column

area_ratio_X [-]

X TLCD cross-sectional area ratio *[vertical column area divided by horizontal column area]*

headLossCoeff_X [-]

X TLCD head loss coeff

rho_X [kg/m³]

X TLCD liquid density

L_Y [m]

Y TLCD total length

B_Y [m]

Y TLCD horizontal length

area_Y [m²]

Y TLCD cross-sectional area of vertical column

area_ratio_Y [-]

Y TLCD cross-sectional area ratio *[vertical column area divided by horizontal column area]*

headLossCoeff_Y [-]

Y TLCD head loss coeff

rho_Y [kg/m³]

Y TLCD liquid density

Prescribed Time Series

A prescribed time series of forces and moments may be applied in place of the StC damper. The force and moment may be applied either in a global coordinate frame, or in a local (following) coordinate frame. This feature is *used only when StC_DOF_MODE==4*.

PrescribedForcesCoord [switch]

Prescribed forces are in global or local coordinates {1: global; 2: local}

PrescribedForcesFile [-]

Filename for the prescribed forces. The format expected is 7 columns: time, FX, FY, FZ, MX, MY, MZ. Values will be interpolated from the file between the given timestep and value sets. The input file may have an arbitrary number of commented header lines, and commented lines at any location in the input file.

Example prescribed time series file (example prescribed force timeseries):

```
# This is an input file for the tower top force time-series in the TMD module of ServoDyn
#
# it has an arbitrary number of header lines denoted with #!% characters
! Another comment line
#
# Time,      Fx,      Fy,      Fz,      Mx,      My,      Mz
# (s)        (N)       (N)       (N)       (N-m)    (N-m)    (N-m)
0.0          0.0      0.0      0.0      0.0      0.0      0.0
```

(continues on next page)

(continued from previous page)

```

4.0      1.0e5    0.0    0.0    0.0    0.0    0.0    # Start ramp -- this
↪is a comment
40.0     1.0e5    0.0    0.0    0.0    0.0    0.0
# 40.0001    0.0    0.0    0.0    0.0    0.0    0.0    # This is a commented
↪line
90.      0.0     0.0    0.0    0.0    0.0    0.0

```

Theory Manual for the Tuned Mass Damper Module in OpenFAST

Author

William La Cava & Matthew A. Lackner Department of Mechanical and Industrial Engineering
University of Massachusetts Amherst Amherst, MA 01003 wlacava@umass.edu, lackner@ecs.
umass.edu

This document was edited by Jason M. Jonkman of NREL to include an independent vertically oriented TMD in OpenFAST. jason.jonkman@nrel.gov

This manual describes updated functionality in OpenFAST that simulates the addition of tuned mass dampers (TMDs) for structural control. The dampers can be added to the blades, nacelle, tower, or substructure. For application studies of these systems, refer to [[stc-LR11a](#), [stc-LR11b](#), [stc-NRL13](#), [stc-SL13](#), [stc-SL11](#), [stc-SL14](#)]. The TMDs are three independent, 1 DOF, linear mass spring damping elements that act in the local x , y , and z coordinate systems of each component. The other functionality of the structural control (StC) module, including an omnidirectional TMD and TLCD are not documented herein. We first present the theoretical background and then describe the code changes.

Theoretical Background

Definitions

Table 4.11: Definitions

Variable	Description
O	origin point of global inertial reference frame
P	origin point of non-inertial reference frame fixed to component (blade, nacelle, tower, substructure) where TMDs are at rest
TMD	location point of a TMD
G	axis orientation of global ref-

Equations of motion

The position vectors of the TMDs in the two reference frames O and P are related by

$$\vec{r}_{TMD/O_G} = \vec{r}_{P/O_G} + \vec{r}_{TMD/P_G}$$

Expressed in orientation N ,

$$\begin{aligned}\vec{r}_{TMD/O_N} &= \vec{r}_{P/O_N} + \vec{r}_{TMD/P_N} \\ \Rightarrow \vec{r}_{TMD/P_N} &= \vec{r}_{TMD/O_N} - \vec{r}_{P/O_N}\end{aligned}$$

Differentiating,¹

$$\dot{\vec{r}}_{TMD/P_N} = \dot{\vec{r}}_{TMD/O_N} - \dot{\vec{r}}_{P/O_N} - \vec{\omega}_{N/O_N} \times \vec{r}_{TMD/P_N}$$

differentiating again gives the acceleration of the TMD w.r.t. P (the nacelle position), oriented with N :

$$\begin{aligned}\ddot{\vec{r}}_{TMD/P_N} &= \ddot{\vec{r}}_{TMD/O_N} - \ddot{\vec{r}}_{P/O_N} - \vec{\omega}_{N/O_N} \times (\vec{\omega}_{N/O_N} \times \vec{r}_{TMD/P_N}) \\ &\quad - \dot{\vec{\omega}}_{N/O_N} \times \vec{r}_{TMD/P_N} - 2\vec{\omega}_{N/O_N} \times \dot{\vec{r}}_{TMD/P_N}\end{aligned}\tag{4.187}$$

The right-hand side contains the following terms:

Table 4.12: RHS terms

$\ddot{\vec{r}}_{TMD/O_N}$	acceleration of the TMD in the <i>inertial</i> frame O_N
$\ddot{\vec{r}}_{P/O_N} = R_{N/G} \ddot{\vec{r}}_{P/O_G}$	acceleration of the Nacelle origin P w.r.t. O_N
$\vec{\omega}_{N/O_N} = R_{N/G} \vec{\omega}_{N/O_G}$	angular velocity of nacelle w.r.t. O_N
$\vec{\omega}_{N/O_N} \times (\vec{\omega}_{N/O_N} \times \vec{r}_{TMD/P_N})$	Centripetal acceleration
$\dot{\vec{\omega}}_{N/O_N} \times \vec{r}_{TMD/P_N}$	Tangential acceleration
$2\vec{\omega}_{N/O_N} \times \dot{\vec{r}}_{TMD/P_N}$	Coriolis acceleration

The acceleration in the inertial frame $\ddot{\vec{r}}_{TMD/O_N}$ can be replaced with a force balance

$$\ddot{\vec{r}}_{TMD/O_N} = \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix}_{TMD/O_N} = \frac{1}{m} \begin{bmatrix} \sum F_X \\ \sum F_Y \\ \sum F_Z \end{bmatrix}_{TMD/O_N} = \frac{1}{m} \vec{F}_{TMD/O_N}$$

Substituting the force balance into Equation (4.187) gives the general equation of motion for a TMD:

$$\begin{aligned}\ddot{\vec{r}}_{TMD/P_N} &= \frac{1}{m} \vec{F}_{TMD/O_N} - \ddot{\vec{r}}_{P/O_N} - \vec{\omega}_{N/O_N} \times (\vec{\omega}_{N/O_N} \times \vec{r}_{TMD/P_N}) \\ &\quad - \dot{\vec{\omega}}_{N/O_N} \times \vec{r}_{TMD/P_N} - 2\vec{\omega}_{N/O_N} \times \dot{\vec{r}}_{TMD/P_N}\end{aligned}\tag{4.188}$$

We will now solve the equations of motion for TMD_X , TMD_Y , and TMD_Z .

TMD_X :

The external forces \vec{F}_{TMD_X/O_N} are given by

$$\vec{F}_{TMD_X/O_N} = \begin{bmatrix} -c_x \dot{x}_{TMD_X/P_N} - k_x x_{TMD_X/P_N} + m_x a_{G_X/O_N} + F_{ext_x} + F_{StopFreq} \\ F_{Y_{TMD_X/O_N}} + m_x a_{G_Y/O_N} \\ F_{Z_{TMD_X/O_N}} + m_x a_{G_Z/O_N} \end{bmatrix}$$

¹ Note that $(Ra) \times (Rb) = R(a \times b)$.

TMD_X is fixed to frame N in the y and z directions so that

$$r_{TMD_X/P_N} = \begin{bmatrix} x_{TMD_X/P_N} \\ 0 \\ 0 \end{bmatrix}$$

The other components of Eqn. (4.188) are:

$$\begin{aligned} \vec{\omega}_{N/O_N} \times (\vec{\omega}_{N/O_N} \times \vec{r}_{TMD_X/P_N}) &= x_{TMD_X/P_N} \begin{bmatrix} -(\dot{\phi}_{N/O_N}^2 + \dot{\psi}_{N/O_N}^2) \\ \dot{\theta}_{N/O_N} \dot{\phi}_{N/O_N} \\ \dot{\theta}_{N/O_N} \dot{\psi}_{N/O_N} \end{bmatrix} \\ 2\vec{\omega}_{N/O_N} \times \dot{\vec{r}}_{TMD_X/P_N} &= \dot{x}_{TMD_X/P_N} \begin{bmatrix} 0 \\ 2\dot{\psi}_{N/O_N} \\ -2\dot{\phi}_{N/O_N} \end{bmatrix} \\ \vec{\alpha}_{N/O_N} \times \vec{r}_{TMD_X/P_N} &= x_{TMD_X/P_N} \begin{bmatrix} 0 \\ \ddot{\psi}_{N/O_N} \\ -\ddot{\phi}_{N/O_N} \end{bmatrix} \end{aligned}$$

Therefore \ddot{x}_{TMD_X/P_N} is governed by the equations

$$\begin{aligned} \ddot{x}_{TMD_X/P_N} &= (\dot{\phi}_{N/O_N}^2 + \dot{\psi}_{N/O_N}^2 - \frac{k_x}{m_x}) x_{TMD_X/P_N} - (\frac{c_x}{m_x}) \dot{x}_{TMD_X/P_N} - \ddot{x}_{P/O_N} + a_{G_X/O_N} \\ &+ \frac{1}{m_x} (F_{ext_X} + F_{StopFrc_X}) \end{aligned} \quad (4.189)$$

The forces $F_{Y_{TMD_X/O_N}}$ and $F_{Z_{TMD_X/O_N}}$ are solved noting $\ddot{y}_{TMD_X/P_N} = \ddot{z}_{TMD_X/P_N} = 0$:

$$F_{Y_{TMD_X/O_N}} = m_x \left(-a_{G_Y/O_N} + \ddot{y}_{P/O_N} + (\ddot{\psi}_{N/O_N} + \dot{\theta}_{N/O_N} \dot{\phi}_{N/O_N}) x_{TMD_X/P_N} + 2\dot{\psi}_{N/O_N} \dot{x}_{TMD_X/P_N} \right) \quad (4.190)$$

$$F_{Z_{TMD_X/O_N}} = m_x \left(-a_{G_Z/O_N} + \ddot{z}_{P/O_N} - (\ddot{\phi}_{N/O_N} - \dot{\theta}_{N/O_N} \dot{\psi}_{N/O_N}) x_{TMD_X/P_N} - 2\dot{\phi}_{N/O_N} \dot{x}_{TMD_X/P_N} \right) \quad (4.191)$$

TMD_Y:

The external forces \vec{F}_{TMD_Y/P_N} on TMD_Y are given by

$$\vec{F}_{TMD_Y/P_N} = \begin{bmatrix} F_{X_{TMD_Y/O_N}} + m_y a_{G_X/O_N} \\ -c_y \dot{y}_{TMD_Y/P_N} - k_y y_{TMD_Y/P_N} + m_y a_{G_Y/O_N} + F_{ext_y} + F_{StopFrc_y} \\ F_{Z_{TMD_Y/O_N}} + m_y a_{G_Z/O_N} \end{bmatrix}$$

TMD_Y is fixed to frame N in the x and z directions so that

$$r_{TMD_Y/P_N} = \begin{bmatrix} 0 \\ y_{TMD_Y/P_N} \\ 0 \end{bmatrix}$$

The other components of Eqn. (4.188) are:

$$\vec{\omega}_{N/O_N} \times (\vec{\omega}_{N/O_N} \times \vec{r}_{TMD_Y/P_N}) = y_{TMD_Y/P_N} \begin{bmatrix} \dot{\theta}_{N/O_N} \dot{\phi}_{N/O_N} \\ -(\dot{\theta}_{N/O_N}^2 + \dot{\psi}_{N/O_N}^2) \\ \dot{\phi}_{N/O_N} \dot{\psi}_{N/O_N} \end{bmatrix}$$

$$2\vec{\omega}_{N/ON} \times \dot{\vec{r}}_{TMD_Y/P_N} = \dot{y}_{TMD_Y/P_N} \begin{bmatrix} -2\dot{\psi}_{N/ON} \\ 0 \\ 2\dot{\theta}_{N/ON} \end{bmatrix}$$

$$\vec{\alpha}_{N/ON} \times \vec{r}_{TMD_Y/P_N} = y_{TMD_Y/P_N} \begin{bmatrix} -\ddot{\psi}_{N/ON} \\ 0 \\ \ddot{\theta}_{N/ON} \end{bmatrix}$$

Therefore \ddot{y}_{TMD_Y/P_N} is governed by the equations

$$\begin{aligned} \ddot{y}_{TMD_Y/P_N} = & (\dot{\theta}_{N/ON}^2 + \dot{\psi}_{N/ON}^2 - \frac{k_y}{m_y}) y_{TMD_Y/P_N} - (\frac{c_y}{m_y}) \dot{y}_{TMD_Y/P_N} - \ddot{y}_{P/ON} + a_{G_Y/ON} \\ & + \frac{1}{m_y} (F_{ext_Y} + F_{StopFrc_Y}) \end{aligned} \quad (4.192)$$

The forces $F_{X_{TMD_Y/ON}}$ and $F_{Z_{TMD_Y/ON}}$ are solved noting $\ddot{x}_{TMD_Y/P_N} = \ddot{z}_{TMD_Y/P_N} = 0$:

$$F_{X_{TMD_Y/ON}} = m_y \left(-a_{G_X/ON} + \ddot{x}_{P/ON} - (\ddot{\psi}_{N/ON} - \dot{\theta}_{N/ON} \dot{\phi}_{N/ON}) y_{TMD_Y/P_N} - 2\dot{\psi}_{N/ON} \dot{y}_{TMD_Y/P_N} \right) \quad (4.193)$$

$$F_{Z_{TMD_Y/ON}} = m_y \left(-a_{G_Z/ON} + \ddot{z}_{P/ON} + (\ddot{\theta}_{N/ON} + \dot{\phi}_{N/ON} \dot{\psi}_{N/ON}) y_{TMD_Y/P_N} + 2\dot{\theta}_{N/ON} \dot{y}_{TMD_Y/P_N} \right) \quad (4.194)$$

TMD_Z :

The external forces $\vec{F}_{TMD_Z/ON}$ are given by

$$\vec{F}_{TMD_Z/ON} = \begin{bmatrix} F_{X_{TMD_Z/ON}} + m_z a_{G_X/ON} \\ F_{Y_{TMD_Z/ON}} + m_z a_{G_Y/ON} \\ -c_z \dot{z}_{TMD_Z/P_N} - k_z z_{TMD_Z/P_N} + m_z a_{G_Z/ON} + F_{ext_z} + F_{StopFrc_Z} + F_{Z_{PreLoad}} \end{bmatrix}$$

where $F_{Z_{PreLoad}}$ is a spring pre-load to shift the neutral position when gravity acts upon the mass for the TMD_Z . TMD_Z is fixed to frame N in the x and y directions so that

$$\vec{r}_{TMD_Z/P_N} = \begin{bmatrix} 0 \\ 0 \\ z_{TMD_Z/P_N} \end{bmatrix}$$

The other components of Eqn. (4.188) are:

$$\vec{\omega}_{N/ON} \times (\vec{\omega}_{N/ON} \times \vec{r}_{TMD_Z/P_N}) = z_{TMD_Z/P_N} \begin{bmatrix} \dot{\theta}_{N/ON} \dot{\psi}_{N/ON} \\ \dot{\phi}_{N/ON} \dot{\psi}_{N/ON} \\ -(\dot{\theta}_{N/ON}^2 + \dot{\phi}_{N/ON}^2) \end{bmatrix}$$

$$2\vec{\omega}_{N/ON} \times \dot{\vec{r}}_{TMD_Z/P_N} = \dot{z}_{TMD_Z/P_N} \begin{bmatrix} 2\dot{\phi}_{N/ON} \\ -2\dot{\theta}_{N/ON} \\ 0 \end{bmatrix}$$

$$\vec{\alpha}_{N/ON} \times \vec{r}_{TMD_Z/P_N} = z_{TMD_Z/P_N} \begin{bmatrix} \ddot{\phi}_{N/ON} \\ -\ddot{\theta}_{N/ON} \\ 0 \end{bmatrix}$$

Therefore \ddot{z}_{TMD_Z/P_N} is governed by the equations

$$\begin{aligned} \ddot{z}_{TMD_Z/P_N} = & (\dot{\theta}_{N/ON}^2 + \dot{\phi}_{N/ON}^2 - \frac{k_z}{m_z}) z_{TMD_Z/P_N} - (\frac{c_z}{m_z}) \dot{z}_{TMD_Z/P_N} - \ddot{z}_{P/ON} + a_{G_Z/ON} \\ & + \frac{1}{m_z} (F_{ext_z} + F_{StopFrc_Z} + F_{Z_{PreLoad}}) \end{aligned} \quad (4.195)$$

The forces $F_{X_{TMDZ/ON}}$ and $F_{Z_{TMDZ/ON}}$ are solved noting $\ddot{x}_{TMDZ/PN} = \ddot{y}_{TMDZ/PN} = 0$:

$$F_{X_{TMDZ/ON}} = m_z \left(-a_{GX/ON} + \ddot{x}_{P/ON} + (\ddot{\phi}_{N/ON} + \dot{\theta}_{N/ON} \dot{\psi}_{N/ON}) z_{TMDZ/PN} + 2\dot{\phi}_{N/ON} \dot{z}_{TMDZ/PN} \right) \quad (4.196)$$

$$F_{Y_{TMDZ/ON}} = m_z \left(-a_{GY/ON} + \ddot{y}_{P/ON} - (\ddot{\theta}_{N/ON} - \dot{\phi}_{N/ON} \dot{\psi}_{N/ON}) z_{TMDZ/PN} - 2\dot{\theta}_{N/ON} \dot{z}_{TMDZ/PN} \right) \quad (4.197)$$

State Equations

Inputs:

The inputs are the component linear acceleration and angular position, velocity and acceleration:

$$\vec{u} = \begin{bmatrix} \ddot{\vec{r}}_{P/OG} \\ \ddot{\vec{R}}_{N/G} \\ \ddot{\vec{\omega}}_{N/OG} \\ \ddot{\vec{\alpha}}_{N/OG} \end{bmatrix} \Rightarrow \begin{bmatrix} \ddot{\vec{r}}_{P/ON} \\ \ddot{\vec{\omega}}_{N/ON} \\ \ddot{\vec{\alpha}}_{N/ON} \end{bmatrix} = \begin{bmatrix} \vec{R}_{N/G} \ddot{\vec{r}}_{P/OG} \\ \vec{R}_{N/G} \ddot{\vec{\omega}}_{N/OG} \\ \vec{R}_{N/G} \ddot{\vec{\alpha}}_{N/OG} \end{bmatrix}$$

States:

The states are the position and velocity of the TMDs along their respective DOFs in the component reference frame:

$$\vec{R}_{TMD/PN} = \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \\ z \\ \dot{z} \end{bmatrix}_{TMD/PN} = \begin{bmatrix} x_{TMDX/PN} \\ \dot{x}_{TMDX/PN} \\ y_{TMDY/PN} \\ \dot{y}_{TMDY/PN} \\ z_{TMDZ/PN} \\ \dot{z}_{TMDZ/PN} \end{bmatrix}$$

The equations of motion can be re-written as a system of non-linear first-order equations of the form

$$\dot{\vec{R}}_{TMD} = A \vec{R}_{TMD} + B$$

where

$$A(\vec{u}) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ (\dot{\phi}_{P/ON}^2 + \dot{\psi}_{P/ON}^2 - \frac{k_x}{m_x}) & -(\frac{c_x}{m_x}) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & (\dot{\theta}_{P/ON}^2 + \dot{\psi}_{P/ON}^2 - \frac{k_y}{m_y}) & -(\frac{c_y}{m_y}) & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & (\dot{\theta}_{P/ON}^2 + \dot{\phi}_{P/ON}^2 - \frac{k_z}{m_z}) & -(\frac{c_z}{m_z}) \end{bmatrix}$$

and

$$B(\vec{u}) = \begin{bmatrix} 0 \\ -\ddot{x}_{P/ON} + a_{GX/ON} + \frac{1}{m_x}(F_{extX} + F_{StopFrcX}) \\ 0 \\ -\ddot{y}_{P/ON} + a_{GY/ON} + \frac{1}{m_y}(F_{extY} + F_{StopFrcY}) \\ 0 \\ -\ddot{z}_{P/ON} + a_{GZ/ON} + \frac{1}{m_z}(F_{extZ} + F_{StopFrcZ} + F_{ZPreLoad}) \end{bmatrix} \quad (4.198)$$

The inputs are coupled to the state variables, resulting in A and B as $f(\vec{u})$.

Outputs

The output vector \vec{Y} is

$$\vec{Y} = \begin{bmatrix} \vec{F}_{PG} \\ \vec{M}_{PG} \end{bmatrix}$$

The output includes reaction forces corresponding to $F_{Y_{TMDX/ON}}$, $F_{Z_{TMDX/ON}}$, $F_{X_{TMDY/ON}}$, $F_{Z_{TMDY/ON}}$, $F_{X_{TMDZ/ON}}$, and $F_{Y_{TMDZ/ON}}$ from Eqns. (4.190), (4.191), (4.193), (4.194), (4.196), and (4.197). The resulting forces \vec{F}_{PG} and moments \vec{M}_{PG} acting on the component are

$$\vec{F}_{PG} = R_{N/G}^T \begin{bmatrix} k_x x_{TMD/PN} + c_x \dot{x}_{TMD/PN} - F_{StopFrcX} - F_{ext_x} - F_{X_{TMDY/ON}} - F_{X_{TMDZ/ON}} \\ k_y y_{TMD/PN} + c_y \dot{y}_{TMD/PN} - F_{StopFrcY} - F_{ext_y} - F_{Y_{TMDX/ON}} - F_{Y_{TMDZ/ON}} \\ k_z z_{TMD/PN} + c_z \dot{z}_{TMD/PN} - F_{StopFrcZ} - F_{ext_z} - F_{Z_{TMDX/ON}} - F_{Z_{TMDY/ON}} - F_{Z_{PreLoad}} \end{bmatrix} \quad (4.199)$$

and

$$\vec{M}_{PG} = R_{N/G}^T \begin{bmatrix} M_X \\ M_Y \\ M_Z \end{bmatrix}_{N/N} = R_{N/G}^T \begin{bmatrix} -(F_{Z_{TMDY/ON}})y_{TMD/PN} + (F_{Y_{TMDZ/ON}})z_{TMD/PN} \\ (F_{Z_{TMDX/ON}})x_{TMD/PN} - (F_{X_{TMDZ/ON}})z_{TMD/PN} \\ -(F_{Y_{TMDX/ON}})x_{TMD/PN} + (F_{X_{TMDY/ON}})y_{TMD/PN} \end{bmatrix}$$

Stop Forces

The extra forces $F_{StopFrcX}$, $F_{StopFrcY}$, and $F_{StopFrcZ}$ are added to output forces in the case that the movement of TMD_X, TMD_Y, or TMD_Z exceeds the maximum track length for the mass. Otherwise, they equal zero. The track length has limits on the positive and negative ends in the TMD direction (X_PSP and X_NSP, Y_PSP and Y_NSP, and Z_PSP and Z_NSP). If we define a general maximum and minimum displacements as x_{max} and x_{min} , respectively, the stop forces have the form

$$F_{StopFrc} = - \begin{cases} k_S \Delta x & : (x > x_{max} \wedge \dot{x} \leq 0) \vee (x < x_{min} \wedge \dot{x} \geq 0) \\ k_S \Delta x + c_S \dot{x} & : (x > x_{max} \wedge \dot{x} > 0) \vee (x < x_{min} \wedge \dot{x} < 0) \\ 0 & : \text{otherwise} \end{cases}$$

where Δx is the distance the mass has traveled beyond the stop position and k_S and c_S are large stiffness and damping constants.

Pre-Load Forces

The extra force $F_{Z_{PreLoad}}$ is added to the output forces as a method to shift the at rest position of the TMD_Z when gravity is acting on it. This is particularly useful for substructure mounted StCs when very large masses with soft spring constants are used. This appears in the term $\vec{F}_{TMDZ/ON}$ and in eq equations of motion given by (4.198) and resulting forces in (4.199).

Code Modifications

The Structural Control (StC) function is a submodule linked into ServoDyn. In addition to references in ServoDyn.f90 and ServoDyn.txt, new files that contain the StC module are listed below.

New Files

- StrucCtrl.f90 : the structural control module
- StrucCtrl.txt : registry file include files, inputs, states, parameters, and outputs shown in Tables 1 and 2
- StrucCtrl_Types.f90: automatically generated

Variables

Table 4.13: Summary of field definitions in the StC registry. Note that state vector \vec{tmd}_x corresponds to \vec{R}_{TMD/P_N} , and that the outputs \vec{F}_{P_G} and \vec{M}_{P_G} are contained in the MeshType object (y.Mesh). X_{DSP} , Y_{DSP} , and Z_{DSP} are initial displacements of the TMDs.

Data Type	Variable name
InitInput	
	InputFile
	Gravity
	\vec{r}_{N/O_G}
Input u	
	$\ddot{\vec{r}}_{P/O_G}$
	\vec{R}_{N/O_G}
	$\vec{\omega}_{N/O_G}$
	$\vec{\alpha}_{N/O_G}$
Parameter p	
	m_x
	c_x
	k_x
	m_y
	c_y
	k_y
	m_z
	c_z
	k_z
	$K_S = [k_{SX} \ k_{SY} \ k_{SZ}]$
	$C_S = [c_{SX} \ c_{SY} \ c_{SZ}]$
	$P_{SP} = [X_{PSP} \ Y_{PSP} \ Z_{PSP}]$
	$P_{NSP} = [X_{NSP} \ Y_{NSP} \ Z_{NSP}]$
	F_{ext}
	Gravity
	TMDX_DOF
	TMDY_DOF
	TMDZ_DOF

continues on next page

Table 4.13 – continued from previous page

Data Type	Variable name
	X_{DSP}
	Y_{DSP}
	Z_{DSP}
State x	
	\vec{tmd}_x
Output y	
	Mesh

The input, parameter, state and output definitions are summarized in Table 1. The inputs from file are listed in Table 2.

Table 4.14: Data read in from TMDInputFile.

Field Name	Field Type	Description
TMD_CMODE	int	Control Mode (1:passive, 2:active)
TMD_X_DOF	logical	DOF on or off
TMD_Y_DOF	logical	DOF on or off
TMD_Z_DOF	logical	DOF on or off
TMD_X_DSP	real	TMD_X initial displacement
TMD_Y_DSP	real	TMD_Y initial displacement
TMD_Z_DSP	real	TMD_Z initial displacement
TMD_X_M	real	TMD mass
TMD_X_K	real	TMD stiffness
TMD_X_C	real	TMD damping
TMD_Y_M	real	TMD mass
TMD_Y_K	real	TMD stiffness
TMD_Y_C	real	TMD damping
TMD_Z_M	real	TMD mass
TMD_Z_K	real	TMD stiffness
TMD_Z_C	real	TMD damping
TMD_X_PSP	real	positive stop position (maximum X mass displacement)
TMD_X_NSP	real	negative stop position (minimum X mass displacement)
TMD_X_K_SX	real	stop spring stiffness
TMD_X_C_SX	real	stop spring damping
TMD_Y_PSP	real	positive stop position (maximum Y mass displacement)
TMD_Y_NSP	real	negative stop position (minimum Y mass displacement)
TMD_Y_K_S	real	stop spring stiffness
TMD_Y_C_S	real	stop spring damping
TMD_Z_PSP	real	positive stop position (maximum Z mass displacement)
TMD_Z_NSP	real	negative stop position (minimum Z mass displacement)
TMD_Z_K_S	real	stop spring stiffness
TMD_Z_C_S	real	stop spring damping
TMD_P_X	real	x origin of P in nacelle coordinate system
TMD_P_Y	real	y origin of P in nacelle coordinate system
TMD_P_Z	real	z origin of P in nacelle coordinate system

Acknowledgements

The authors would like to thank Dr. Jason Jonkman for reviewing this manual.

TLCD: Derivations of Equation of Motion

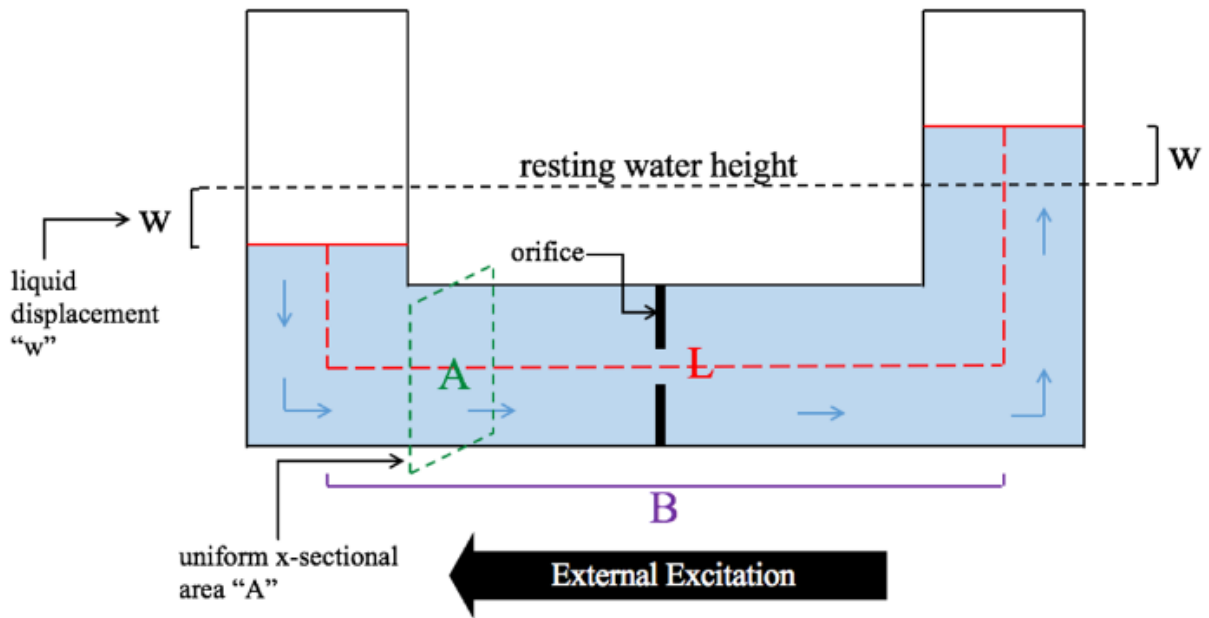


Fig. 4.44: Schematic of TLCD design.

Definitions:

Table 4.15: TLCD Definitions

Variable	Description
O	origin point of global inertial reference frame, located at center of base of resting turbine
P	origin point of local reference frame (e.g. fixed to nacelle), in the center of the horizontal liquid column
W_R	point attached to the top center of the right liquid column (moving)
W_L	point attached to the top center of the left liquid column (moving)
i	axis orientation of inertial reference frame (global)
l	axis orientation of local reference frame
w	position of the liquid water column as defined in Figure Fig. 4.44
g	gravity vector in the inertial reference frame (global)

Right Vertical Liquid Column

Starting with the right vertical column, we define the following vector expressions:

Variable	Description
$\vec{r}_i^{O \rightarrow P} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}_i^{O \rightarrow P}$	position vector from point O to point P in inertial coordinate system
$\vec{r}_l^{P \rightarrow W_R} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}_l^{P \rightarrow W_R}$	position vector from point P to point W_R in local coordinate system
$\vec{\omega}_i^l = \begin{bmatrix} \theta \\ \phi \\ \psi \end{bmatrix}_i$	angular velocity frame l with respect to inertial reference frame i
$\vec{r}_i^{O \rightarrow W_R} = \vec{r}_i^{O \rightarrow P} + \vec{r}_l^{P \rightarrow W_R}$	position vector from point P to point W_R in local coordinate system

Taking the derivative of the last expression for $\vec{r}_i^{O \rightarrow W_R}$ yields the velocity of point W_R in the global reference frame:

$$\dot{\vec{r}}_i^{W_R} = \dot{\vec{r}}_i^P + \dot{\vec{r}}_l^{W_R} + \vec{\omega}_i^l \times \vec{r}_l^{P \rightarrow W_R}.$$

Repeating this step once more yields its acceleration:

$$\ddot{\vec{r}}_i^{W_R} = \ddot{\vec{r}}_i^P + \ddot{\vec{r}}_l^{W_R} + 2\vec{\omega}_i^l \times \dot{\vec{r}}_l^{W_R} + \dot{\vec{\omega}}_i^l \times \vec{r}_l^{P \rightarrow W_R} + \vec{\omega}_i^l \times (\vec{\omega}_i^l \times \vec{r}_l^{P \rightarrow W_R})$$

Following Newton's Second Law, the left part of this expression can be replaced with a force balance:

$$\ddot{\vec{r}}_i^{W_R} = \frac{1}{m_R} \begin{bmatrix} \sum F_x \\ \sum F_y \\ \sum F_z \end{bmatrix}^{W_R} = \frac{1}{m_R} \begin{bmatrix} F_x^{W_R/S} + m_R g_x \\ F_y^{W_R/S} + m_R g_y \\ m_R g_z \end{bmatrix}^{W_R}$$

where g is the gravity vector in the inertial frame. The vector describing the position of the right hand column in the local reference frame (i) can be written as:

$$\vec{r}_l^{P \rightarrow W_R} = \begin{bmatrix} B/2 \\ 0 \\ \frac{L-B}{2} + w \end{bmatrix}_l^{P \rightarrow W_R}.$$

Movement of the liquid in the vertical columns is restricted to the z-direction in reference frame N, thus the expression

for the acceleration of the right liquid column becomes:

$$\begin{aligned} \frac{1}{m_R} \begin{bmatrix} F_x^{W_R/S} + m_R g_x \\ F_y^{W_R/S} + m_R g_y \\ m_R g_z \end{bmatrix}^{W_R} &= \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix}_i^P + \begin{bmatrix} 0 \\ 0 \\ \ddot{\omega} \end{bmatrix}_l^{W_R} \\ &+ 2 \begin{bmatrix} \dot{\theta} \\ \dot{\phi} \\ \dot{\psi} \end{bmatrix}_i^l \times \begin{bmatrix} 0 \\ 0 \\ \dot{\omega} \end{bmatrix}_l^{W_R} \\ &+ \begin{bmatrix} \ddot{\theta} \\ \ddot{\phi} \\ \ddot{\psi} \end{bmatrix}_i^l \times \begin{bmatrix} B/2 \\ 0 \\ \frac{L-B}{2} + w \end{bmatrix}_l^{P \rightarrow W_R} \\ &+ \begin{bmatrix} \dot{\theta} \\ \dot{\phi} \\ \dot{\psi} \end{bmatrix}_i^l \times \left(\begin{bmatrix} \dot{\theta} \\ \dot{\phi} \\ \dot{\psi} \end{bmatrix}_i^l \times \begin{bmatrix} B/2 \\ 0 \\ \frac{L-B}{2} + w \end{bmatrix}_l^{P \rightarrow W_R} \right) \end{aligned}$$

Computing all cross-products yields three distinct expressions in the x, y, and z dimensions:

$$\begin{aligned} x : \quad & \frac{1}{m_R} (F_x^{W_R/S} + m_R g_x) \\ &= \ddot{x}_i^P + 2\dot{\phi}\dot{w} + \ddot{\phi} \left(\frac{L-B}{2} + w \right) - \dot{\phi}^2 \frac{B}{2} - \dot{\psi}^2 \frac{B}{2} + \dot{\psi}\dot{\theta} \left(\frac{L-B}{2} + w \right) \\ y : \quad & \frac{1}{m_R} (F_y^{W_R/S} + m_R g_y) \\ &= \ddot{y}_i^P - 2\dot{\theta}\dot{w} + \ddot{\psi} \frac{B}{2} - \ddot{\theta} \left(\frac{L-B}{2} + w \right) + \dot{\psi}\dot{\phi} \left(\frac{L-B}{2} + w \right) + \dot{\theta}\dot{\phi} \frac{B}{2} \\ z : \quad & g_z \\ &= \ddot{z}_i^P + \ddot{w} - \ddot{\phi} \frac{B}{2} + \dot{\theta}\dot{\psi} \frac{B}{2} - \dot{\theta}^2 \left(\frac{L-B}{2} + w \right) - \dot{\phi}^2 \left(\frac{L-B}{2} + w \right) \end{aligned}$$

Left Vertical Liquid Column

Following the same methodology as above the equations describing the movement of the left vertical liquid column can be determined.

Similarly, the acceleration of the left liquid column can be replaced by a force balance:

$$\ddot{\vec{r}}_i^{W_L} = \frac{1}{m_L} \begin{bmatrix} \sum F_x \\ \sum F_y \\ \sum F_z \end{bmatrix}^{W_L} = \frac{1}{m_L} \begin{bmatrix} F_x^{W_L/S} + m_L g_x \\ F_y^{W_L/S} + m_L g_y \\ m_L g_z \end{bmatrix}^{W_L}$$

where g is the gravity vector in the inertial frame. The vector describing the position of the left hand column in the local reference frame (i) can be written as:

$$\vec{r}_l^{P \rightarrow W_L} = \begin{bmatrix} -B/2 \\ 0 \\ \frac{L-B}{2} - w \end{bmatrix}_l^{P \rightarrow W_L}.$$

The final equation for the acceleration of the left liquid column becomes:

$$\begin{aligned} \frac{1}{m_L} \begin{bmatrix} F_x^{W_L/S} + m_L g_x \\ F_y^{W_L/S} + m_L g_y \\ m_L g_z \end{bmatrix}^{W_L} &= \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix}_i^P + \begin{bmatrix} 0 \\ 0 \\ -\ddot{w} \end{bmatrix}_l^{W_L} \\ &+ 2 \begin{bmatrix} \dot{\theta} \\ \dot{\phi} \\ \dot{\psi} \end{bmatrix}_i^l \times \begin{bmatrix} 0 \\ 0 \\ -\dot{w} \end{bmatrix}_l^{W_L} \\ &+ \begin{bmatrix} \ddot{\theta} \\ \ddot{\phi} \\ \ddot{\psi} \end{bmatrix}_i^l \times \begin{bmatrix} -B/2 \\ 0 \\ \frac{L-B}{2} - w \end{bmatrix}_l^{P \rightarrow W_L} \\ &+ \begin{bmatrix} \dot{\theta} \\ \dot{\phi} \\ \dot{\psi} \end{bmatrix}_i^l \times \left(\begin{bmatrix} \dot{\theta} \\ \dot{\phi} \\ \dot{\psi} \end{bmatrix}_i^l \times \begin{bmatrix} -B/2 \\ 0 \\ \frac{L-B}{2} - w \end{bmatrix}_l^{P \rightarrow W_L} \right) \end{aligned}$$

The x, y, and z equations then become:

$$\begin{aligned} x : \quad & \frac{1}{m_L} (F_x^{W_L/S} + m_L g_x) \\ &= \ddot{x}_i^P - 2\dot{\phi}\dot{w} + \ddot{\phi} \left(\frac{L-B}{2} - w \right) + \dot{\phi}^2 \frac{B}{2} + \dot{\psi}^2 \frac{B}{2} + \dot{\psi}\dot{\theta} \left(\frac{L-B}{2} - w \right) \\ y : \quad & \frac{1}{m_L} (F_y^{W_L/S} + m_L g_y) \\ &= \ddot{y}_i^P + 2\dot{\theta}\dot{w} - \ddot{\psi} \frac{B}{2} - \ddot{\theta} \left(\frac{L-B}{2} - w \right) + \dot{\psi}\dot{\phi} \left(\frac{L-B}{2} - w \right) - \dot{\theta}\dot{\phi} \frac{B}{2} \\ z : \quad & g_z \\ &= \ddot{z}_i^P - \ddot{w} + \ddot{\phi} \frac{B}{2} - \dot{\theta}\dot{\psi} \frac{B}{2} - \dot{\theta}^2 \left(\frac{L-B}{2} - w \right) - \dot{\phi}^2 \left(\frac{L-B}{2} - w \right) \end{aligned}$$

Horizontal Liquid Column

As the movement of the liquid in the horizontal column (H) is restricted to the x-dimension in local reference frame, l , the position vector can be expressed as:

$$\vec{r}_l^{P \rightarrow W_H} = \begin{bmatrix} w \\ 0 \\ 0 \end{bmatrix}_l^{P \rightarrow W_H}.$$

Furthermore, the force balance on the horizontal liquid column is...

$$\ddot{\vec{r}}_i^{W_H} = \frac{1}{m_H} \begin{bmatrix} \sum F_x \\ \sum F_y \\ \sum F_z \end{bmatrix}^{W_H} = \frac{1}{m_H} \begin{bmatrix} m_H g_x - \frac{1}{2} \rho A \xi |\dot{w}| \dot{w} \\ F_y^{W_H/S} + m_H g_y \\ F_z^{W_H/S} + m_H g_z \end{bmatrix}^{W_H}$$

where the ρ term represents the damping force applied to the liquid as it passes through the restricted orifice, and g is the gravity vector in the inertial frame.

The final expression for the acceleration of the water through the horizontal column becomes:

$$\begin{aligned} \frac{1}{m_H} \begin{bmatrix} m_H g_x - \frac{1}{2} \rho A \xi |\dot{w}| \dot{w} \\ F_y^{W_H/S} + m_H g_y \\ F_z^{W_H/S} + m_H g_z \end{bmatrix}^{W_H} &= \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix}_i^P + \begin{bmatrix} \ddot{w} \\ 0 \\ 0 \end{bmatrix}_l^{W_H} \\ &+ 2 \begin{bmatrix} \dot{\theta} \\ \dot{\phi} \\ \dot{\psi} \end{bmatrix}_i^l \times \begin{bmatrix} \dot{w} \\ 0 \\ 0 \end{bmatrix}_l^{W_H} \\ &+ \begin{bmatrix} \ddot{\theta} \\ \ddot{\phi} \\ \ddot{\psi} \end{bmatrix}_i^l \times \begin{bmatrix} w \\ 0 \\ 0 \end{bmatrix}_l^{P \rightarrow W_H} \\ &+ \begin{bmatrix} \dot{\theta} \\ \dot{\phi} \\ \dot{\psi} \end{bmatrix}_i^l \times \left(\begin{bmatrix} \dot{\theta} \\ \dot{\phi} \\ \dot{\psi} \end{bmatrix}_i^l \times \begin{bmatrix} w \\ 0 \\ 0 \end{bmatrix}_l^{P \rightarrow W_H} \right) \end{aligned}$$

The x, y, and z equations thus become:

$$\begin{aligned} x : \quad & g_x - \frac{1}{m_H} \left(\frac{1}{2} \rho A \xi |\dot{w}| \dot{w} \right) \\ &= \ddot{x}_i^P + \ddot{w} - \dot{\phi}^2 w - \dot{\psi}^2 w \\ y : \quad & \frac{1}{m_H} \left(F_y^{W_H/S} + m_H g_y \right) \\ &= \ddot{y}_i^P + 2\dot{\psi}\dot{w} + \ddot{\psi}w + \dot{\theta}\dot{\phi}w \\ z : \quad & \frac{1}{m_H} \left(F_z^{W_H/S} + m_H g_z \right) \\ &= \ddot{z}_i^P - \dot{\phi}\dot{w} - \ddot{\phi}w + \dot{\theta}\dot{\psi}w \end{aligned}$$

Recalling that the displacement of the liquid in the horizontal column, w is zero, as the center of mass of the liquid in the horizontal column always remains at point l even as liquid accelerates through the pipe. Consequently, removing the w terms from these equations gives the expressions:

$$\begin{aligned} x : \quad & g_x - \frac{1}{m_H} \left(\frac{1}{2} \rho A \xi |\dot{w}| \dot{w} \right) \\ &= \ddot{x}_i^P + \ddot{w} \\ y : \quad & \frac{1}{m_H} \left(F_y^{W_H/S} + m_H g_y \right) \\ &= \ddot{y}_i^P + 2\dot{\psi}\dot{w} \\ z : \quad & \frac{1}{m_H} \left(F_z^{W_H/S} + m_H g_z \right) \\ &= \ddot{z}_i^P - \dot{\phi}\dot{w} \end{aligned}$$

Now that the accelerations of the three liquid columns have been determined individually, we can extract the inertial forces and derive a singular equation to describe the acceleration of the liquid in the column.

The inertial forces are then written as:

$$\begin{aligned}
F_x^{W_R/S} &= m_R \left(\ddot{x}_i^P + 2\dot{\phi}\dot{w} + \ddot{\phi} \left(\frac{L-B}{2} + w \right) - \dot{\phi}^2 \frac{B}{2} - \dot{\psi}^2 \frac{B}{2} + \dot{\psi}\dot{\phi} \left(\frac{L-B}{2} + w \right) - g_x \right) \\
F_y^{W_R/S} &= m_R \left(\ddot{y}_i^P - 2\dot{\theta}\dot{w} + \ddot{\psi} \frac{B}{2} - \ddot{\theta} \left(\frac{L-B}{2} + w \right) + \dot{\psi}\dot{\phi} \left(\frac{L-B}{2} + w \right) + \dot{\theta}\dot{\phi} \frac{B}{2} - g_y \right) \\
F_x^{W_L/S} &= m_L \left(\ddot{x}_i^P - 2\dot{\phi}\dot{w} + \ddot{\phi} \left(\frac{L-B}{2} - w \right) + \dot{\phi}^2 \frac{B}{2} + \dot{\psi}^2 \frac{B}{2} + \dot{\psi}\dot{\phi} \left(\frac{L-B}{2} - w \right) - g_x \right) \\
F_y^{W_L/S} &= m_L \left(\ddot{y}_i^P + 2\dot{\theta}\dot{w} - \ddot{\psi} \frac{B}{2} - \ddot{\theta} \left(\frac{L-B}{2} - w \right) + \dot{\psi}\dot{\phi} \left(\frac{L-B}{2} - w \right) - \dot{\theta}\dot{\phi} \frac{B}{2} - g_y \right) \\
F_y^{W_H/S} &= m_H \left(\ddot{y}_i^P + 2\dot{\psi}\dot{w} - g_y \right) \\
F_z^{W_H/S} &= m_H \left(\ddot{z}_i^P - \dot{\phi}\dot{w} - g_z \right)
\end{aligned}$$

Equation for \ddot{w} from right liquid column (z-dimension):

$$\ddot{w} = -\ddot{z}_i^P + \ddot{\phi} \frac{B}{2} - \dot{\theta}\dot{\psi} \frac{B}{2} + \dot{\theta}^2 \left(\frac{L-B}{2} + w \right) + \dot{\phi}^2 \left(\frac{L-B}{2} + w \right) + g_z$$

Equation for \ddot{w} from left liquid column (z-dimension):

$$\ddot{w} = \ddot{z}_i^P + \ddot{\phi} \frac{B}{2} - \dot{\theta}\dot{\psi} \frac{B}{2} - \dot{\theta}^2 \left(\frac{L-B}{2} - w \right) - \dot{\phi}^2 \left(\frac{L-B}{2} - w \right) - g_z$$

Equation for \ddot{w} from horizontal liquid column (x-dimension):

$$\ddot{w} = -\ddot{x}_i^P + g_x - \frac{1}{m_H} \left(\frac{1}{2} \rho A \xi |\dot{w}| \dot{w} \right)$$

From Newton's Second Law, we know that the acceleration of the total liquid mass can be described accordingly:

$$m_T \ddot{w} = m_R (\ddot{w}) m_L (\ddot{w}) m_H (\ddot{w})$$

Where

$$\begin{aligned}
m_T &= \rho A L \\
m_R &= \rho A \left(\frac{L-B}{2} + w \right) \\
m_L &= \rho A \left(\frac{L-B}{2} - w \right) \\
m_H &= \rho A B
\end{aligned}$$

Combining the above equations gives us the expression:

$$\begin{aligned}
\rho A L \ddot{w} &= \rho A \left(\frac{L-B}{2} + w \right) \left[-\ddot{z}_i^P + \ddot{\phi} \frac{B}{2} - \dot{\theta}\dot{\psi} \frac{B}{2} \right. \\
&\quad \left. + \dot{\theta}^2 \left(\frac{L-B}{2} + w \right) + \dot{\phi}^2 \left(\frac{L-B}{2} + w \right) + g_z \right] \\
&\quad + \rho A B \left(-\ddot{x}_i^P + g_x - \frac{1}{m_H} \left(\frac{1}{2} \rho A \xi |\dot{w}| \dot{w} \right) \right)
\end{aligned}$$

Finally, simplifying this expression gives us the final equation, describing the movement of the liquid through the

TLCD:

$$\begin{aligned}
 \rho AL\ddot{w} = & -2\rho Aw\ddot{z}_i^P + \rho AB\ddot{\phi}\left(\frac{L-B}{2}\right) - \rho AB\dot{\theta}\dot{\psi}\left(\frac{L-B}{2}\right) \\
 & + 2\rho Aw\dot{\theta}^2(L-B) + 2\rho Aw\dot{\phi}^2(L-B) \\
 & + 2\rho Awg_z - \rho AB\ddot{z}_i^P + \rho ABg_x \\
 & - \frac{1}{2}\rho AB\xi|\dot{w}|\dot{w}
 \end{aligned}$$

Orthogonal TLCD

Following the same methodology as above in the side-side orientation (as opposed to fore-aft) yields the following equations for the front, back, and horizontal orthogonal columns:

Back Vertical Orthogonal Liquid Column

$$\begin{aligned}
 x : \quad & \frac{1}{m_B} \left(F_x^{W_B/S} + m_B g_x \right) \\
 = & \ddot{z}_i^P + 2\dot{\phi}\dot{w}_o + \ddot{\phi} \left(\frac{L-B}{2} + w_o \right) + \dot{\psi}^2 \frac{B}{2} - \dot{\phi}\dot{\theta} \frac{B}{2} + \dot{\psi}\dot{\theta} \left(\frac{L-B}{2} + w_o \right) \\
 y : \quad & \frac{1}{m_B} \left(F_y^{W_B/S} + m_B g_y \right) \\
 = & \ddot{y}_i^P - 2\dot{\theta}\dot{w}_o - \ddot{\theta} \left(\frac{L-B}{2} + w_o \right) + \dot{\psi}\dot{\phi} \left(\frac{L-B}{2} + w_o \right) + \dot{\psi}^2 \frac{B}{2} + \dot{\theta}^2 \frac{B}{2} \\
 z : \quad & g_z \\
 = & \ddot{z}_i^P + \ddot{w}_o - \ddot{\theta} \frac{B}{2} - \dot{\theta}^2 \left(\frac{L-B}{2} + w_o \right) - \dot{\phi}^2 \left(\frac{L-B}{2} + w_o \right) - \dot{\phi}\dot{\psi} \frac{B}{2}
 \end{aligned}$$

Front Vertical Orthogonal Liquid Column

$$\begin{aligned}
 x : \quad & \frac{1}{m_F} \left(F_x^{W_F/S} + m_F g_x \right) \\
 = & \ddot{z}_i^P - 2\dot{\phi}\dot{w}_o + \ddot{\phi} \left(\frac{L-B}{2} - w_o \right) - \dot{\psi}^2 \frac{B}{2} + \dot{\phi}\dot{\theta} \frac{B}{2} + \dot{\psi}\dot{\theta} \left(\frac{L-B}{2} - w_o \right) \\
 y : \quad & \frac{1}{m_F} \left(F_y^{W_F/S} + m_F g_y \right) \\
 = & \ddot{y}_i^P + 2\dot{\theta}\dot{w}_o - \ddot{\theta} \left(\frac{L-B}{2} - w_o \right) + \dot{\psi}\dot{\phi} \left(\frac{L-B}{2} - w_o \right) - \dot{\psi}^2 \frac{B}{2} - \dot{\theta}^2 \frac{B}{2} \\
 z : \quad & g_z \\
 = & \ddot{z}_i^P - \ddot{w}_o + \ddot{\theta} \frac{B}{2} - \dot{\theta}^2 \left(\frac{L-B}{2} - w_o \right) - \dot{\phi}^2 \left(\frac{L-B}{2} - w_o \right) + \dot{\phi}\dot{\psi} \frac{B}{2}
 \end{aligned}$$

Horizontal Orthogonal Liquid Column

$$\begin{aligned}
 x : \quad & \frac{1}{m_H} \left(F_x^{W_H/S} + m_H g_x \right) \\
 & = \ddot{x}_i^P - 2\dot{\psi}\dot{w}_o \\
 y : \quad & \frac{1}{m_H} \left(m_H g_y - \frac{1}{2} \rho A \xi |\dot{w}_o| \dot{w}_o \right) \\
 & = \ddot{y}_i^P + \ddot{w}_o \\
 z : \quad & \frac{1}{m_H} \left(F_z^{W_H/S} + m_H g_z \right) \\
 & = \ddot{z}_i^P + 2\dot{\theta}\dot{w}_o
 \end{aligned}$$

Extracting the inertial forces from these equations leaves us with:

$$\begin{aligned}
 F_x^{W_B/S} &= m_B \left(\ddot{x}_i^P + 2\dot{\phi}\dot{w}_o + \ddot{\phi} \left(\frac{L-B}{2} + w_o \right) + \ddot{\psi} \frac{B}{2} - \dot{\phi}\dot{\theta} \frac{B}{2} + \dot{\psi}\dot{\theta} \left(\frac{L-B}{2} + w_o \right) - g_x \right) \\
 F_y^{W_B/S} &= m_B \left(\ddot{y}_i^P - 2\dot{\theta}\dot{w}_o - \ddot{\theta} \left(\frac{L-B}{2} + w_o \right) + \dot{\psi}\dot{\phi} \left(\frac{L-B}{2} + w_o \right) + \dot{\psi}^2 \frac{B}{2} + \dot{\theta}^2 \frac{B}{2} - g_y \right) \\
 F_x^{W_F/S} &= m_F \left(\ddot{x}_i^P - 2\dot{\phi}\dot{w}_o + \ddot{\phi} \left(\frac{L-B}{2} - w_o \right) - \ddot{\psi} \frac{B}{2} + \dot{\phi}\dot{\theta} \frac{B}{2} + \dot{\psi}\dot{\theta} \left(\frac{L-B}{2} - w_o \right) - g_x \right) \\
 F_y^{W_F/S} &= m_F \left(\ddot{y}_i^P + 2\dot{\theta}\dot{w}_o - \ddot{\theta} \left(\frac{L-B}{2} - w_o \right) + \dot{\psi}\dot{\phi} \left(\frac{L-B}{2} - w_o \right) - \dot{\psi}^2 \frac{B}{2} - \dot{\theta}^2 \frac{B}{2} - g_y \right) \\
 F_x^{W_H/S} &= m_H \left(\ddot{x}_i^P - 2\dot{\psi}\dot{w}_o - g_x \right) \\
 F_z^{W_H/S} &= m_H \left(\ddot{z}_i^P + 2\dot{\theta}\dot{w}_o - g_z \right)
 \end{aligned}$$

The remaining equations, when combined, yield the final equation:

$$\begin{aligned}
 \rho A L \ddot{w} &= \rho A \left(\frac{L-B}{2} + w_o \right) \left[-\ddot{z}_i^P + \ddot{\theta} \frac{B}{2} + \dot{\theta}^2 \left(\frac{L-B}{2} + w_o \right) \right. \\
 &\quad \left. + \dot{\phi}^2 \left(\frac{L-B}{2} + w_o \right) + \dot{\phi}\dot{\psi} \frac{B}{2} + g_z \right] \\
 &+ \rho A \left(\frac{L-B}{2} - w_o \right) \left[\ddot{z}_i^P + \ddot{\theta} \frac{B}{2} - \dot{\theta}^2 \left(\frac{L-B}{2} - w_o \right) \right. \\
 &\quad \left. - \dot{\phi}^2 \left(\frac{L-B}{2} - w_o \right) + \dot{\phi}\dot{\psi} \frac{B}{2} - g_z \right] \\
 &+ \rho A B \left(-\ddot{y}_i^P + g_y \right) - \frac{1}{2} \rho A \xi |\dot{w}_o| \dot{w}_o
 \end{aligned}$$

Which can be simplified to become:

$$\begin{aligned}
 \rho A L \ddot{w}_o &= -2\rho A w_o \ddot{z}_i^P + \rho A B \ddot{\theta} \left(\frac{L-B}{2} \right) - \rho A B \dot{\phi}\dot{\psi} \left(\frac{L-B}{2} \right) \\
 &+ 2\rho A w_o \dot{\theta}^2 (L-B) + 2\rho A w_o \dot{\phi}^2 (L-B) \\
 &+ 2\rho A w_o g_z - \rho A B \ddot{y}_i^P + \rho A B g_y \\
 &- \frac{1}{2} \rho A B \xi |\dot{w}_o| \dot{w}_o
 \end{aligned}$$

4.2.12 TurbSim Users Guide Placeholder

Output Files

HAWC Full-Field Files

When TurbSim is requested to write HAWC-formatted output files (`WrHAWCFF=TRUE`), it will generate four files. `<RootName>-u.bin`, `<RootName>-v.bin`, and `<RootName>-w.bin` are binary files that contain the full-field turbulence data for the 3 wind-speed components. `<RootName>.HAWC` is a text summary file that indicates the number of points in the binary files and how they should be scaled. The data in this file is written in a format that can be copied into a HAWC2 input file.

Notes:

1. The `factor_scaling` values in the summary file indicate the inverse of the values TurbSim used to scale the data in the HAWC files. `factor_scaling` can theoretically be used in HAWC2 to obtain the original data generated by TurbSim. TurbSim scales the data so that HAWC2 will obtain the standard-deviation ratios of 1.0 (u/u), 0.8 (v/u), and 0.5 (w/u) as a work-around for an issue in how HAWC2 scales turbulence files. Please note that these ratios may not work well with non-IEC turbulence models.
2. HAWC-formatted files are always periodic, so all of the analysis time steps are written.
3. The u-component wind speed files have the mean *hub-height* wind speed removed, so they will contain any shear that was defined.
4. For HAWC2 simulations, it is recommended that TurbSim be run without shear (`PLExp=0`) and without any mean flow angles (`VFlowAng=0`, `HFlowAng=0`). These values can instead be added in the HAWC2 input file or in the InflowWind input file for OpenFAST.

Appendix

TurbSim Input Files

1) Primary TurbSim Input Files: (TurbSim input file example):

This is the primary input file for TurbSim. Most simulations will require only this file.

2) TurbSim secondary input files for user-defined input

Input files that can be specified in the primary input file to import user-defined data.

(user-defined profiles example):

(user-defined spectra example):

(user-defined time-series example):

4.2.13 C++ API Users Guide

The C++ API provides a high level API to run OpenFAST through a C++ gluecode. The primary purpose of the C++ API is to help interface OpenFAST to external programs like CFD solvers that are typically written in C++. The installation of C++ API is enabled via CMake by turning on the `BUILD_OPENFAST_CPP_API` flag.

A sample glue-code `FAST_Prog.cpp` is provided as a demonstration of the usage of the C++ API. The glue-code allows for the simulation of multiple turbines using OpenFAST in parallel over multiple processors. The glue-code takes a single input file named `cDriver.i` ([download](#)).

```

# -*- mode: yaml -*-
#
# C++ glue-code for OpenFAST - Example input file
#

#Total number of turbines in the simulation
nTurbinesGlob: 3
#Enable debug outputs if set to true
debug: False
#The simulation will not run if dryRun is set to true
dryRun: False
#Flag indicating whether the simulation starts from scratch or restart
simStart: init # init/trueRestart/restartDriverInitFAST
#Start time of the simulation
tStart: 0.0
#End time of the simulation. tEnd <= tMax
tEnd: 1.0
#Max time of the simulation
tMax: 4.0
#Time step for FAST. All turbines should have the same time step.
dtFAST: 0.00625
#Restart files will be written every so many time steps
nEveryCheckPoint: 160

Turbine0:
  #The position of the turbine base for actuator-line simulations
  turbine_base_pos: [ 0.0, 0.0, 0.0 ]
  #The number of actuator points along each blade for actuator-line simulations
  num_force_pts_blade: 0
  #The number of actuator points along the tower for actuator-line simulations.
  num_force_pts_tower: 0
  #The checkpoint file for this turbine when restarting a simulation
  restart_filename: "banana"
  #The FAST input file for this turbine
  FAST_input_filename: "t1_Test05.fst"
  #A unique turbine id for each turbine
  turb_id: 1

Turbine1:
  turbine_base_pos: [ 0.0, 0.0, 0.0 ]
  num_force_pts_blade: 0
  num_force_pts_tower: 0
  restart_filename: "banana"
  FAST_input_filename: "t2_Test05.fst"
  turb_id: 2

Turbine2:
  turbine_base_pos: [ 0.0, 0.0, 0.0 ]
  num_force_pts_blade: 0
  num_force_pts_tower: 0
  restart_filename: "banana"
  FAST_input_filename: "t3_Test05.fst"
  turb_id: 3

```

Command line invocation

```
mpiexec -np <N> openfastcpp
```

Common input file options

nTurbinesGlob

Total number of turbines in the simulation. The input file must contain a number of turbine specific sections (*Turbine0*, *Turbine1*, ..., *Turbine(n-1)*) that is consistent with *nTurbinesGlob*.

debug

Enable debug outputs if set to true

dryRun

The simulation will not run if *dryRun* is set to true. However, the simulation will read the input files, allocate turbines to processors and prepare to run the individual turbine instances. This flag is useful to test the setup of the simulation before running it.

simStart

Flag indicating whether the simulation starts from scratch or restart. *simStart* takes on one of three values:

- **init** - Use this option when starting a simulation from $t=0s$.
- **trueRestart** - While OpenFAST allows for restart of a turbine simulation, external components like the Bladed style controller may not. Use this option when all components of the simulation are known to restart.
- **restartDriverInitFAST** - When the *restartDriverInitFAST* option is selected, the individual turbine models start from $t=0s$ and run up to the specified restart time using the inflow data stored at the actuator nodes from a hdf5 file. The C++ API stores the inflow data at the actuator nodes in a hdf5 file at every OpenFAST time step and then reads it back when using this restart option. This restart option is especially useful when the glue code is a CFD solver.

tStart

Start time of the simulation

tEnd

End time of the simulation. $tEnd \leq tMax$

tMax

Max time of the simulation

dtFAST

Time step for FAST. All turbines should have the same time step.

nEveryCheckPoint

Restart files will be written every so many time steps

Turbine specific input options

turbine_base_pos

The position of the turbine base for actuator-line simulations

num_force_pts_blade

The number of actuator points along each blade for actuator-line simulations

num_force_pts_tower

The number of actuator points along the tower for actuator-line simulations.

restart_filename

The checkpoint file for this turbine when restarting a simulation

FAST_input_filename

The FAST input file for this turbine

turb_id

A unique turbine id for each turbine

4.2.14 FAST.Farm User's Guide and Theory Manual

The FAST.Farm implementation plan is also available for download: [FAST.Farm Development Plan](#).

The documentation here was derived from the FAST.Farm User's Guide and Theory Manual by Jason Jonkman and Kelsey Shaler.

Nomenclature

Table 4.16: List of Available FAST.Farm

ABLSolver	atmospheric boundary layer solver
AWAE	ambient wind and array effects (module)
$a(r)$	axial induction factor, distributed radially
a_K	coherence decrement parameter
BEM	blade-element momentum
b_K	coherence offset parameter
$C_{\text{HWkDfl}}^{\text{O}}, C_{\text{HWkDfl}}^{\text{OY}}, C_{\text{HWkDfl}}^{\text{X}}, \text{ and } C_{\text{HWkDfl}}^{\text{XY}}$	calibrated parameters in the horizontal wake-deflection correction
c_{max}	maximum blade chord length
C_{Meander}	calibrated parameter for wake meandering
C_{NearWake}	calibrated parameter in the near-wake correction
C_{WakeDiam}	calibrated parameter in the wake-diameter calculation
$C_{\nu\text{Amb}}^{\text{DMax}}, C_{\nu\text{Amb}}^{\text{DMin}}, C_{\nu\text{Amb}}^{\text{Exp}}, \text{ and } C_{\nu\text{Amb}}^{\text{FMin}}$	calibrated parameters in the eddy-viscosity filter function for ambient turbulence
$C_{\nu\text{Shr}}^{\text{DMax}}, C_{\nu\text{Shr}}^{\text{DMin}}, C_{\nu\text{Shr}}^{\text{Exp}}, \text{ and } C_{\nu\text{Shr}}^{\text{FMin}}$	calibrated parameters in the eddy-viscosity filter function for the wake shear layer
$\text{AzimAvg } C_t(r) \text{ and } \text{FiltAzimAvg } C_t(r)$	azimuthally averaged thrust-force coefficient (normal to a rotor disk), distributed radially, and its low-pass time-filtered value
$Coh_{i,j}$	magnitude of partial coherence between points i and j
DLL	dynamic-link library
DWM	dynamic wake meandering
D_{Grid}	Assumed rotor diameter when generating TurbSim inflow
D_{Rotor} and $\text{Filt } D_{n_p}^{\text{Rotor}}$	rotor diameter and its low-pass time-filtered value at wake plane n_p
$D_{n_p}^{\text{Wake}}$	wake diameter at wake plane n_p
FLORIS	FLOw Redirection and Induction in Steady state

Table 4.16 – continued from pr

f	frequency
f_c	cutoff (corner) frequency of the low-pass time filter
$\vec{f}_{n_b}(r)$	aerodynamic applied loads distributed radially per unit length for blade n_b
f_{\max}	maximum excitation frequency
$F_{\nu\text{Amb}}(x)$	eddy-viscosity filter function associated with ambient turbulence
$F_{\nu\text{Shr}}(x)$	eddy-viscosity filter function associated with the wake shear layer
HFM	high-fidelity modeling
HPC	high-performance computer
I	three-by-three identify matrix
K	velocity components u , v , and w
$k_{\nu\text{Amb}}$	calibrated parameter for the influence of ambient turbulence in the eddy viscosity
$k_{\nu\text{Shr}}$	calibrated parameter for the influence of the wake shear layer in the eddy viscosity
LES	large-eddy simulation
MFoR	moving frame of reference
MPI	message-passing interface
NaN	not a number
NREL	National Renewable Energy Laboratory
N and n	number of discrete-time steps and discrete-time-step counter
N_b and n_b	number of rotor blades and blade counter
$N_{n_p}^{\text{Polar}}$ and $n_{n_p}^{\text{Polar}}$	number of points in the polar grid of wake plane n_p and point counter
N^{Wake} and n^{Wake}	number of wakes overlapping a given wind data point in the wind domain and wake counter
N_P and n_P	number of wake planes and wake-plane counter
N_r and n_r	number of radial nodes and radii counter
N_t and n_t	number of wind turbines and turbine counter
OF	OpenFAST (module)
OpenMP	open multiprocessing
\vec{p}^{Hub}	global position of a rotor center
$\vec{p}_{n_p}^{\text{Plane}}$	global position of the center of wake plane n_p
RAM	random-access memory
RSS	root-sum-squared
r and r^{Plane}	radius in the axisymmetric coordinate system
\hat{r}^{Plane}	radial unit vector in the axisymmetric coordinate system
S	global X -, Y -, and Z -coordinate
SC	super controller (module)
SOWFA	Simulator fOr Wind Farm Applications
t	simulation time
TI_{Amb} and $TI_{\text{Amb}}^{\text{Filt}}_{n_p}$	ambient turbulence intensity of the wind at a rotor and its low-pass time-filtered value for w
u^d	discrete-time inputs
V_{Advect}	advection speed of the synthetic wind data
$\vec{V}_{\text{Amb}}^{\text{High}}$	ambient wind across a high-resolution wind domain around a turbine
$\vec{V}_{\text{Amb}}^{\text{Low}}$	ambient wind across a low-resolution wind domain throughout the wind farm
$\vec{V}_{\text{Dist}}^{\text{High}}$	disturbed wind across a high-resolution wind domain around a turbine
$\vec{V}_{\text{Dist}}^{\text{Low}}$	disturbed wind across a low-resolution wind domain throughout the wind farm
V_{Hub}	mean hub-height wind speed
$\vec{V}_{n_p}^{\text{Plane}}$ and $\vec{V}_{n_p}^{\text{Plane}}^{\text{Filt}}$	advection, deflection, and meandering velocity and its low-pass time-filtered value of wake
V_r	radial velocity in the axisymmetric coordinate system
$V_{r,n_p}^{\text{Wake}}(r)$	radial wake-velocity deficit at wake plane n_p , distributed radially
VTk	Visualization Toolkit
$\text{DiskAvg } V_x^{\text{Rel}}$ and $\text{FiltDiskAvg } V_x^{\text{Rel}}$	rotor-disk-averaged relative wind speed (ambient plus wakes of neighboring turbines plus tu

Table 4.16 – continued from pr

V_x	axial velocity in the axisymmetric coordinate system
$V_{x_{n_p}}^{\text{Wake}}(r)$	axial wake-velocity deficit at wake plane n_p , distributed radially
$\text{DiskAvg } V_x^{\text{Wind}}$ and $\text{FiltDiskAvg } V_x^{\text{Wind}}$	rotor-disk-averaged ambient wind speed, normal to the disk, and its low-pass time-filtered v
$w_{n_p}^{\text{Wind}}$	weighting in the spatial averaging for wind data point n_p^{Wind}
WD	wake dynamics (module)
WISDEM	Wind-Plant Integrated System Design & Engineering Model
x and $x_{n_p}^{\text{Plane}}$	downwind distance from a rotor to wake plane n_p in the axisymmetric coordinate system
X, Y , and Z	inertial-frame coordinates, with Z directed vertically upward, opposite gravity, X directed h
\hat{X}, \hat{Y} , and \hat{Z}	unit vectors of the inertial-frame coordinate system, parallel to the X, Y , and X coordinates
x^d	discrete-time states
$X^d(\)$	discrete-time state functions
\hat{x}^{Disk}	orientation of a rotor centerline
$\hat{x}_{n_p}^{\text{Plane}}$	orientation of wake plane n_p
y^d	discrete-time outputs
$Y^d(\)$	discrete-time output functions
z_{bot}	bottom vertical location of synthetic turbulence inflow grid
α	low-pass time-filter parameter
Δt	discrete time step (increment)
γ^{YawErr} and $\text{Filt } \gamma_{n_p}^{\text{YawErr}}$	nacelle-yaw error of a rotor and its low-pass time-filtered value at wake plane n_p
ν_T	eddy viscosity
ρ	air density
2D	two dimensional
3D	three dimensional

Introduction

FAST.Farm is a midfidelity multiphysics engineering tool for predicting the power performance and structural loads of wind turbines within a wind farm. FAST.Farm uses [OpenFAST](#) to solve the aero-hydro-servo-elastic dynamics of each individual turbine, but considers additional physics for wind farm-wide ambient wind in the atmospheric boundary layer; a wind-farm super controller; and wake deficits, advection, deflection, meandering, and merging. FAST.Farm is based on some of the principles of the dynamic wake meandering (DWM) model – including passive tracer modeling of wake meandering – but addresses many of the limitations of previous DWM implementations. FAST.Farm maintains low computational cost to support the often highly iterative and probabilistic design process. Applications of FAST.Farm include reducing wind farm underperformance and loads uncertainty, developing wind farm controls to enhance operation, optimizing wind farm siting and topology, and innovating the design of wind turbines for the wind-farm environment. The existing implementation of FAST.Farm also forms a solid foundation for further development of wind farm dynamics modeling as wind farm physics knowledge grows from future computations and experiments.

The main idea behind the DWM model is to capture key wake features pertinent to accurate prediction of wind farm power performance and wind turbine loads, including the wake-deficit evolution (important for performance) and the wake meandering and wake-added turbulence (important for loads). The wake-deficit evolution and wake meandering are illustrated in [Fig. 4.45](#).

Although fundamental laws of physics are applied, appropriate simplifications have been made to minimize the computational expense, and high-fidelity modeling (HFM) solutions, e.g., using the Simulator fOr Wind Farm Applications ([SOWFA](#)), have been used to inform and calibrate the submodels. In the DWM model, the wake-flow processes are treated via the “splitting of scales,” in which small turbulent eddies (less than two diameters) affect wake-deficit evolution and large turbulent eddies (greater than two diameters) affect wake meandering.

FAST.Farm is a nonlinear time-domain multiphysics engineering tool composed of multiple submodels, each representing different physics domains of the wind farm. FAST.Farm is implemented as open-source software that follows the programming requirements of the FAST modularization framework, whereby the submodels are implemented as

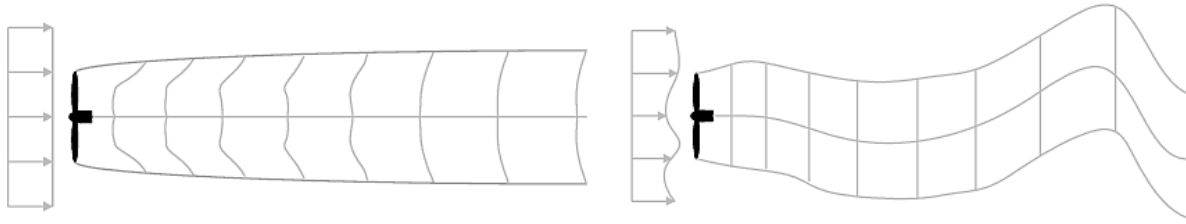


Fig. 4.45: Axisymmetric wake deficit (left) and meandering (right) evolution.

modules interconnected through a driver code. The submodel hierarchy of FAST.Farm is illustrated in Fig. 4.46.

Wake advection, deflection, and meandering; near-wake correction; and wake-deficit increment are submodels of the wake-dynamics (*WD*) model, implemented in a single module. Ambient wind and wake merging are submodels of the ambient wind and array effects (*AWAE*) model, implemented in a single module. Combined with the super controller (*SC*) and OpenFAST (*OF*) modules, FAST.Farm has four modules and one driver. There are multiple instances of the *OF* and *WD* modules – one instance for each wind turbine/rotor.

FAST.Farm Driver

The FAST.Farm driver, also known as the “glue code,” is the code that couples individual modules together and drives the overall time-domain solution forward. Additionally, the FAST.Farm driver reads an input file of simulation parameters, checks the validity of these parameters, initializes the modules, writes results to a file, and releases memory at the end of the simulation.

Super Controller Module

The *SC* module of FAST.Farm – essentially identical to the super controller available in [SOWFA](#) allows wind-farm-wide control logic to be implemented by the user, including sending and receiving commands from the individual turbine controllers in OpenFAST. The logic of such a super controller could be developed through the application of the National Renewable Energy Laboratory (NREL) code FLOW Redirection and Induction in Steady state ([FLORIS](#)).

OpenFAST Module

The *OF* module of FAST.Farm is a wrapper that enables the coupling of [OpenFAST](#) to FAST.Farm. OpenFAST models the dynamics (loads and motions) of distinct turbines in the wind farm, capturing the environmental excitations (wind inflow and, for offshore systems, waves, current, and ice) and coupled system response of the full system (the rotor, drivetrain, nacelle, tower, controller, and, for offshore systems, the substructure and station-keeping system). OpenFAST itself is an interconnection of various modules, each corresponding to different physical domains of the coupled aero-hydro-servo-elastic solution. There is one instance of the *OF* module for each wind turbine, which, in parallel mode, are parallelized through open multiprocessing (OpenMP). At initialization, the number of wind turbines, associated OpenFAST primary input file(s), and turbine origin(s) in the global *X-Y-Z* inertial-frame coordinate system are specified by the user of FAST.Farm. Turbine origins are defined as the intersection of the undeflected tower centerline and the ground or, for offshore systems, the mean sea level. The global inertial-frame coordinate system is defined with *Z* directed vertically upward (opposite gravity), *X* directed horizontally nominally downwind (along the zero-degree wind direction), and *Y* directed horizontally transversely. This coordinate system is not tied to specific compass directions. Among other time-dependent inputs from FAST.Farm, OpenFAST uses the disturbed wind (ambient plus wakes) across a high-resolution wind domain (in both time and space) around the turbine as input. This high-resolution domain

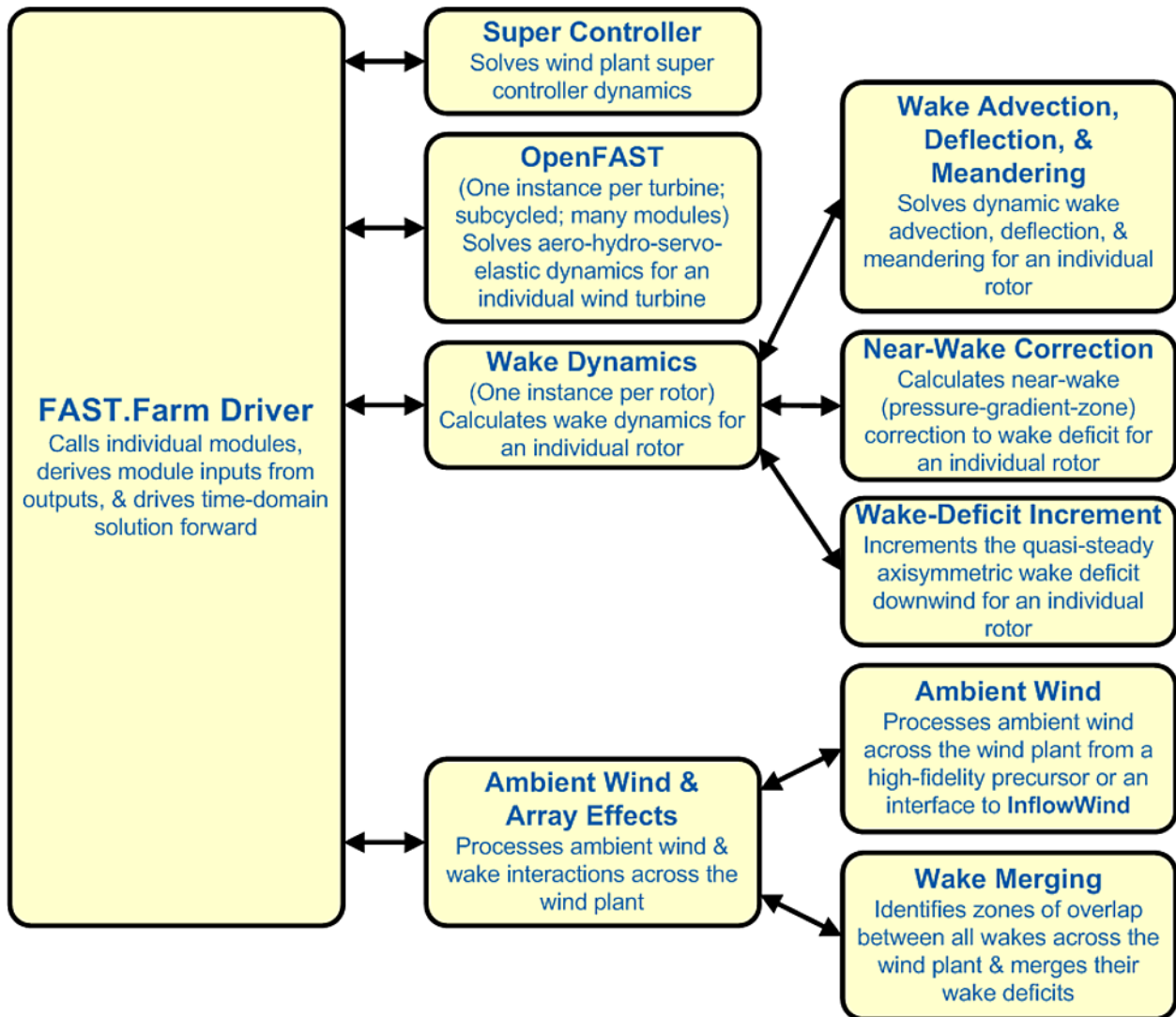


Fig. 4.46: FAST.Farm submodel hierarchy.

ensures that the individual turbine loads and responses calculated by OpenFAST are accurately driven by flow through the wind farm, including wake and array effects.

Wake Dynamics Module

The *WD* module of FAST.Farm calculates wake dynamics for an individual rotor, including wake advection, deflection, and meandering; a near-wake correction; and a wake-deficit increment. The near-wake correction treats the near-wake (pressure-gradient zone) correction of the wake deficit. The wake-deficit increment shifts the quasi-steady-state axisymmetric wake deficit nominally downwind. There is one instance of the *WD* module for each rotor. The wake-dynamics calculations involve many user-specified parameters that may depend, e.g., on turbine operation or atmospheric conditions and can be calibrated to better match experimental data or by using an HFM solution as a benchmark. Default values have been derived for each calibrated parameter based on *SOWFA* simulations, but these can be overwritten by the user.

The wake-deficit evolution is solved in discrete time on an axisymmetric finite-difference grid consisting of a fixed number of wake planes, each with a fixed radial grid of nodes. The radial finite-difference grid can be considered a plane because the wake deficit is assumed to be axisymmetric. A wake plane can be thought of as a cross section of the wake wherein the wake deficit is calculated.

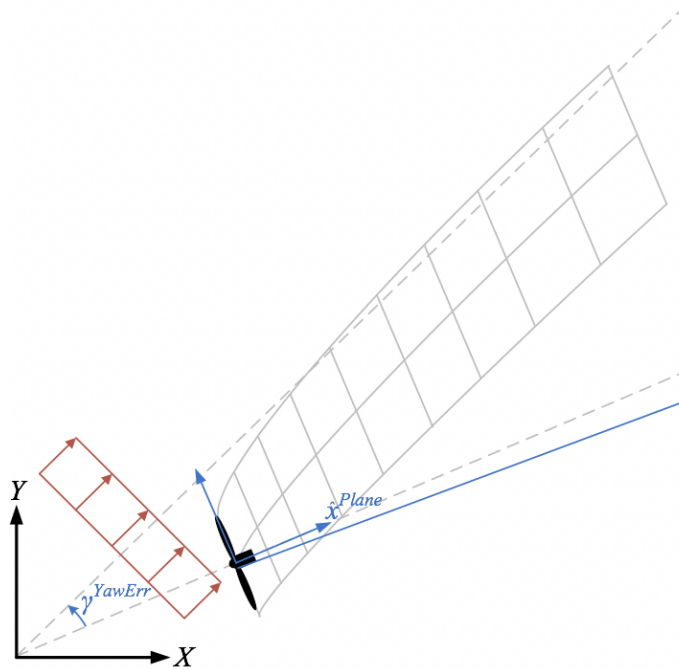


Fig. 4.47: Wake deflection resulting from inflow skew, including a horizontal wake-deflection correction. The lower dashed line represents the rotor centerline, the upper dashed line represents the wind direction, and the solid blue line represents the horizontal wake-deflection correction (offset from the rotor centerline).

By simple extensions to the passive tracer solution for transverse (horizontal and vertical) wake meandering, the wake-dynamics solution in FAST.Farm is extended to account for wake deflection, as illustrated in Fig. 4.47, and wake advection, as illustrated in Fig. 4.48, among other physical improvements such as:

1. Calculating the wake-plane velocities by spatially averaging the disturbed wind instead of the ambient wind (in the AWAE module)
2. Orientating the wake planes with the rotor centerline instead of the wind direction

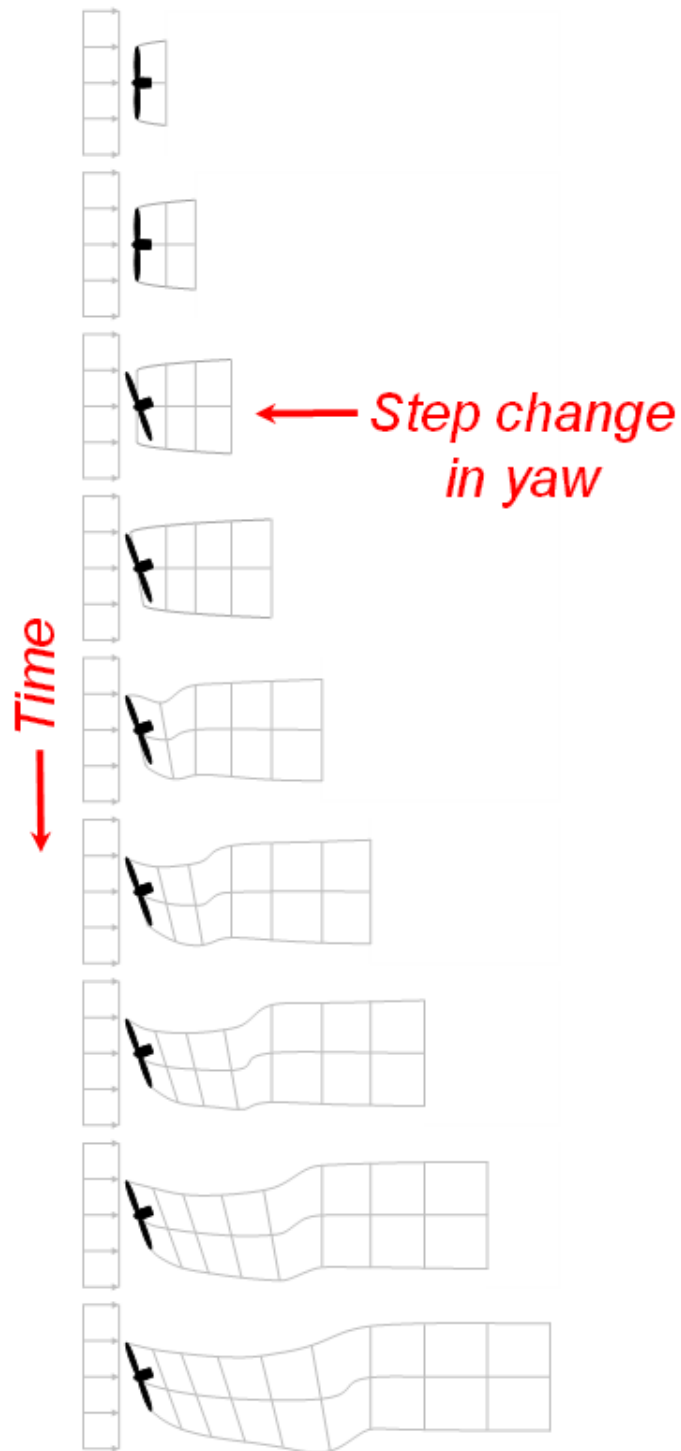


Fig. 4.48: Wake advection for a single turbine resulting from a step change in yaw angle.

3. Low-pass time filtering the local conditions at the rotor, as input to the wake dynamics module, to account for transients in inflow, turbine control, and/or turbine motion instead of considering time-averaged conditions.

With these extensions, the passive tracer solution enables:

1. The wake centerline to deflect based on inflow skew, because in skewed inflow, the wake deficit normal to the disk introduces a velocity component that is not parallel to the ambient flow
2. The wake to accelerate from near wake to far wake, because the wake deficits are stronger in the near wake and weaken downwind
3. The wake-deficit evolution to change based on conditions at the rotor, because low-pass time filtering conditions are used instead of time-averaging
4. The wake to meander axially in addition to transversely, because local axial winds are considered
5. The wake shape to be elliptical instead of circular in skewed flow when looking downwind (the wake shape remains circular when looking down the rotor centerline).

From item 1 above, a horizontally asymmetric correction to the wake deflection is accounted for, i.e., a correction to the wake deflection resulting from the wake-plane velocity, which physically results from the combination of wake rotation and shear not modeled directly in the *WD* module (see [Fig. 4.47](#) for an illustration). This horizontal wake deflection correction is a simple linear correction (with a slope and offset), similar to the correction implemented in the wake model of [FLORIS](#). Such a correction is important for accurate modeling of nacelle-yaw-based wake-redirection (wake-steering) wind farm control.

From item 3, low-pass time filtering is important because the wake reacts slowly to changes in local conditions at the rotor and because the wake evolution is treated in a quasi-steady-state fashion.

The near-wake correction submodel of the *WD* module computes the wake-velocity deficits at the rotor disk, as an inlet boundary condition for the wake-deficit evolution. To improve the accuracy of the far-wake solution, the near-wake correction accounts for the drop-in wind speed and radial expansion of the wake in the pressure-gradient zone behind the rotor that is not otherwise accounted for in the solution for the wake-deficit evolution.

As with most DWM implementations, the *WD* module of FAST.Farm models the wake-deficit evolution via the thin shear-layer approximation of the Reynolds-averaged Navier-Stokes equations under quasi-steady-state conditions in axisymmetric coordinates, with turbulence closure captured by using an eddy-viscosity formulation. The thin shear-layer approximation drops the pressure term and assumes that the velocity gradients are much bigger in the radial direction than in the axial direction.

Ambient Wind and Array Effects Module

The *AWAE* module of FAST.Farm processes ambient wind and wake interactions across the wind farm, including the ambient wind submodel, which processes ambient wind across the wind farm and the wake-merging submodel, which identifies zones of overlap between all wakes across the wind farm and merges their wake deficits. The calculations in the *AWAE* module make use of wake volumes, which are volumes formed by a (possibly curved) cylinder starting at a wake plane and extending to the next adjacent wake plane along a line connecting the centers of the two wake planes. If the adjacent wake planes (top and bottom of the cylinder) are not parallel, e.g., for transient simulations involving variations in nacelle-yaw angle, the centerline will be curved. [Fig. 4.49](#) illustrates some of the concepts.

The calculations in the *AWAE* module also require looping through all wind data points, turbines, and wake planes; these loops have been sped up in the parallel mode of FAST.Farm by implementation of open multiprocessing (OpenMP) parallelization.

Ambient wind may come from either a high-fidelity precursor simulation or an interface to the *InflowWind* module in OpenFAST. The use of the *InflowWind* module enables the use of simple ambient wind, e.g., uniform wind, discrete wind events, or synthetically generated turbulent wind data. Synthetically generated turbulence can be generated from, e.g., TurbSim or the Mann model, in which the wind is propagated through the wind farm using Taylor's frozen-turbulence assumption. This method is most applicable to small wind farms or a subset of wind turbines within a larger

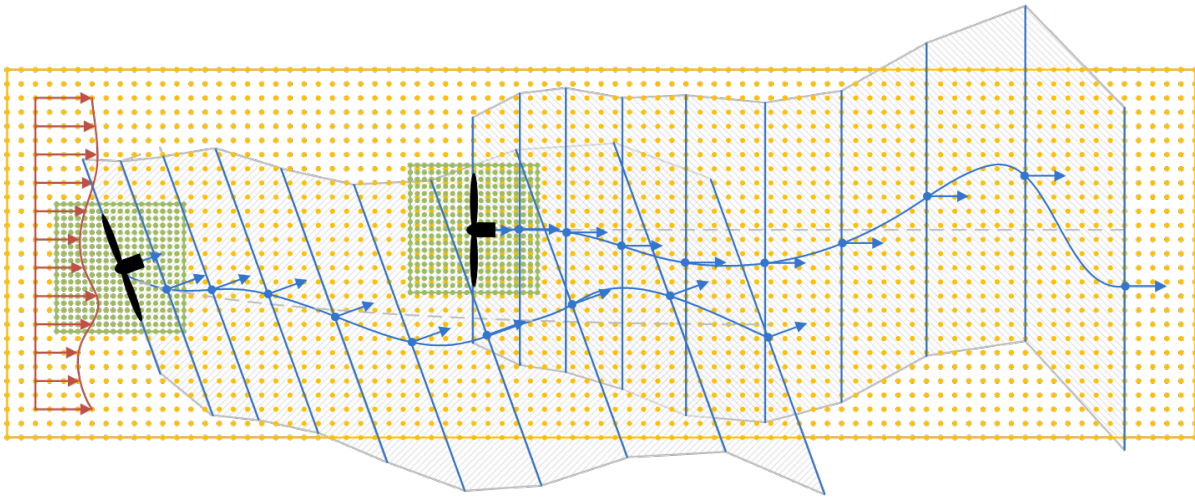


Fig. 4.49: Wake planes, wake volumes, and zones of wake overlap for a two-turbine wind farm, with the upwind turbine yawed. The yellow points represent the low-resolution wind domain and the green points represent the high-resolution wind domains around each turbine. The blue points and arrows represent the centers and orientations of the wake planes, respectively, with the wake planes identified by the blue lines normal to their orientations. The gray dashed lines represent the mean trajectory of the wake and the blue curves represent the instantaneous [meandered] trajectories. The wake volumes associated with the upwind turbine are represented by the upward hatch patterns, the wake volumes associated with the downwind turbine are represented by the downward hatch patterns, and the zones of wake overlap are represented by the crosshatch patterns. (For clarity of the illustration, the instantaneous (meandered) wake trajectory is shown as a smooth curve, but will be modeled as piece-wise linear between wake planes when adjacent wake planes are parallel. The wake planes and volumes are illustrated with a diameter equal to twice the wake diameter, but the local diameter depends on the calculation. As illustrated, a wake plane or volume may extend beyond the boundaries of the low-resolution domain of ambient wind data.)

wind farm. FAST.Farm can also use ambient wind generated by a high-fidelity precursor large-eddy simulation (LES) of the entire wind farm (without wind turbines present), such as the atmospheric boundary layer solver (ABLSolver) preprocessor of SOWFA. This atmospheric precursor simulation captures more physics than synthetic turbulence – as illustrated in Fig. 4.50 – including atmospheric stability, wind-farm-wide turbulent length scales, and complex terrain effects.

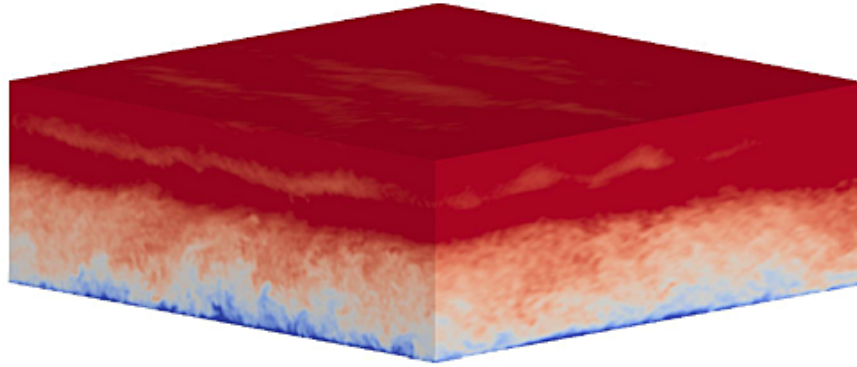


Fig. 4.50: Example flow generated by ABL Solver.

This method is more computationally expensive than using the ambient wind modeling options of InflowWind, but it is much less computationally expensive than a SOWFA simulation with wind turbines present. FAST.Farm requires ambient wind to be available in two different resolutions in both space and time. Because wind will be spatially averaged across wake planes within the *AWAE* module, FAST.Farm needs a low-resolution wind domain throughout the wind farm wherever turbines may potentially reside. For accurate load calculation by OpenFAST, FAST.Farm also needs high-resolution wind domains around each wind turbine (encompassing any turbine displacement). The high-resolution domains will occupy the same space as portions of the low-resolution domain, requiring domain overlap.

When using ambient wind generated by a high-fidelity precursor simulation, the *AWAE* module reads in the three-component wind-velocity data across the high- and low-resolution domains that were computed by the high-fidelity solver within each time step. These values are stored in files for use in a given driver time step. The wind data files, including spatial discretizations, must be in Visualization Toolkit (VTK) format and are specified by users of FAST.Farm at initialization. Visualization Toolkit is an open-source, freely available software system for three-dimensional (3D) computer graphics, image processing, and visualization. When using the *InflowWind* inflow option, the ambient wind across the high- and low-resolution domains are computed by calling the *InflowWind* module. In this case, the spatial discretizations are specified directly within the FAST.Farm primary input file. These wind data from the combined low- and high-resolution domains within a given driver time step represent the largest memory requirement of FAST.Farm.

In previous implementations of DWM, the wind turbine and wake dynamics were solved individually or serially, not considering two-way wake-merging interactions. Additionally, there was no method available to calculate the disturbed wind in zones of wake overlap. Wake merging is illustrated by the FAST.Farm simulation of Fig. 4.51.

In FAST.Farm, the wake-merging submodel of the *AWAE* module identifies zones of wake overlap between all wakes across the wind farm by finding wake volumes that overlap in space. Wake deficits are superimposed in the axial direction based on the root-sum-squared (RSS) method. Transverse components (radial wake deficits) are superimposed by vector sum. The RSS method assumes that the local kinetic energy of the axial deficit in a merged wake equals the sum of the local energies of the axial deficits for each wake at the given wind data point. The RSS method only applies to an array of scalars. This method works well for axial deficits because overlapping wakes likely have similar axial directions; therefore, only the magnitude of the vector is important in the superposition. A vector sum is applied to the transverse components (radial wake deficits) because any given radial direction is dependent on the azimuth angle in the axisymmetric coordinate system.

To visualize the ambient wind and wake interactions across the wind farm, FAST.Farm includes visualization capability through the generation of output files in VTK format. OpenFAST can further generate VTK-formatted output files for visualizing the wind turbine based on either surface or stick-figure geometry. The VTK files generated by FAST.Farm and OpenFAST can be read with standard open-source visualization packages such as ParaView or VisIt.

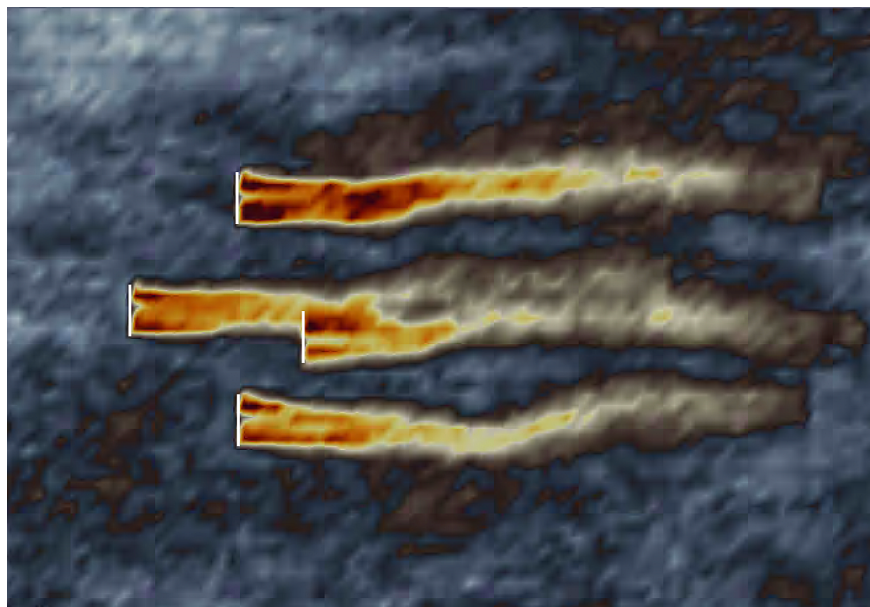


Fig. 4.51: Wake merging for closely spaced rotors.

FAST.Farm Parallelization

FAST.Farm can be compiled and run in serial or parallel mode. Parallelization has been implemented in FAST.Farm through OpenMP, which allows FAST.Farm to take advantage of multicore computers by dividing computational tasks among the cores/threads within a node (but not between nodes) to speed up a single simulation. The size of the wind farm and number of wind turbines is limited only by the available random-access memory (RAM). In parallel mode, each instance of the OpenFAST submodel can be run in parallel on separate threads at the same time the ambient wind within the *AWAE* module is being read in another thread. Thus, the fastest simulations require at least one more core than the number of wind turbines in the wind farm. Furthermore, the output calculations within the *AWAE* module are parallelized into separate threads. Because of the small timescales involved and sophisticated physics, the *OF* submodel is the computationally slowest FAST.Farm module. The output calculation of the *AWAE* module is the only major calculation that cannot be solved in parallel to OpenFAST; therefore, at best, the parallelized FAST.Farm solution may execute only slightly more slowly than stand-alone OpenFAST simulations – computationally inexpensive enough to run the many simulations necessary for wind turbine/farm design and analysis.

To support the modeling of large wind farms, single simulations involving memory parallelization and parallelization between nodes of a multinode high-performance computer (HPC) through a message-passing interface (MPI) is likely required. MPI has not yet been implemented within FAST.Farm.

Organization of the Guide

The remainder of this documentation is structured as follows: [Section 4.2.14](#) details how to obtain the FAST.Farm software archive and how to run FAST.Farm. [Section 4.2.14](#) describes the FAST.Farm input files. [Section 4.2.14](#) discusses the output files generated by FAST.Farm. [Section 4.2.14](#) provides modeling guidance when using FAST.Farm. The FAST.Farm theory is covered in [Section 4.2.14](#). [Section 4.2.14](#) outlines future work, and the bibliography provides background and other information sources. Example FAST.Farm primary input and ambient wind data files are shown in [Section 4.2.14](#) and [Section 4.2.14](#). A summary of available output channels is found in [Section 4.2.14](#).

Running FAST.Farm

As FAST.Farm is a module of OpenFAST, the process of downloading, compiling, and running FAST.Farm is the same as that for OpenFAST. Such instructions are available in the *Installing OpenFAST* documentation.

Note: To improve the speed of OpenFAST compiled with FAST.Farm enabled, the user may wish to compile in single precision with *OpenMP*. To do so, add the `-DDOUBLE_PRECISION:BOOL=OFF -DOPENMP=ON` options with CMake.

Note: Checkpoint-restart capability has not yet been implemented within FAST.Farm.

Input Files

The primary FAST.Farm input file defines ambient wind, the wind turbine layout within the wind farm, the wake axisymmetric finite-difference grid, calibrated parameters for wake dynamics, visualization output, output file specifications, and auxiliary parameters. Ambient wind data optionally generated from the high-fidelity precursor atmospheric simulation are stored in separate files referenced in the primary FAST.Farm input file. Properties for each wind turbine in the wind farm are stored in the standard OpenFAST input files, referenced by their primary OpenFAST input file (one for each wind turbine) in the primary FAST.Farm input file.

No lines should be added or removed from the input files, except in tables where the number of rows is specified.

Units

FAST.Farm uses the SI system (kg, m, s, N).

FAST.Farm Primary Input File

The FAST.Farm primary input file is organized into several functional sections:

- Simulation Control
- Super Controller
- Ambient Wind
- Wind Turbines
- Wake Dynamics
- Visualization
- Output.

Each section corresponds to an aspect of the FAST.Farm model – see the subsections below. A sample FAST.Farm primary input file is given in [Section 4.2.14](#). Where there is a one-to-one equivalency between an input parameter and a variable in the FAST.Farm theory documented in [Section 4.2.14](#), the variable in [Section 4.2.14](#) is shown in parentheses after the input parameter in the subsections below.

The input file begins with two lines of header information that is for your use, but is not used by the software.

Simulation Control

Echo [flag] specifies if you wish to have FAST.Farm echo the contents of the FAST.Farm primary input file (useful for debugging errors in the input file). If **Echo** = TRUE, an echo file will be generated. The echo file has the naming convention of *<RootName>.ech*, where *<RootName>* is the name of the FAST.Farm primary input file, excluding its file extension.

AbortLevel [quoted string] indicating what error level should cause an abort. Options are: “WARNING,” “SEVERE,” or “FATAL.” **AbortLevel** in FAST.Farm is used the same way as the level set in stand-alone OpenFAST, but the **AbortLevel** set in FAST.Farm will override the levels set in the OpenFAST primary input file of each wind turbine in the wind farm. Setting FAST.Farm to abort on fatal errors is typical, but see the FAST v8 ReadMe document for additional guidance.

TMax [sec] is the total length of the simulation to be run. The first output is calculated at $t = 0$; the last output is calculated at $t = \mathbf{TMax}$. The **TMax** set in FAST.Farm will override the simulation length set in the OpenFAST primary input file of each wind turbine in the wind farm.

UseSC [flag] indicates if the wind-farm-wide super controller is to be used. If **UseSC** = TRUE, the super controller will be called. If **UseSC** = FALSE, the super controller will not be called, but each wind turbine may still have an individual controller specified in the OpenFAST module *ServoDyn*.

Mod_AmbWind [switch] indicates the ambient wind source. There are three options: 1) use ambient wind data generated by a high-fidelity precursor simulation in VTK format [**Mod_AmbWind=1**], 2) use ambient wind data as defined by the FAST.Farm interface to the *InflowWind* module, with one instance of *InflowWind* [**Mod_AmbWind=2**], or 3) use ambient wind data as defined by the FAST.Farm interface to the *InflowWind* module, with multiple instances of *InflowWind* [**Mod_AmbWind=3**]. The distinct Ambient Wind subsections below pertain to each option.

Super Controller

SC_FileName [quoted string] sets the name and location of the dynamic library containing the super controller code. It is only used when **UseSC** = TRUE. The dynamic library should be compiled as a *.dll* file in Windows or a *.so* file in Linux or Mac OS. **The file name must be in quotations** and can contain an absolute or a relative path. The super controller is used in conjunction with individual wind turbine controllers defined in the style of the DISCON dynamic library of the DNV GL’s Bladed wind turbine software package, with minor modification. See [Section 4.2.14](#) for more information.

Ambient Wind: Precursor in Visualization Toolkit Format

The input parameters in this section are only used when **Mod_AmbWind** = 1, indicating the use of ambient wind generated by a high-fidelity precursor simulation. In this case, the ambient wind, including their spatial discretization, must be stored in VTK format – as described in [Section 4.2.14](#) – and is used directly without modification by FAST.Farm.

DT_Low-VTK [sec] (t) sets the time step of the low-resolution ambient wind data files and calculation, as well as the global (driver/glue-code) time step of FAST.Farm. **DT_Low-VTK** is the same as **DT_Low** in this documentation. The modules of FAST.Farm are called every **DT_Low** seconds, although OpenFAST and its modules may use a time step that is an integer multiple smaller than or equal to **DT_Low**.

DT_High-VTK [sec] sets the time step of the high-resolution ambient wind data files and calculation and **must be an integer multiple smaller than or equal to DT_Low**. **DT_High-VTK** is the same as **DT_High** in this documentation. It is essential that **DT_Low** and **DT_High** are small enough to ensure solution accuracy and match the time resolution used when generating the ambient wind data from the high-fidelity precursor simulation. **DT_Low** should be consistent with the timescales of wake dynamics, e.g., on the order of seconds and smaller for higher mean wind speeds. **DT_High** should be sufficient for accurate aerodynamic load calculations, e.g., on the order of fractions of a second. Further guidance on choosing appropriate time steps is given in [Section 4.2.14](#).

WindFilePath [quoted string] specifies the path to the directory where the low- and high-resolution ambient wind data files are stored. The path can be specified relative to the location of the FAST.Farm primary input file or with an absolute path. It is recommended to use quotes around the path. If there are spaces in the file or path names, these quotes are required. **FAST.Farm requires that the ambient wind data files be stored in specific subdirectories of the directory specified by WindFilePath and with specific filenames.** The low-resolution ambient wind data files must be named *Amb.t<n_{low}>.vtk* and stored in a subdirectory named *Low*. In the file names, *<n_{low}>* is an integer (without leading zeros) between 0 (at $t = 0$) and $N-1$, where $N = \text{FLOOR}\left(\frac{T_{Max}}{DT_{Low}}\right) + 1$ is the number of low-resolution time steps. The high-resolution ambient wind data files must be named *Amb.t<n_{high}>.vtk*, where *<n_{high}>* is an integer (without leading zeros) between 0 (at $t = 0$) and $\frac{DT_{Low}}{DT_{High}}(N - 1)$. The files must be stored in a subdirectory named *HighT<n_i>*, where *<n_i>* is an integer (without leading zeros) between 1 and the total number of wind turbines (**NumTurbines**). Subdirectory *HighT<n_i>* must contain the high-resolution ambient wind data corresponding to wind turbine *<n_i>*, specified in the Wind Turbines section of the FAST.Farm primary input file – see [Section 4.2.14](#). The VTK format of each ambient wind data file – for both the low-resolution and high-resolution domains – is identical, as described in [Section 4.2.14](#).

ChkWndFiles [flag] specifies if FAST.Farm should check the ambient wind data files for consistency before running the simulation (preventing a possible crash later). As this check is time intensive, it is recommended that **ChkWndFiles** be set to FALSE (to disable the check) if the ambient wind data have previously been checked, such as in a prior simulation. If set to TRUE, FAST.Farm will check to ensure that:

- The number of low-resolution ambient wind data files is sufficient to run the entire simulation (up to $t = T_{Max}$). If more files are in the subdirectory, only the first N will be used.
- The number of high-resolution ambient wind data files is sufficient to run the entire simulation (up to $t = T_{Max}$) for all wind turbines. If there are more subdirectories, only the first **NumTurbines** will be used. If more files are in each subdirectory, only the first $\frac{DT_{Low}}{DT_{High}}(N - 1) + 1$ will be used.
- The spatial resolution (number of grid points, origin, and spacing) of each low-resolution ambient wind data file is the same.
- The spatial resolution (number of grid points, origin, and spacing) of each high-resolution ambient wind data file is the same for a given wind turbine.
- The number of grid points in each high-resolution domain is the same for all wind turbines in the wind farm.

Ambient Wind: InflowWind Module

The input parameters in this section are only used when **Mod_AmbWind** = 2 or 3, indicating the use of ambient wind through one or multiple instances of the *InflowWind* module. In this case, the ambient wind specified within *InflowWind* is interpolated to the low- and high-resolution domains for use within FAST.Farm.

DT_Low [sec] (Δt) sets the time step of the low-resolution ambient wind calculation, as well as the global (driver/glue-code) time step of FAST.Farm. The modules of FAST.Farm are called every **DT_Low** seconds, although OpenFAST and its modules may choose to use a time step that is an integer multiple smaller than or equal to **DT_Low**.

DT_High [sec] sets the time step of the high-resolution ambient wind data calculation and must be an integer multiple smaller than or equal to **DT_Low**. It is essential that **DT_Low** and **DT_High** are small enough to ensure solution accuracy. **DT_Low** should be consistent with the timescales of wake dynamics, e.g., on the order of seconds and smaller for higher mean wind speeds. **DT_High** should be sufficient for accurate aerodynamic load calculations, e.g., on the order of fractions of a second. Further guidance on choosing appropriate time steps is given in [Section 4.2.14](#).

The next nine input parameters set the spatial discretization of the low-resolution ambient wind domain. The low-resolution domain is stored as a structured 3D grid of wind data points (representing the corners of 3D cells) in the global X-Y-Z inertial-frame coordinate system, as illustrated generically in [Fig. 4.52](#).

NX_Low, **NY_Low**, and **NZ_Low** [integer] set the number of wind data points in each direction.

X0_Low, **Y0_Low**, and **Z0_Low** [m] set the origin of the grid (lowest-most X-Y-Z coordinate).

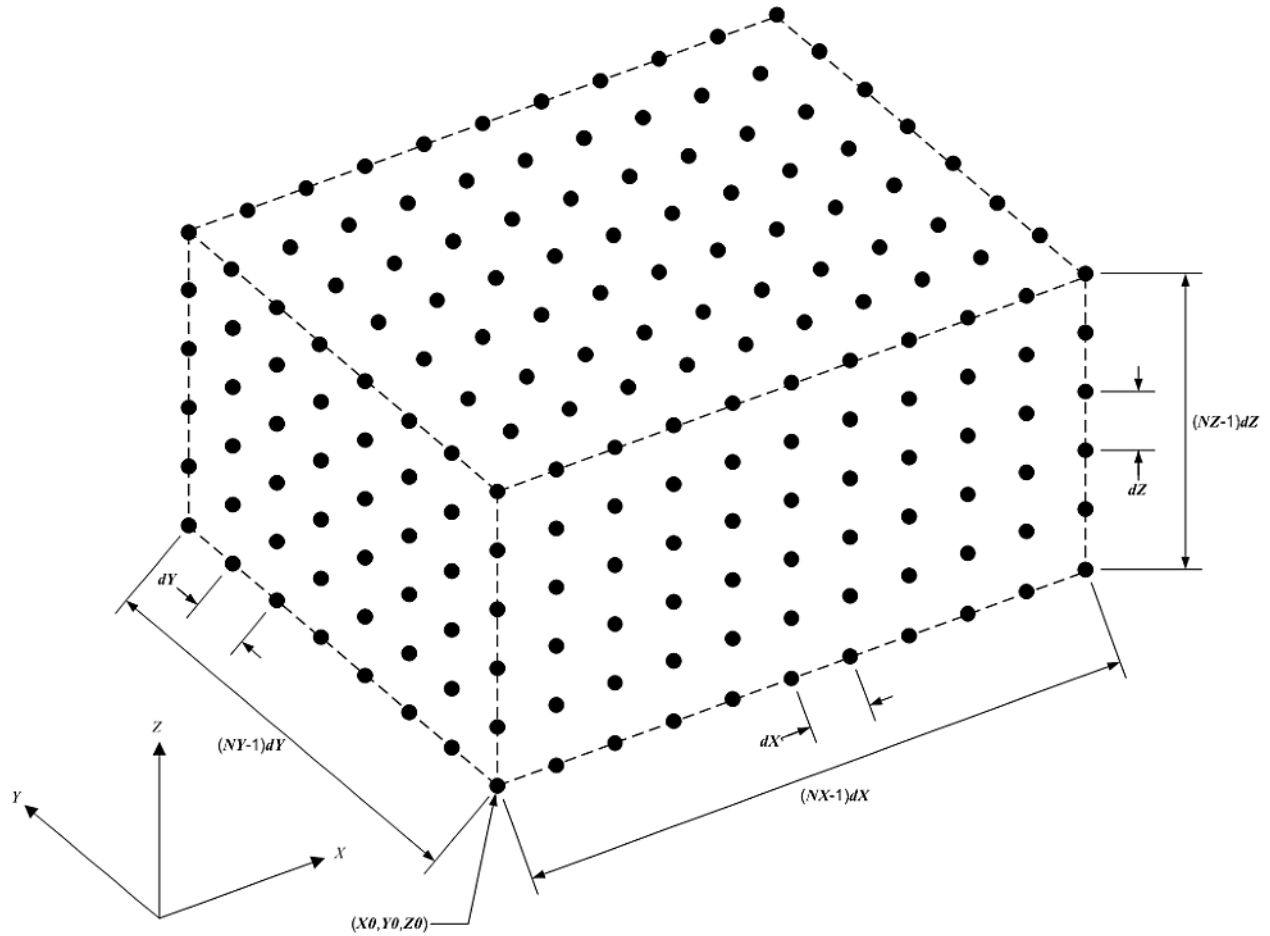


Fig. 4.52: Structured 3D grid for the low- or high-resolution domains.

dX_Low, **dY_Low**, and **dZ_Low** [m] set the spatial discretization in each direction.

The total low-resolution domain size has dimensions $(\mathbf{NX_Low}-1)\mathbf{dX_Low} \times (\mathbf{NY_Low}-1)\mathbf{dY_Low} \times (\mathbf{NZ_Low}-1)\mathbf{dZ_Low}$. The low-resolution domain should extend throughout the wind farm wherever turbines and wakes may potentially reside with a resolution sufficient so that the spatial averaging is accurate, e.g., on the order of tens of meters for utility-scale wind turbines. Further guidance on choosing appropriate spatial discretization is given in [Section 4.2.14](#).

Like the low-resolution domain, each high-resolution domain is stored as a structured 3D grid of wind data points in the global *X-Y-Z* inertial-frame coordinate system – as illustrated generically in [Fig. 4.52](#).

NX_High, **NY_High**, and **NZ_High** [integer] set the number of wind data points in each direction. These values are the same for each wind turbine and so only need to be set once.

The origin and spatial discretization for the high-resolution wind domain for each turbine are specified in the Wind Turbines section of the FAST.Farm primary input file below.

InflowFile [quoted string] specifies the name of the primary input file for the *InflowWind* module, which can be specified relative to the location of the FAST.Farm primary input file or specified with an absolute path. It is recommended to use quotes around the file name. If there are spaces in the file or path names, these quotes are required. See [Section 4.2.14](#) for information on the contents of this file.

Wind Turbines

NumTurbines [integer] (N_t) is the number of wind turbines in the wind farm and determines the number of rows in the subsequent table (after two table header lines).

For each wind turbine:

- **WT_X**, **WT_Y**, and **WT_Z** [m] specify the origin in the global *X-Y-Z* inertial-frame coordinate system. The origin is defined as the intersection of the undeflected tower centerline and the ground or, for offshore systems, mean sea level.
- **WT_FASTInFile** [quoted string] specifies the name of the OpenFAST primary input file associated with each turbine. Each wind turbine is numbered within FAST.Farm as an integer (n_t) between 1 and **NumTurbines** corresponding to the row in the table. The OpenFAST primary input file name can be specified relative to the location of the FAST.Farm primary input file or with an absolute path. It is recommended to use quotes around the file name. Identical wind turbines can use the same OpenFAST primary input file, except if the corresponding OpenFAST model makes use of a Bladed-style controller in DLL format or, for offshore wind turbines, if different wave conditions are required for each turbine. If a Bladed-style DLL controller is being used, distinct Bladed-style controller DLLs must be used (each with a unique name). This requires the need for distinct *ServoDyn* primary input files, referencing the appropriate DLL name, and distinct OpenFAST primary input files, each referencing the appropriate *ServoDyn* primary input file name. If different wave conditions are required for each turbine, the distinct wave conditions (e.g., based on unique random wave seeds) for each wind turbine must be set in the *HydroDyn* primary input file and distinct OpenFAST primary input files must be used, each referencing the appropriate *HydroDyn* primary input file name. See [Section 4.2.14](#) for information on the contents of the OpenFAST input files.
- When **Mod_AmbWind** = 2 or 3, the Wind Turbines table has six additional columns to complete the spatial discretization of the high-resolution wind domain for each wind turbine:
 - **X0_High**, **Y0_High**, and **Z0_High** [m] set the origin of the grid.
 - **dX_High**, **dY_High**, **dZ_High** [m] set spatial discretization in each direction.

The total high-resolution domain size has dimensions $(\mathbf{NX_High}-1)\mathbf{dX_High} \times (\mathbf{NY_High}-1)\mathbf{dY_High} \times (\mathbf{NZ_High}-1)\mathbf{dZ_High}$. Each high-resolution domain must extend around the corresponding wind turbine, encompassing any turbine displacement. The domains should have a resolution sufficient for accurate aerodynamic load calculations,

e.g., on the order of the blade chord length. The high-resolution domains will occupy the same space as portions of the low-resolution domain, requiring domains overlap.

Wake Dynamics

With FAST.Farm, each wake plane is treated as a radial finite-difference grid, as shown in Fig. 4.53.

These planes are defined by the following parameters:

- **dr** [m] sets the radial increment. To ensure the wake deficits are accurately computed by FAST.Farm, **dr** should be set so that FAST.Farm sufficiently resolves the wake deficit within each plane.
- **NumRadii** [integer] (N_r) sets the number of radii. To ensure the wake deficits are accurately computed by FAST.Farm, **NumRadii** should be set so that the diameter of each wake plane, $2(\text{NumRadii}-1)\text{dr}$, is large relative to the rotor diameter.
- **NumPlanes** [integer] (N_p) sets the number of wake planes. To ensure the wake deficits are accurately captured by FAST.Farm, **NumPlanes** should be set so that the wake planes propagate a sufficient distance downstream, preferably until the wake deficit decays away.

The next 20 inputs are user-specified calibration parameters and options that influence the wake-dynamics calculations. The parameters may depend, e.g., on turbine operation or atmospheric conditions that can be calibrated to better match experimental data or by using an HFM benchmark. Default values have been derived for each calibrated parameter based on SOWFA simulations ([ff-Deal18]), but these can be overwritten by the user.

f_c [Hz] (f_c) is the cutoff (corner) frequency of the low-pass time filter for the wake advection, deflection, and mean-dering model and must be greater than zero. If the DEFAULT keyword is specified in place of a numerical value, **f_c** is set to 0.0007.

C_HWkDfl_O [m] (C_{HWkDfl}^O) is the calibrated parameter for the wake deflection correction defining the horizontal offset at the rotor. If the DEFAULT keyword is specified in place of a numerical value, **C_HWkDfl_O** is set to 0.0.

C_HWkDfl_OY [m/deg] (C_{HWkDfl}^{OY}) is the calibrated parameter for the wake deflection correction defining the horizontal offset at the rotor scaled with yaw error. If the DEFAULT keyword is specified in place of a numerical value, **C_HWkDfl_OY** is set to 0.3.

C_HWkDfl_x [-] (C_{HWkDfl}^x) is the calibrated parameter for the wake deflection correction defining the horizontal offset scaled with downstream distance. If the DEFAULT keyword is specified in place of a numerical value, **C_HWkDfl_x** is set to 0.0.

C_HWkDfl_xY [1/deg] (C_{HWkDfl}^{xY}) is the calibrated parameter for the wake deflection correction defining the horizontal offset scaled with downstream distance and yaw error. If the DEFAULT keyword is specified in place of a numerical value, **C_HWkDfl_xY** is set to -0.004.

C_NearWake ($C_{NearWake}$) [-] is the calibrated parameter for the near-wake correction and must be greater than one. If the DEFAULT keyword is specified in place of a numerical value, **C_NearWake** is set to 1.8.

k_vAmb [-] (k_{vAmb}) is the calibrated parameter for the ambient turbulence influence in the eddy viscosity and must be greater than zero. If the DEFAULT keyword is specified in place of a numerical value, **k_vAmb** is set to 0.05.

k_vShr [-] (k_{vShr}) is the calibrated parameter for the wake shear layer influence in the eddy viscosity and must be greater than zero. If the DEFAULT keyword is specified in place of a numerical value, **k_vShr** is set to 0.016.

C_vAmb_DMin [-] (C_{vAmb}^{DMin}) is a calibrated parameter in the eddy viscosity filter function for ambient turbulence. It defines the transitional diameter fraction between the minimum and exponential regions and must be greater than or equal to zero. If the DEFAULT keyword is specified in place of a numerical value, **C_vAmb_DMin** is set to 0.0.

C_vAmb_DMax [-] (C_{vAmb}^{DMax}) is a calibrated parameter in the eddy viscosity filter function for ambient turbulence. It defines the transitional diameter fraction between the exponential and maximum regions and must be greater than **C_vAmb_DMin**. If the DEFAULT keyword is specified in place of a numerical value, **C_vAmb_DMax** is set to 1.0.

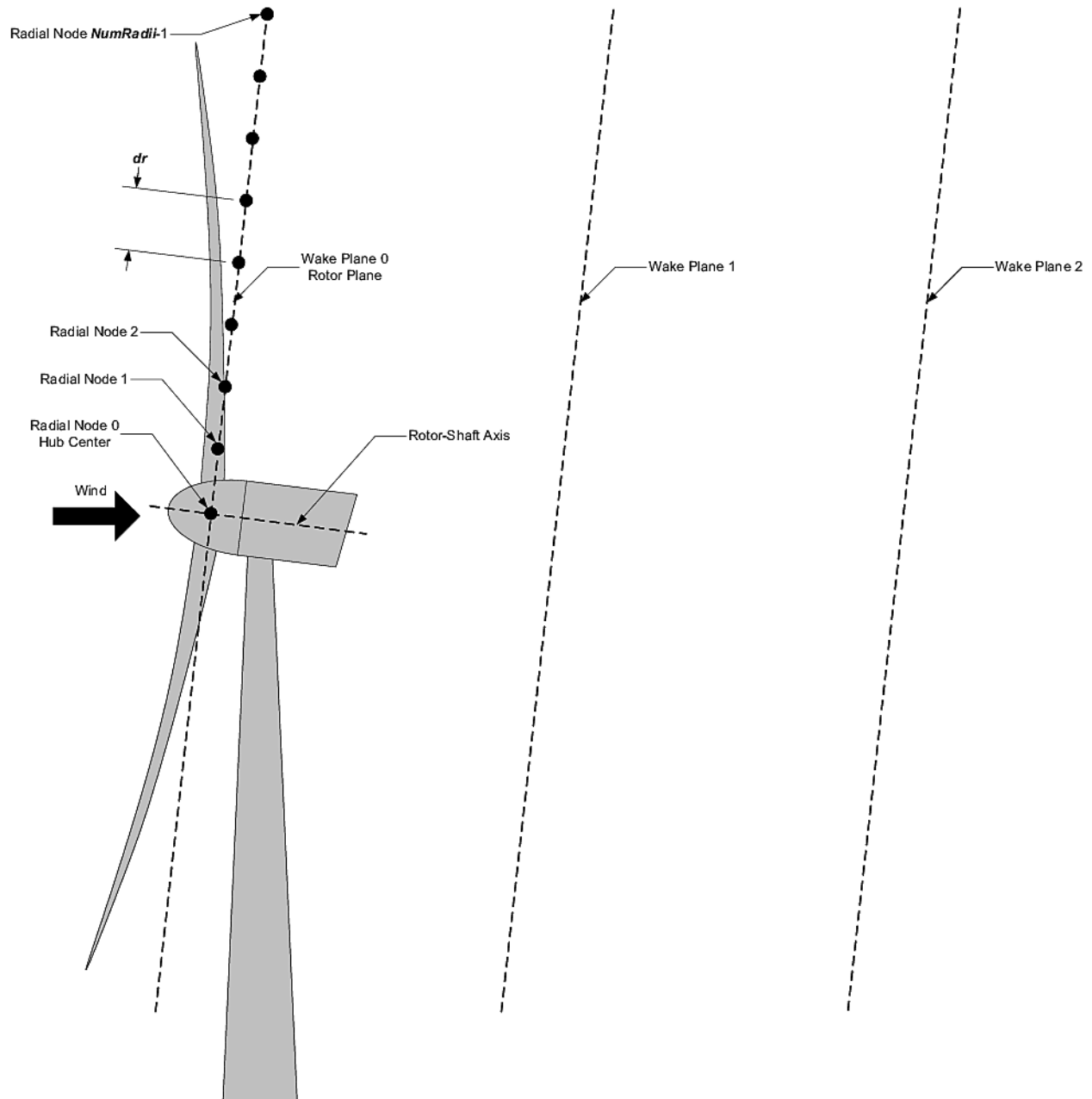


Fig. 4.53: Radial finite-difference grid. For clarity of the illustration, the number and size of the wake planes are shown smaller than they should be.

C_vAmb_FMin [-] ($C_{\nu_{Amb}}^{FMin}$) is a calibrated parameter in the eddy viscosity filter function for ambient turbulence. It defines the value in the minimum region and must be between zero and one (inclusive). If the DEFAULT keyword is specified in place of a numerical value, **C_vAmb_FMin** is set to 1.0.

C_vAmb_Exp [-] ($C_{\nu_{Amb}}^{Exp}$) is a calibrated parameter in the eddy viscosity filter function for ambient turbulence. It defines the exponent in the exponential region and must be greater than zero. If the DEFAULT keyword is specified in place of a numerical value, **C_vAmb_Exp** is set to 0.01.

C_vShr_DMin [-] ($C_{\nu_{Shr}}^{DMin}$) is a calibrated parameter in the eddy viscosity filter function for the wake shear layer. It defines the transitional diameter fraction between the minimum and exponential regions and must be greater than or equal to zero. If the DEFAULT keyword is specified in place of a numerical value, **C_vShr_DMin** is set to 3.0.

C_vShr_DMax [-] ($C_{\nu_{Shr}}^{DMax}$) is a calibrated parameter in the eddy viscosity filter function for the wake shear layer. It defines the transitional diameter fraction between the exponential and maximum regions and must be greater than **C_vShr_DMin**. If the DEFAULT keyword is specified in place of a numerical value, **C_vShr_DMax** is set to 25.0.

C_vShr_FMin [-] ($C_{\nu_{Shr}}^{FMin}$) is a calibrated parameter in the eddy viscosity filter function for the wake shear layer. It defines the value in the minimum region and must be between zero and one (inclusive). If the DEFAULT keyword is specified in place of a numerical value, **C_vShr_FMin** is set to 0.2.

C_vShr_Exp [-] ($C_{\nu_{Shr}}^{Exp}$) is a calibrated parameter in the eddy viscosity filter function for the wake shear layer. It defines the exponent in the exponential region and must be greater than zero. If the DEFAULT keyword is specified in place of a numerical value, **C_vShr_Exp** is set to 0.1.

Mod_WakeDiam [switch] specifies the wake diameter calculation model (method). There are four options: 1) use the rotor diameter [**Mod_WakeDiam=1**]; 2) use a velocity-based method [**Mod_WakeDiam=2**]; 3) use a mass-flux based method [**Mod_WakeDiam=3**]; or 4) use a momentum-flux based method [**Mod_WakeDiam=4**]. If the DEFAULT keyword is specified in place of a numerical value, **Mod_WakeDiam** is set to 1.

C_WakeDiam [-] ($C_{WakeDiam}$) is the calibrated parameter for the wake diameter calculation and must be greater than zero and less than 0.99. It is unused when **Mod_WakeDiam=1**. If the DEFAULT keyword is specified in place of a numerical value, **C_WakeDiam** is set to 0.95.

Mod_Meander [switch] specifies the spatial filter model (method) for wake meandering. There are three options: 1) use a uniform spatial average [**Mod_Meander=1**]; 2) use a truncated *jinc* [**Mod_Meander=2**]; or 3) use a windowed *jinc* [**Mod_Meander=3**]. If the DEFAULT keyword is specified in place of a numerical value, **Mod_Meander** is set to 3.

C_Meander [-] ($C_{Meander}$) is the calibrated parameter for the wake meandering and must be greater than or equal to one. If the DEFAULT keyword is specified in place of a numerical value, **C_Meander** is set to 1.9.

Visualize

WrDisWind [flag] specifies whether full 3D low- and high-resolution disturbed wind data output files will be generated. These files show the ambient wind and wake interactions across the wind farm for visualization and are generated if **WrDisWind=TRUE**. The VTK data format and spatial resolutions (number of grid points, origin, and spacing) of these output files match those of the corresponding low- and high-resolution ambient wind data used by the FAST.Farm simulation. The VTK files are written to a directory named *vtk_ff* where the FAST.Farm primary file is stored. The naming conventions of these output files are *<RootName>.Low.Dis.<n_{low and *<RootName>.HighT<n_{tt for the low- and high-resolution disturbed wind data files, respectively, where *<RootName>* is the name of the FAST.Farm primary input file, excluding its file extension, where *<n_{t and *<n_{low are as specified in [Section 4.2.14](#), but include leading zeros.}*}*}*}*

For visualization, FAST.Farm can also output low-resolution disturbed (including wakes) wind data output files that are two-dimensional (2D) slices of the full low-resolution domain, specified by the following 7 inputs. Up to nine 2D slices parallel to the X-Y, Y-Z, and/or X-Z planes can be output.

- **NOutDisWindXY** [integer] specifies the number of 2D slices parallel to the X-Y plane where low-resolution disturbed wind data output files are output (0 to 9).
- **OutDisWindZ** [m] is a list **NOutDisWindXY** values long of the Z coordinates of each plane that will be output. These values are in the **global inertial-frame coordinate system**, separated by any combination of commas, semicolons, spaces, and/or tabs.
- **NOutDisWindYZ** [integer] specifies the number of 2D slices parallel to the Y-Z plane where low-resolution disturbed wind data output files are output (0 to 9).
- **OutDisWindX** [m] is a list **NOutDisWindYZ** values long of the X coordinates of each plane that will be output. These values are in the **global inertial-frame coordinate system**, separated by any combination of commas, semicolons, spaces, and/or tabs.
- **NOutDisWindXZ** [integer] specifies the number of 2D slices parallel to the X-Z plane where low-resolution disturbed wind data output files are output (0 to 9).
- **OutDisWindY** [m] is a list **NOutDisWindXZ** values long of the Y coordinates of each plane that will be output. These values are in the **global inertial-frame coordinate system**, separated by any combination of commas, semicolons, spaces, and/or tabs.

The VTK files are written to a directory named *vtk_ff* where the FAST.Farm primary file is stored. The naming conventions of these output files are *<RootName>.Low.DisXY<n_{Out}>.<n_{low}>.vtk*, *<RootName>.Low.DisYZ<n_{Out}>.<n_{low}>.vtk*, and *<RootName>.Low.DisXZ<n_{Out}>.<n_{low}>.vtk* for the X-Y, Y-Z, and X-Z slices, respectively, where *<n_{Out}>* is an integer between 1 and 9 corresponding to which slice is output. *<RootName>* and *<n_{low}>* are as defined in [Section 4.2.14](#), but include leading zeros.

WrDisDT [sec] specifies the time step (inverse of the frame rate) of all disturbed wind data output files and must be an integer multiple larger than or equal to **DT_Low**. This input is unused when **WrDisWind** = FALSE and when **NOutDisWindXY**, **NOutDisWindYZ**, and **NOutDisWindXZ** are set to zero. If the DEFAULT keyword is specified in place of a numerical value, **WrDisDT** is set to **DT_Low**. Note that the full high-resolution disturbed wind data output files are not output at a frame rate of 1/**DT_High**, but are only output every **WrDisDT** seconds.

Visualizing the ambient wind and wake interactions can be useful for interpreting results and debugging problems. However, FAST.Farm will generate $n + 1$ files per output option when **WrDisWind** = TRUE and/or when **NOutDisWindXY**, **NOutDisWindYZ**, and/or **NOutDisWindXZ** are set greater than zero. This file generation will slow down FAST.Farm and take up a lot of disk space, especially when generating full low- and high-resolution disturbed wind data files. Therefore, disabling visualization is recommended when running many FAST.Farm simulations. See [Section 4.2.14](#) for visualization output file details.

Output

SumPrint [flag] specifies if a summary file is generated. The file is generated if **SumPrint**=TRUE, with the name *<RootName>.sum*, where *<RootName>* is as defined above. See [Section 4.2.14](#) for summary file details.

ChkptTime [sec] specifies how frequently checkpoint files are written for a potential restart, but **is currently unused by FAST.Farm**.

TStart [sec] specifies the simulation time at which FAST.Farm will begin writing data in the time-series results output file. Note that output files may not be generated at **TStart** seconds if **TStart** is not an integer multiple of **DT_Low**.

OutFileFmt [switch] specifies which type of time-series results output file will be generated. Three options are available, and are the same as those in OpenFAST: 1) generates an ASCII text file [**OutFileFmt**=1]; 2) generates a binary file [**OutFileFmt**=2]; or 3) generates both ASCII text and binary files [**OutFileFmt**=3]. **However, FAST.Farm currently only supports text-based output files. Therefore, OutFileFmt must be set to 1.**

TabDelim [flag] specifies how columns in the ASCII text output time-series results are delimited. If **TabDelim** = TRUE, the columns are tab-delimited. Otherwise, the columns are delimited with spaces. **TabDelim** is not used when **OutFileFmt** = 2.

OutFmt [string] specifies the ASCII text-based output file channel format (excluding the time channel). Values printed in the time-series results output file should result in a field that is 10 characters long; “E10.3E2” is a common setting for **OutFmt**. The time channel is printed using the “F10.4” format. **OutFmt** is not used when **OutFileFmt** = 2. See [Section 4.2.14](#) for details on time-series results files.

FAST.Farm can output wake-related quantities for up to 9 individual turbines, not considering the effects of wake merging, at up to 20 radial nodes and up to 9 downstream distances. These outputs are specified with the 4 following inputs:

- **NOutRadII** [integer] specifies the number of radial nodes to be outputted (0 to 20).
- **OutRadII** [integer] specifies the node numbers between 0 (at the wake center) and **NumRadII**-1 (at the outer extent of the radial finite-difference grid). Values are a list of length **NOutRadII**, separated by any combination of commas, semicolons, spaces, and/or tabs.
- **NOutDist** [integer] specifies the number of downstream distances that output is requested for (0 to 9).
- **OutDist** [m] specifies the downstream distances (not wake-plane numbers) and each must be greater or equal to zero. Values are a list of length **NOutDist**, separated by any combination of commas, semicolons, spaces, and/or tabs. The downstream distances are measured normal to the wake planes and **an OutDist of zero corresponds to the rotor plane**. Wake output quantities are linearly interpolated between wake planes. Only wake-related quantities for the first 9 turbines can be output and all wakes have the same output radial node numbers and downstream distances. The outputs specified in the **OutList** section determine which quantities are actually output at these output radial node numbers and downstream distances.

FAST.Farm can also output ambient wind velocities (not including wakes) and disturbed wind velocities (including wakes) at up to nine points (positions) in the low-resolution wind domain, defined with the following inputs:

- **NWindVel** [integer] specifies the number of points where wind will be output (0 to 9).
- **WindVelX**, **WindVelY**, and **WindVelZ** [m] specifies X, Y, Z and coordinates, respectively, in the **global inertial-frame coordinate system**. Values are lists of length **NWindVel** separated by any combination of commas, semicolons, spaces, and/or tabs. The outputs specified in the **OutList** section determine which wind velocities are actually output at these points.
- **OutList** [quoted strings] controls output quantities generated by FAST.Farm. Enter one or more lines containing quoted strings that in turn contain one or more output parameter names. Separate output parameter names by any combination of commas, semicolons, spaces, and/or tabs. If you prefix a parameter name with a minus sign, “-”; underscore, “_”; or the characters “m” or “M”, FAST.Farm will multiply the value for that channel by -1 before writing the data. The output columns are written in the order they are listed in the input file. FAST.Farm allows for the use of multiple lines so that lists can be broken into meaningful groups and so the lines can be shorter. Comments may be entered after the closing quote on any of the lines. Entering a line with the string “END” at the beginning of the line or at the beginning of a quoted string found at the beginning of the line will cause FAST.Farm to quit scanning for more lines of channel names. Wake-related output quantities are generated for the requested output radial node numbers and downstream distances through the **OutRadII** and **OutDist** lists above. Ambient and disturbed wind velocities are generated for the requested points through the **WindVelX**, **WindVelY**, and **WindVelZ** lists above. If FAST.Farm encounters an unknown/invalid channel name, it warns the users but will remove the suspect channel from the output file. Please refer to [Section 4.2.14](#) for a complete list of possible output parameters.

Ambient Wind Precursor Files in Visualization Toolkit Format

When using ambient wind generated by a high-fidelity precursor simulation with **Mod_AmbWind** = 1, ambient wind data files for both the low- and high-resolution domains must be pre-generated. Each of these ambient wind data files must follow the [simple legacy serial VTK file format](#). A sample VTK-formatted file is given in [Section 4.2.14](#).

FAST.Farm requires that the ambient wind data files be stored in specific subdirectories of the directory specified by **WindFilePath** and with specific file names. The low-resolution ambient wind data files must be stored in a subdirectory named *Low* and be named *Amb.t<n_{low}>.vtk*, where *<n_{low}>* is as specified in [Section 4.2.14](#). The high-resolution ambient wind data files must be stored in a subdirectory named *HighT<n_t>* and be named *Amb.t<n_{high}>.vtk*, where *<n_t>* and *<n_{high}>* are as specified in [Section 4.2.14](#). Subdirectory *HighT<n_t>* should contain the high-resolution ambient wind data corresponding to wind turbine *n_t* specified in the Wind Turbines section of the FAST.Farm primary input file – see [Section 4.2.14](#).

Each VTK-formatted input file begins with a file version and identifier, but is not checked by FAST.Farm. The second line is the header information that is for identifying specific cases, but is not used by FAST.Farm. The third line must include the single word ASCII, designating the file format currently supported by FAST.Farm.

The fourth line must contain the words *DATASET STRUCTURED_POINTS*, designating the data set structure currently supported by FAST.Farm. The next three lines set the spatial discretization of the domain. Each domain is stored as a structured 3D grid of wind data points (representing the corners of 3D cells) in the global *X-Y-Z* inertial-frame coordinate system – as illustrated generically in [Fig. 4.52](#). The number of wind data points in each direction are set by DIMENSIONS followed by three integers separated by white space representing **NX**, **NY**, and **NZ**; the origin of the grid (lowest-most *X-Y-Z* coordinate) is set by ORIGIN followed by three floating real numbers separated by white space representing **X0**, **Y0**, and **Z0**; and the spatial discretization in each direction are set by SPACING followed by three floating real numbers separated by white space representing **dX**, **dY**, and **dZ**. The total domain size has dimensions $(\text{NX}-1)\text{dX} \times (\text{NY}-1)\text{dY} \times (\text{NZ}-1)\text{dZ}$.

The eighth line must contain the word *POINT_DATA* followed by an integer number specifying the number of wind data points, i.e., **NX** × **NY** × **NZ**. The ninth line must contain the word *VECTORS* followed by the data name (not used by FAST.Farm) and *FLOAT*, which defines the format of the data stored on the grid. Alternatively, the ninth line must contain the word *FIELD* followed by the data name (not used by FAST.Farm) and 1 and the tenth line must contain the array name (not used by FAST.Farm) followed by 3, the number of wind data points, i.e., **NX** × **NY** × **NZ**, and *FLOAT*. The remaining **NX** × **NY** × **NZ** lines of the file contain the *X-Y-Z* components of the ambient wind velocity at each wind data point stored as three floating real numbers separated by white space. The first data point corresponds to the *ORIGIN* and the remaining points involve looping through *X*, then *Y*, and then *Z*. For a ground or wave surface that is not flat and level – e.g., complex terrain or time-varying sea-surface elevation for offshore systems – the wind velocity components at a given wind data point should be written as NaN (not a number)¹ if that point is below the surface (not exposed to the atmosphere).

Ambient Wind with InflowWind Module Input Files

When using ambient wind through the interface to the *InflowWind* module with **Mod_AmbWind** = 2 or 3, the ambient wind is specified within standard *InflowWind* input files described in the OpenFAST documentation. The name of the primary *InflowWind* input file is specified by input parameter **InflowFile** in FAST.Farm. Please note that **InflowFile** is independent of the *InflowWind* primary input file used by the OpenFAST model of each wind turbine.

The *InflowWind* primary input file is processed the same when running FAST.Farm simulations as it would when running simulations in stand-alone OpenFAST. The only difference is that input parameter **OutList** in the *InflowWind* primary input file is ignored and replaced with equivalent output settings in FAST.Farm. All wind file type options and their associated input options are supported by FAST.Farm. Wind file type options are specified with input parameter **WindType** in the *InflowWind* primary input file. The available input options include steady wind, uniform time-varying wind, e.g., discrete gusts, and, full-field turbulent wind (in TurbSim, Bladed, and HAWC formats).

¹ FAST.Farm will treat such wind data points as outside the domain, and so, not used in any calculations.

The wind data specified within *InflowWind* must encompass the entire low- and high-resolution domains defined within FAST.Farm for the entire simulation. This is because the ambient wind data specified within *InflowWind* will be interpolated to low- and high-resolution domains for use within FAST.Farm. To ensure this when using full-field turbulent wind data in *InflowWind*, it is recommended that:

- The full-field wind data files be generated periodically so that the wind domain in *InflowWind* effectively extends forever along the wind propagation direction.
- The input parameter **PropagationDir** in the *InflowWind* primary input file be set to 0, ± 90 , or 180 degrees so that the wind propagates along the $\pm X$ or $\pm Y$ axes of the FAST.Farm inertial-frame coordinate system (the exact direction should depend on the orientation of the wind turbines and farm).

When using full-field turbulent wind data in *InflowWind*, it is recommended that the 2D grid where the full-field turbulent wind data are defined be coincident with either the Y - Z grid of the high-resolution domain when **PropagationDir** = 0 or 180 degrees or the X - Z grid of the high-resolution domain when **PropagationDir** = ± 90 degrees for each wind turbine. This is done to avoid doubly interpolating the wind data (once by FAST.Farm when generating the high-resolution domain and once by OpenFAST when accessing high-resolution wind at turbine analysis nodes).

When using ambient wind through multiple instances of the *InflowWind* module, i.e., when **Mod_AmbWind** = 3, only one *InflowWind* input file is specified. However, multiple wind data files are used, each with a different name. Specifically, the file name in the *InflowWind* input file in this case specifically refers only to the directory path of the wind files. The wind file root names are required to be *Low* for the low-resolution domain and *HighTn_t* for the high-resolution domain associated with turbine n_t .² Setting **Mod_AmbWind** to 2 or 3 has no influence when steady inflow is used (**WindType** = 1). When using full-field turbulent wind data in *InflowWind* with **Mod_AmbWind** = 3, it is required that:

- The full-field wind data files be generated periodically. This effectively extends the wind domain forever along the wind propagation direction.
- The input parameter **PropagationDir** in the *InflowWind* input file be set to 0 degrees so that the wind propagates along the X axis of the FAST.Farm inertial-frame coordinate system.
- The wind data files associated with the high-resolution ambient wind be spatially and temporally synchronized with the low-resolution wind data file. The spatial synchronization must be based on the global X - Y - Z offsets of each turbine origin relative to the origin of the inertial frame coordinate system.

OpenFAST Input Files

In addition to the FAST.Farm-specific input files, the OpenFAST model of each wind turbine also requires input files.

WT_FASTInFile [quoted string] specifies the OpenFAST primary input file for each wind turbine, including path. This is required in addition to the FAST.Farm-specific input files. The OpenFAST primary file, in turn, identifies several module-level input files. These OpenFAST input files are described in the OpenFAST documentation. Identical wind turbines can use the same OpenFAST primary input file, except if the corresponding OpenFAST model makes use of a Bladed-style controller in DLL format or, for offshore wind turbines, if different wave conditions are required for each turbine. If a Bladed-style DLL controller is being used, distinct Bladed-style controller DLLs must be used (each with a unique name). This requires the need for distinct *ServoDyn* primary input files, referencing the appropriate DLL name, and distinct OpenFAST primary input files, each referencing the appropriate *ServoDyn* primary input file name. If different wave conditions are required for each turbine, the distinct wave conditions (e.g., based on unique random wave seeds) for each wind turbine must be set in the *HydroDyn* primary input file and distinct OpenFAST primary input files must be used, each referencing the appropriate *HydroDyn* primary input file name.

Please note that the following input parameters in OpenFAST are interpreted differently when running FAST.Farm simulations than when running simulations in stand-alone OpenFAST.

AbortLevel in the OpenFAST primary input file is ignored and replaced with the equivalent input set in the FAST.Farm primary input.

² When HAWC format is used (**WindType** = 5), *_u*, *_v*, *_w* must be appended to the file names.

TMax in the OpenFAST primary input file is ignored and replaced with the equivalent input set in the FAST.Farm primary input.

CompInflow in the OpenFAST primary input file must be set to 1 (to use the *InflowWind* module).

CompAero in the OpenFAST primary input file must be set to 2 (to use the *AeroDyn v15* module).

WindType and its associated input parameters in the OpenFAST *InflowWind* module primary input file are ignored and replaced with the disturbed wind (including wakes) computed across the high-resolution domain for each wind turbine.

PropagationDir in the OpenFAST *InflowWind* module primary input file is ignored.

PCMode, **VSContrl**, **HSSBRMode**, and **YCMode** in the OpenFAST *ServoDyn* module primary input file must not be set to 4 because the Simulink/Labview interface is not currently supported by FAST.Farm.

All input parameters across the various OpenFAST input files pertaining to the wind turbine geometry defined relative to the origin of the OpenFAST inertial-frame coordinate system remain unchanged. Turbine origins are defined as the intersection of the undeflected tower centerline and the ground or, for offshore systems, mean sea level. Note, however, this origin ((0,0,0) in the OpenFAST inertial-frame coordinate system) is located at (**WT_X**,**WT_Y**,**WT_Z**) in the FAST.Farm global X-Y-Z inertial-frame coordinate system.

Output Files

FAST.Farm produces five types of output files: an echo file, a summary file, visualization output files, a time-series results file, and OpenFAST-related files. The following sections detail the purpose and contents of these files.

Echo File

If **Echo** = TRUE in the FAST.Farm primary input file, the contents of the file will be echoed to a file with the naming convention *<RootName>.ech*, where *<RootName>* is as defined in [Section 4.2.14](#). The echo file is helpful for debugging the primary input file. The contents of an echo file will be truncated if FAST.Farm encounters an error while parsing the primary input file. The error usually corresponds to the line after the last successfully echoed line.

Summary File

If **SumPrint** = TRUE in the FAST.Farm primary input file, FAST.Farm will generate a summary file with the naming convention of *<RootName>.sum*. This file summarizes key information about the wind farm model, including the wind turbine locations and OpenFAST primary input files; wake dynamics finite-difference grid and parameters; time steps of the various model components; and the name units and order of the outputs that have been selected.

Visualization Output Files

If **WrDisWind** = TRUE in the FAST.Farm primary input file, FAST.Farm will generate full 3D low- and high-resolution disturbed wind data output files, i.e., the ambient wind and wake interactions across the wind farm for visualization. The VTK data format and spatial resolutions (number of grid points, origin, and spacing) of these output files matches those of the corresponding low- and high-resolution ambient wind data used by the FAST.Farm simulation. The VTK files are written to a directory named *vtk_ff* where the FAST.Farm primary file is stored. The naming conventions of these output files are *<RootName>.Low.Dis.<n_{low and *<RootName>.HighT<n_{thigh for the low- and high-resolution disturbed wind data files, respectively, where *<n_{t, *<n_{low, and *<n_{high are as defined in [Section 4.2.14](#), but with leading zeros.}*}*}*}*}*

- Likewise, if **NOutDisWindXY**, **NOutDisWindYZ**, or **NOutDisWindXZ** are set to be greater than zero in the FAST.Farm primary input file, FAST.Farm will generate low-resolution disturbed wind data (including wakes) output files that are 2D slices of the full low-resolution domain. The 2D slices are parallel to the X-Y, Y-Z, and/or X-Z planes of the global inertial-frame coordinate system, respectively. The VTK files are written to a directory named *vtk_ff* where the FAST.Farm primary file is stored. The naming conventions of these output files are *<RootName>.Low.DisXY<nOut>.<nLow>.vtk*, *<RootName>.Low.DisYZ<nOut>.<nLow>.vtk*, and
- *<RootName>.Low.DisXZ<nOut>.<nLow>.vtk* for the X-Y, Y-Z, and X-Z slices, respectively, where *<nOut>* is as defined in [Section 4.2.14](#), but with leading zeros.

The time step (inverse of the frame rate) of all disturbed wind data files is set by input parameter **WrDisDT** in the FAST.Farm primary input file. Note that the full high-resolution disturbed wind data output files are not output at a frame rate of $1/\text{DT_High}$, but are only output every **WrDisDT** seconds.

Each visualization output file follows the same VTK format used for the ambient wind data files for the high-fidelity precursor simulations. See [Section 4.2.14](#) for details on the file format.

Visualizing the ambient wind and wake interactions can be useful for interpreting results and debugging problems. However, FAST.Farm will generate many files per output option when **WrDisWind** = TRUE and/or when **NOutDisWindXY**, **NOutDisWindYZ**, and/or **NOutDisWindXZ** are set greater than zero. This file generation will slow down FAST.Farm and take up a lot of disk space, especially when generating full low- and high-resolution disturbed wind data files. Therefore, disabling visualization is recommended when running many FAST.Farm simulations.

Time-Series Results File

The FAST.Farm time-series results are written to an ASCII text-based file with the naming convention *<RootName>.out*. The results are in table format, where each column is a data channel with the first column always being the simulation time; each row corresponds to a simulation output time step. The data channels are specified in the **OutList** section of the *OUTPUT* section of the FAST.Farm primary input file. The column format of the FAST.Farm-generated file is specified using the **OutFmt** parameter of the FAST.Farm primary input file.

OpenFAST Output Files

In addition to the FAST.Farm-generated output files, the OpenFAST model of each wind turbine may also generate output files. The various output files that OpenFAST may generate (at both the driver/glue code and module levels, as specified within the OpenFAST input files) are described in the OpenFAST documentation and include summary (*.sum*) files, time-series results (ASCII *.out* or binary *.outb*) files, visualization (*.vtk*) files, etc. FAST.Farm simulations will generate these same files, but with the the path/rootname changed to *<RootName of WT_FASTInFile>.T<n_t>*.

Modeling Guidance

This chapter includes modeling guidance for setting up and running a FAST.Farm simulation. This includes guidance on inflow wind generation; low- and high-resolution grid discretization; parameter selection; super controller use; and solutions for commonly encountered errors.

FAST.Farm Setup Overview

This section includes a high-level overview of how to set up ambient inflow and FAST.Farm simulations in particular, the information needed to calculate various parameters, as shown in Fig. 4.54.

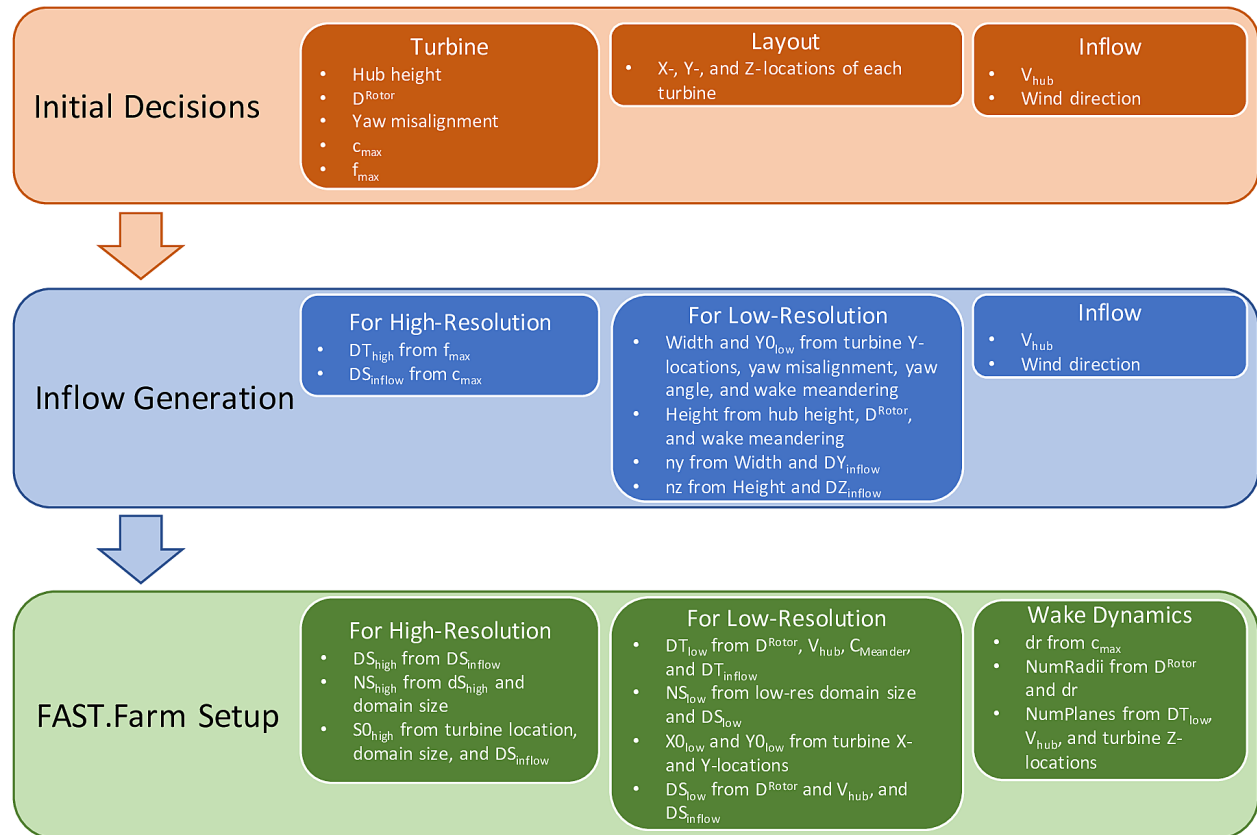


Fig. 4.54: Information flowchart for setting up inflow generation and FAST.Farm simulations. Here, $S = X, Y$, or Z .

Note that this schematic only includes information relevant to FAST.Farm simulations. Typically, additional inflow information is required to generate inflow and the OpenFAST models. The specific equations that should be used to compute the input parameters are discussed in Section 4.2.14. It is highly recommended that the Python notebooks provided in the FAST.Farm [tools repository](#) be used when setting up new inflow or a FAST.Farm case. Improperly setting these parameters can lead to common errors and/or excessive interpolation, which should be avoided. Note that this chapter assumes a wind direction of 0° —i.e., ambient wind that propagates along the $+X$ axis of the global inertial frame coordinate system.

When generating a FAST.Farm simulation setup and corresponding inflow, planning is important. Improper planning could result in FAST.Farm errors and/or needing to regenerate the inflow. Values that should be known *a priori* are:

- wind turbine rotor diameter (D^{Rotor});
- wind turbine hub height;
- maximum turbine chord length (c_{max});
- maximum turbine natural frequency (f_{max});
- X, Y, and Z locations of all turbines in the wind farm;
- desired mean inflow hub-height wind velocity; and
- mean inflow wind direction.

The values that must be computed using this information are:

- inflow and FAST.Farm domain size (height, width, and length);
- FAST.Farm high- and low-resolution domain origin locations (**S0_High** and **S0_Low**, where $S = X, Y$, or Z);
- high- and low-resolution temporal discretization values (**DT_High** and **DT_Low**);
- high- and low-resolution spatial discretization values (**DS_High** and **DS_Low**);
- number of grid points in the high- and low-resolution domains (**NS_High** and **NS_Low**);
- actual mean inflow hub-height wind velocity (V_{hub});
- additional wake dynamics properties (**dr**, **NumRadii**, and **NumPlanes**).

With this information, inflow generation can begin. Though not required, it is recommended to complete inflow generation before setting up the FAST.Farm simulation. This is because the realized spatial discretization values and/or mean hub height velocity can differ from what is desired. Having the correct values of these parameters leads to less interpolation of the wind data in FAST.Farm simulations, which would otherwise reduce the ambient turbulence.

When setting up the inflow generation, the recommended spatial and temporal discretizations should be used, as discussed in [Section 4.2.14](#). If using:

- **Mod_AmbWind** = 1, a high-fidelity must be generated and all discretization values can be specified as the exact desired value.
- **Mod_AmbWind** = 2, a single synthetic inflow (TurbSim or Mann) must be generated using the high-resolution discretization values recommended herein.
- **Mod_AmbWind** = 3, multiple synthetic inflows must be generated. In this case, the recommended high-resolution discretizations should be used for all high-resolution inflows generated. For the low-resolution inflow generation, the recommended high-resolution temporal discretization and low-resolution spatial discretization should be used.

If using synthetic inflow (TurbSim or Mann), the inflow streamwise spatial discretization, **DX_Inflow**, is not specified by the user, but is instead based on Taylor's frozen-turbulence assumption. Because the streamwise discretization of the FAST.Farm domain should be based on the inflow streamwise discretization, the user should compute this value using the inflow time step (**DT_High**) and the advection speed of the synthetic wind data, V_{Advect} . The V_{Advect} may differ from the actual wind speed at hub height, V_{Hub} , as discussed in [Section 4.2.14](#), and should be computed directly from the generated synthetic inflow. Therefore, the exact resulting **DX_Inflow** will not be known until after the inflow has been generated. Additionally, **DX_Inflow** will likely be much smaller than the desired values of **DX_Low** and **DX_High**.

When setting up the FAST.Farm simulation itself, many of the values that were used for inflow generation will be used again here to specify the FAST.Farm domain. Note that this domain specification in FAST.Farm is only needed when using synthetic turbulence inflow. The origin of the low-resolution domain (**X0_Low**, **Y0_Low**, and **Z0_Low**) should be determined based on:

- the minimum turbine X - and Y -locations;
- turbine yaw misalignment;
- inflow wind direction; and
- the expected range of wake meandering.

Specifically, **X0_Low** must accommodate all turbine locations as well as allow enough room to analyze the undisturbed inflow upstream of the wind farm, if desired. **Y0_Low** must accommodate all turbine locations as well as the horizontal wake meandering. When using TurbSim, which cannot generate wind at ground level, **Z0_Low** should be close to but above ground level.

The FAST.Farm domain width and height are then computed using:

- the turbine locations;

- the calculated **Y0_Low** and **Z0_Low** values;
- the horizontal and vertical meandering distance requirements;
- turbine yaw misalignment; and
- the inflow wind direction.

The domain length should be based on the streamwise extent of the wind farm and, if desired, allow enough room to analyze the waked outflow downstream of the wind farm.

The low-resolution domain in FAST.Farm (**DY_Low** and **DZ_Low**) and number of grid points (**NY_Low** and **NZ_Low**) can then be computed using:

- the domain width and height;
- the lateral and vertical spacing of the generated inflow; and
- **DY_Inflow** and **DZ_Inflow**.

The low-resolution temporal discretization (**DT_Low**) should be computed using:

- the turbine diameter;
- inflow hub-height velocity; and
- the inflow temporal discretization.

The streamwise spacing and number of grid points (**DX_Low** and **NX_Low**) should also be based on **DT_Low** and the mean wind speed.

The final domain parameters to calculate are the locations of the high-resolution domains (**X0_High**, **Y0_High**, and **Z0_High**) and the number of grid points required to make up the domains (**NX_High**, **NY_High**, and **NZ_High**). These quantities should be determined from:

- **DS_High** values;
- turbine locations; and
- the size of the high-resolution domains.

The **DS_High** values should be selected based on recommended high-resolution domain discretization criteria, discussed in [Section 4.2.14](#).

Additional wake dynamics quantities are needed when specifying the FAST.Farm input file, as discussed further in [Section 4.2.14](#). It is recommended to base **dr** on c_{max} ; **NumRadii** on wake diameter and **dr**; and **NumPlanes** on **DT_Low**, inflow hub-height velocity, and the distance between turbine locations.

A sample turbine layout and domain locations are shown in [Fig. 4.55](#).

Inflow Wind Generation

This section includes guidelines by which turbulent inflow should be generated for use with FAST.Farm.

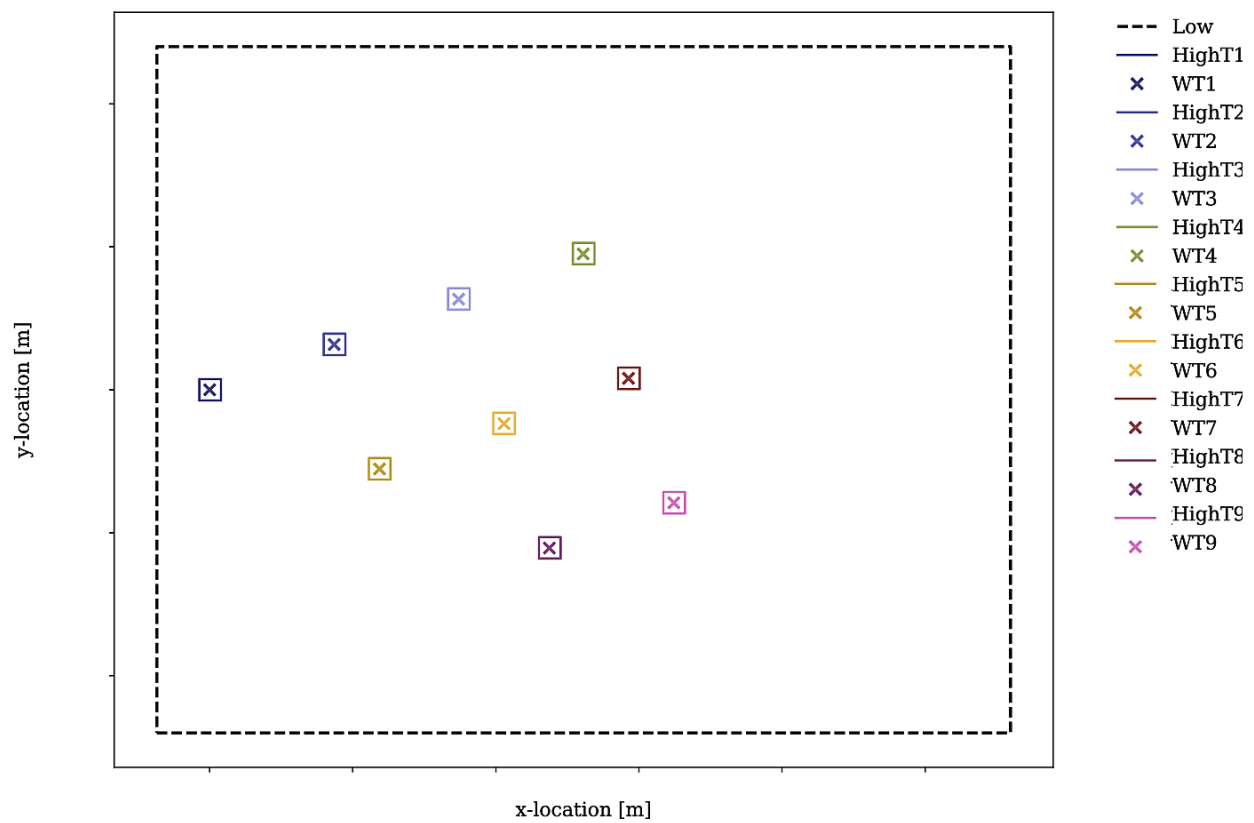


Fig. 4.55: Schematic of example 9-turbine wind farm layout, including low- and high-resolution domains and turbine locations.

High-Fidelity Precursor Ambient Inflow

There are many different methods by which high-fidelity precursor ambient inflow can be generated. This section focuses on generating such inflow using [SOWFA](#).

When using SOWFA to generate FAST.Farm precursor inflow, the *ABLSolver* preprocessor is used. It is important to note the baseline high-fidelity solution is not directly used as inflow for FAST.Farm, but is instead sampled within a specified domain and discretization. This sampling is done through SOWFA and specified in a SOWFA input file. The inflow data are written out in 3D volume VTK-formatted files, as described in [Section 4.2.14](#). These are large ASCII-formatted files; as such, decreasing the precision to, e.g., 3 digits is recommended. The domain size and low-resolution domain discretization used for SOWFA simulations is much larger than what is required for FAST.Farm simulations. Therefore, sampling files must be set up to generate boundary conditions for use with FAST.Farm, based on FAST.Farm discretization suggestions detailed in [Section 4.2.14](#). Two sampling files are needed: one for the low-resolution sampling for the farm-scale domain and one for the high-resolution sampling for the turbine-scale domains. Each sampling file defines the spatial and temporal discretization that will be used in the FAST.Farm simulations. The low-resolution domain file defines a single low-resolution domain that will be used for the FAST.Farm simulations; the high-resolution domain file defines each high-resolution domain that will be used for the FAST.Farm simulations. Thus, it is important to know exactly where all turbines will be located in the FAST.Farm simulation before generating the inflow. Note that this FAST.Farm sampling step can be computationally expensive. Therefore, it is recommended that users make sure all inputs are correct before executing SOWFA, including turbine locations and discretization levels.

An example Python notebook is provided in the FAST.Farm [tools repository](#) to assist in setting up these files for a given FAST.Farm simulation.

Complex Terrain

Complex terrain, or a time-varying sea-surface elevation for offshore systems, can be modeled in FAST.Farm by providing ambient inflow data that are terrain following, e.g., by modeling the surface boundary condition in an LES precursor. The VTK format used by FAST.Farm is spatially uniform. To accommodate complex terrain or waves with a uniform grid, the wind speed for points below the terrain surface should be set to NaN. Any NaN value will be trapped by FAST.Farm and marked as outside of the domain, and so, unused by calculations within the AWAE module. When the ambient wind inflow is terrain following, the wakes will naturally follow the terrain as well, even though FAST.Farm does not include any explicit models for complex terrain, flow recirculation or separation, or local pressure gradients.

If using a SOWFA inflow precursor, the complex terrain is accounted for in the SOWFA inflow precursor generation, and so, no modification to the *vtk* files is required to account for complex terrain when sampling for a FAST.Farm simulation.

Synthetic Turbulence Ambient Inflow

Synthetically generated turbulent inflow can be used in FAST.Farm to accurately predict turbine response and wake dynamics across different atmospheric conditions. There are several ways to achieve this; any method can be used as long as it produces an output file in a format supported by *InflowWind*. Modeling guidance for TurbSim and the Mann model are discussed next.

TurbSim

When using the NREL tool **TurbSim v2**, different options are available to drive the synthetic turbulence towards specific desired outcomes, e.g.;

1. standard or user-defined time-averaged wind profile (shear, veer);
2. standard or user-defined velocity spectra in three directions (along the wind, u, and transverse, v and w);
3. standard or user-defined spatial point-to-point coherence; and
4. standard or user-defined component-to-component correlations (Reynolds stresses).

Additionally, TurbSim v2 allows the user to generate turbulent wind that is consistent with user-defined three-component wind time series at one or more points (i.e., constrained wind). These options can be used separately or in some combination (though user-defined spectra and user-defined time series can not be used together). When defined appropriately, all these methods can result in good statistical comparison of turbine response and wake dynamics between FAST.Farm results and a reference data set, e.g., compared with an LES precursor or physically measured inflow. However, attention must be paid when generating these inflows to ensure atmospheric conditions are modeled properly.

In particular, TurbSim generates wind velocities transversely throughout the domain using u-, v-, and w-spatial-coherence models based on a selection of coherence model equations and their associated parameters. These models and parameters can either be specified explicitly or left as *default* values in TurbSim. When the IEC spatial-coherence model is selected, spatial coherence is computed using Eq. (4.200) ([ff-Jon14]).

$$Coh_{i,jK}(f) = \exp \left(-a_K \sqrt{\left(\frac{fr}{V_{Advect}} \right)^2 + (rb_K)^2} \right) \quad (4.200)$$

where V_{Advect} is the average wind speed at the hub height specified in TurbSim, which is also the advection speed in *InflowWind*; $Coh_{i,jK}$ is the spatial coherence between points i and j for the velocity components $K = u, v, w$; r is the distance between points i and j ; a_K is the coherence decrement parameter; and b_K is the coherence offset parameter. It was discovered in [ff-Seal19b] that the use of the IEC coherence model with default coherence parameters together with the IEC Kaimal spectra results in negligible wake meandering. This is because the default v- and w-coherence parameters in TurbSim are set such that a_K are very large numbers and $b_K = 0$, effectively resulting in no coherence ($Coh_{i,jK}(f) = 0$) ([ff-Jon14]).¹ This lack of meandering is nonphysical and will have a nonphysical impact on the response of downstream turbines. Instead of using the default values, the v- and w-coherence parameters were specified in [ff-Seal19b] to identically equal the u-coherence parameters specified in the IEC standard, such that: $SCMod2 = SCMod3 = IEC$; $a_K = 12.0$ and $b_K = 0.00035273 \text{ m}^{-1}$; and $CohExp = 0.0$. ([ff-Jon14]). Properly setting spatial coherence parameters for the transverse wind velocity components is necessary to accurately predict wake meandering. It is also important to note that, in TurbSim, the a_K and b_K values must be specified within quotation marks (e.g., "12.0 0.00035273") or, at present, the values are set to 0.

When using TurbSim to generate the full-field turbulent wind data for FAST.Farm, one often wants the TurbSim grid to extend well above the hub height to capture vertical wake meandering due to the w component of turbulence. Because TurbSim requires that **HubHt** > 0.5***GridHeight**, it is often necessary to specify an artificially high **HubHt** in TurbSim. To properly set the **HubHt** parameter, the following equation is suggested:

$$\text{HubHt} = z_{\text{bot}} + \text{GridHeight} - 0.5D_{\text{grid}}$$

¹ TurbSim effectively neglects the spatial v- and w-coherence in the default IEC case because these are not prescribed by the IEC design standards.

where z_{bot} is the desired bottom vertical location of the grid (just above ground level) and $D_{\text{grid}} = \text{MIN}(\text{GridWidth}, \text{GridHeight})$. Note that the **HubHt** parameter is used by TurbSim as the reference height for the wind speed used (V_{Advect}) to define the wind-speed standard deviations and spatial coherence in the IEC turbulence models, as well as the advection speed (in *InflowWind*) for all models. Thus, the resulting wind-speed standard deviations and spatial coherence in the IEC turbulence models will not be what is expected without explicit consideration of the difference in wind speeds between the **HubHt** used by TurbSim and the actual turbine hub height. The advection speed (in *InflowWind*) will likely also be faster than it would be when the actual hub height speed is used. A separate reference height (**RefHt**) is specified in TurbSim, which is the height at which, e.g., the reference wind speed is enforced. This value is also used to properly set the power law velocity profile. Future work is needed to [decouple the HubHt parameter from the TurbSim grid generation](#).

It is generally recommended that the full-field wind data files be generated periodically. This effectively extends the wind domain forever along the wind propagation direction.

When using ambient wind through multiple instances of the *InflowWind* module, i.e., when **Mod_AmbWind** = 3, only one *InflowWind* input file is specified. However, multiple wind data files are used, each with a different name. Specifically, the file name in the *InflowWind* input file in this case refers only to the directory path of the wind files. The wind file root names are required to be *Low* for the low-resolution domain and *HighT* for the high-resolution domain associated with turbine n_t .² When steady inflow in *InflowWind* is used (**WindType** = 1), setting **Mod_AmbWind** to 2 or 3 produces identical results. When using full-field turbulent wind data in *InflowWind* with **Mod_AmbWind** = 3, it is required that:

- The full-field wind data files be generated periodically. This effectively extends the wind domain forever along the wind propagation direction.
- The input parameter **PropagationDir** in the *InflowWind* input file be set to 0 degrees so that the wind propagates along the *X* axis of the FAST.Farm inertial-frame coordinate system.
- The wind data files associated with the high-resolution ambient wind be spatially and temporally synchronized with the low-resolution wind data file. The spatial synchronization must be based on the global *X-Y-Z* offsets of each turbine origin relative to the origin of the inertial frame coordinate system. For each wind turbine, the velocity time series at the turbine location should be extracted from the low-resolution TurbSim domain. To account for turbine downstream distance, each time series should then be offset in time based on the freestream velocity and turbine location. This time series should then be used to generate the high-resolution TurbSim inflow for each turbine. The TurbSim user's manual contains details on how to generate a TurbSim inflow using a specified time series [ff-Jon14].

Mann Model

When generating stochastic turbulence with the Mann model, 11 user-defined inputs are required: **prefix**, **alpha_epsilon**, **L**, **gamma**, **seed**, **nx**, **ny**, **nz**, **dx**, **dy**, and **dz**. The parameters that should be selected in conjunction with FAST.Farm parameters are discussed here.

dx, **dy**, and **dz** – These parameters should be selected based on the high-resolution spatial discretization recommendations discussed below in [Section 4.2.14](#).

nx – This value is required to be a power of 2. To ensure no repetition of the turbulence box for the duration of the simulation, the following equation is recommended:

$$\mathbf{nx} = 2^{\text{CEILING}\left[\log_2\left(\frac{V_{\text{Advect}} T_{\text{Max}}}{\mathbf{dx}}\right)\right]}$$

where V_{Advect} is the advection speed of the Mann box and $\text{CEILING}[x]$ rounds x to the next highest integer. This equation ensures that the turbulence box will not repeat during the simulation and also that the power of two criteria is satisfied.

² When HAWC format is used (**WindType** = 5), *_u*, *_v*, *_w* must be appended to the file names.

ny and **nz** – These values are also required to be powers of 2. With this requirement in mind, these values should be selected to ensure the entire desired domain width (Y) and height (Z) are captured, as discussed below in [Section 4.2.14](#).

The *InflowWind* input file has a specific section for using a Mann turbulence box. This section requires the input of **nx**, **ny**, **nz**, **dx**, **dy**, **dz**, **RefHt**, and **URef**. These values should be specified exactly as those used to generate the inflow. Note that **dx**, **dy**, and **dz** specified in *InflowWind* should be the same as **dx_High**, **dy_High**, and **dz_High** in FAST.Farm, respectively. **RefHt** should be defined as follows:

$$\text{RefHt} = 0.5\text{dz}(\text{nz} - 1) + z_{\text{bot}}$$

where **URef** is the mean wind speed at the reference height, and dictates the advection speed of the Mann box, identified here as V_{Advect} .

When using a Mann box, it is important to know that **the x-axis direction is opposite the convention used by InflowWind. Although the interpretation in InflowWind (including OpenFAST and FAST.Farm) is consistent with how Mann boxes are used in other aeroelastic software, the interpretation is nonphysical.** If desired, the user can adjust the FAST.Farm source code to read the x-axis in reverse. Correcting this error universally across all aeroelastic software that use Mann boxes is needed [future work](#).

Low- and High-Resolution Domain Discretization

Spatial and temporal discretization can affect wake meandering, turbine structural response, and resulting wake and load calculations. This section summarizes recommendations for discretization values in terms of geometry and wind speed that will ensure a converged solution, while maximizing computational efficiency. For details on how these recommendations were formed, see [\[ff-Seal19a\]](#). Though developed for FAST.Farm use, these guidelines are likely applicable to any DWM-type model or aeroelastic analysis.

Low-Resolution Domain

The low-resolution domain in FAST.Farm is primarily responsible for wake meandering and merging. As such, convergence was assessed by comparing trends in standard deviation of horizontal and vertical meandering wake center positions for the wakes behind each turbine at various distances downstream. It was found that the mean horizontal and vertical wake trajectories have negligible dependence of **DT_Low** or **DS_Low**. The following equation can be used to ensure convergence of wake meandering in the low-resolution domain:

$$\text{DT_Low} \leq \frac{C_{\text{Meander}} D^{\text{Wake}}}{10V_{\text{Hub}}}$$

This equation is based on the low-pass cutoff frequency for wake meandering $\left(\frac{V_{\text{Hub}}}{C_{\text{Meander}} D^{\text{Wake}}}\right)$ from [\[ff-Leal08\]](#) (in which $C_{\text{Meander}} = 2$, but C_{Meander} defaults to 1.9 in FAST.Farm) and effectively specifies that the highest frequency of wake meandering should be resolved by at least 10 time steps. Note that D^{Wake} can be approximated as D^{Rotor} in this calculation.

Spatial discretization convergence was assessed in the same manner as temporal discretization. Minimal sensitivity to spatial discretization was found for the low-resolution domain in the range of spatial discretizations considered. Nonetheless, the following equation is recommended for identifying the maximum suggested **DS_Low**, where S refers to X , Y , or Z and the denominator has the units [m/s]:

$$\text{DS_Low} \leq \frac{C_{\text{Meander}} D^{\text{Wake}} V_{\text{Hub}}}{150\text{m/s}} = \frac{\text{DT_Low} V_{\text{Hub}}^2}{15\text{m/s}}$$

For all synthetic turbulence methods, it is recommended that **DX_Low** = $V_{\text{Advect}} \text{DT_Low}$ to avoid interpolating in X -direction. Note the use of the advection speed, V_{Advect} , to calculate **DX_Low**, rather than the actual hub-height wind speed, V_{Hub} . Additionally, **X0_Low** should be an integer multiple of **DX_Low**.

High-Resolution Domain

The high-resolution wind domain in FAST.Farm is primarily responsible for ambient and waked inflow local to a turbine. As such, convergence was assessed by comparing trends in mean and standard deviation of turbine structural motions and loads for each turbine.

Required discretization levels vary depending on the quantity of interest. Thus, it is important to decide what structural components will be considered when selecting a high-resolution discretization level. Most notably, tower-base moments are the most sensitive to **DT_High**, whereas generator power and blade deflections and moments show little dependence on this value. To capture the full structural response, **DT_High** should be selected based on the highest frequencies influencing the structural excitation, including rotational sampling of turbulence and response, i.e., natural frequencies, of the pertinent structural components, f_{\max} (in Hz), as in Equation (4.201), where the factor of 2 is taken from the Nyquist sampling theorem. This is a frequently used rule of thumb in wind turbine aeroelastic analysis under excitation from turbulent inflow.

$$\mathbf{DT_High} \leq \frac{1}{2f_{\max}} \quad (4.201)$$

The required **DS_High** approximately corresponds to the maximum blade chord length of the turbine, c_{\max} , as in Equation (4.202). Selecting a **DS_High** equivalent to this value has long been a rule-of-thumb in wind turbine aeroelastic analysis under excitation from turbulent inflow.

$$\mathbf{DS_High} \leq c_{\max} \quad (4.202)$$

Parameter Selection

Setting up a FAST.Farm simulation can involve specifying a large number of parameters, especially if the *InflowWind* module is used for the ambient wind. This section summarizes best practices for selecting some of these parameters. References are made to desired versus realized values. The discrepancies between these values are discussed in Section 4.2.14.

InflowWind Domain Parameters

Care must be taken when setting up a FAST.Farm simulation using the *InflowWind* ambient wind inflow option. It is highly recommended that the distributed [Python notebooks](#) be used when setting up a new case. Improperly setting these parameters can lead to common errors and/or excessive interpolation, which should be avoided. The methods and rules of thumb that are used in those Python notebooks are also discussed here.

Low-Resolution Domain

NX_Low, **NY_Low**, **NZ_Low** – These quantities should be based on **DS_Low** and the desired domain size (S_{dist_Low}), where $S=X, Y$ or Z . This integer quantity should be computed as:

$$\mathbf{NS_Low} = \mathit{CEILING} \left(\frac{S_{dist_Low}}{\mathbf{DS_Low}} \right) + 1$$

X0_Low – This quantity must be less than the X location of the furthest upstream turbine. It is recommended to set this value further upstream to allow for analysis of the ambient inflow. If using a Mann box, this value should be 0.

Y0_Low – This quantity must be less than the lowest Y location of any turbine (**WT_Y_**). Additional clearance is required to accommodate wake meandering, wake deflection, and spatial averaging used in the *AWAE* module. This value may be computed as:

$$\mathbf{Y0_Low} \leq \mathbf{WT_Y_min} - 3D^{\text{Rotor}}$$

Additional clearance should be allowed for appreciable wake meandering and/or yaw. For **Mod_AmbWind** = 2, the synthetic inflow data are centered around $Y=0$. Because of this, **Y0_Low** should equal $-Y_{dist_Low}/2$. This is the same for the low-resolution domain with **Mod_AmbWind** = 3.

Z0_Low – It is recommended that this value be set close to but above ground level. When using TurbSim, this value can not be at or below ground level because TurbSim cannot generate wind at these locations.

DX_Low, DY_Low, DZ_Low – Desired spatial values are not discussed here, as they are covered in detail in [Section 4.2.14](#). However, the actual quantities used might differ from the desired values when using synthetic inflow, as discussed in [Section 4.2.14](#). To determine the actual quantity, the following equation is suggested when using synthetic inflow:

$$\mathbf{DS_Low} = \text{FLOOR} \left(\frac{\mathbf{DS_Low_Desired}}{\mathbf{DS_High}} \right) * \mathbf{DS_High}$$

Use of this equation is the best way to ensure that **DS_Low** will be a multiple integer of **DS_High**, reducing interpolation smoothing.

High-Resolution Domain

Xdist_High, Ydist_High, Zdist_High – Though not direct inputs, these lengths, widths, and heights of the high-resolution domains should be selected based on the size and location of the turbines. The following values are recommended:

If tower aerodynamic loads are desired, the high-resolution domain should span the entire tower and rotor:

$$\mathbf{Zdist_High} = \mathbf{HubHt} + \frac{1.1 D^{\text{Rotor}}}{2}$$

These parameters might need to be increased to account for large structural motion, such as for floating offshore wind applications.

NX_High, NY_High, NZ_High – These quantities should be based on **DS_High** and the desired domain size (*Sdist_High*), where $S=X, Y, \text{ or } Z$. This integer quantity should be computed as:

X0_High, Y0_High, Z0_High – These quantities are set for each turbine. They should be based on turbine location and set so that the turbine is contained inside the high-resolution domain. It is recommended that **X0_High** and **Y0_High** are set approximately $1.1 D^{\text{Rotor}}/2$ lower than the turbine location. For the high-resolution domains with **Mod_AmbWind** = 3, the synthetic inflow data are centered around each turbine, based on **WT_X/Y/Z**.

DX_High, DY_High, DZ_High – Desired spatial values are not discussed here, as they are covered in detail in [Section 4.2.14](#).

Wake Dynamics Parameters

Wake dynamics parameters define the axisymmetric finite-difference grid used for each wake plane. These planes are defined by the following parameters:

- **dr** – This value should be set so that FAST.Farm sufficiently resolves the wake deficit within each plane. The following value is suggested:

$$\mathbf{dr} \leq c_{\max}$$

- **NumRadii** – To ensure the wake deficits are accurately computed by FAST.Farm, **NumRadii** should be set so that the diameter of each wake plane, $2(\mathbf{NumRadii}-1)\mathbf{dr}$, is large relative to the rotor diameter. The following value is suggested:

$$\mathbf{NumRadii} \geq \frac{3D^{\text{Rotor}}}{2 \mathbf{dr}} + 1$$

- **NumPlanes** – To ensure the wake deficits are accurately captured by FAST.Farm, **NumPlanes** should be set so that the wake planes propagate a sufficient distance downstream, preferably until the wake deficit decays away (x_{dist}), with typical values between $10 - 20 \times D^{\text{Rotor}}$. The following value is suggested:

$$\text{NumPlanes} \geq \frac{x_{\text{dist}}}{\text{DT_Low} \bar{V}}$$

where \bar{V} is the average convection speed of the wake, which can be approximated as

$$\bar{V} = V_{\text{Hub}} \left(1 - \frac{\bar{a}}{2} \right)$$

where \bar{a} is the time- and spatial-temporal-average of the axial induction at the rotor disk. \bar{a} is expected to be around 1/3 below rated wind speed (for optimal aerodynamic efficiency) and decreases above rated wind speed to near zero before the cut-out wind speed.

Note that because new wake planes are added each time step as the simulation begins, increasing **NumPlanes** will also increase the initial transient time of the simulation. The start-up transient time is estimated by Equation (4.203).

$$t_{\text{startup}} = \text{DT_Low}(\text{NumPlanes} - 2) \quad (4.203)$$

- **Mod_WakeDiam** – A value of **1** is recommended. For further details on the options for this parameter, see Equation (4.223).
- **Mod_Meander** – A value of **3** is recommended. For further details on the options for this parameter, see Equation (4.230).

The remaining 20 inputs are user-specified calibration parameters and options that influence the wake-dynamics calculations. The parameters may depend, e.g., on turbine operation or atmospheric conditions that can be calibrated to better match experimental data or by using an HFM benchmark. Default values have been derived for each calibrated parameter based on **SOWFA** simulations for the NREL 5MW turbine ([ff-Deal18]), but these can be overwritten by the user.

Super Controller

When **UseSC** is set to **TRUE**, the super controller is enabled. The super controller code must be compiled as a dynamic library file – a *.dll* file in Windows or a *.so* file in Linux or Mac OS. This super controller dynamic library is essentially identical to the super controller available in **SOWFA**. The super controller is used in conjunction with individual wind turbine controllers defined in the style of the DISCON dynamic library of the DNV GL's Bladed wind turbine software package, with minor modification.

The inputs to the super controller are commands or measurements from individual turbine controllers.³ The outputs of super controller module are the global controller commands and individual turbine controller commands.

The super controller dynamic library must be compiled with five procedures, whose arguments are outlined in Table 4.17.

³ The super controller also has as input a placeholder for future global (e.g., wind) measurements in addition to commands or measurements from the individual turbine controllers. But the global inputs are currently null.

Table 4.17: Arguments for Each Procedure of the Super Controller Dynamic Library

Procedure	Inputs	Outputs	Comments
sc_init	<ul style="list-style-type: none"> nTurbines 	<ul style="list-style-type: none"> nInpGlobal NumCtrl2SC NumParamGlobal NumParamTurbine NumStatesGlobal <ul style="list-style-type: none"> NumStatesTurbine NumSC2CtrlGlob NumSC2Ctrl errStat errMsg 	<ul style="list-style-type: none"> Set numbers of inputs, outputs, states, and parameters nInpGlobal must currently be set to zero in FAST.Farm
sc_getinitData	<ul style="list-style-type: none"> nTurbines NumParamGlobal NumParamTurbine NumSC2CtrlGlob NumSC2Ctrl NumStatesGlobal <ul style="list-style-type: none"> NumStatesTurbine 	<ul style="list-style-type: none"> <ul style="list-style-type: none"> ParamGlobal(1:NumParamGlobal) <ul style="list-style-type: none"> ParamTurbine(1:NumParamTurbine*nTurbines) <ul style="list-style-type: none"> from_SCglob(1:NumSC2CtrlGlob) <ul style="list-style-type: none"> from_SC(1:NumSC2Ctrl*nTurbines) <ul style="list-style-type: none"> from_SCglob(1:NumSC2CtrlGlob) <ul style="list-style-type: none"> from_SC(1:NumSC2Ctrl*nTurbines) <ul style="list-style-type: none"> StatesGlob(1:NumStatesGlobal) <ul style="list-style-type: none"> StatesTurbine(1:NumStatesTurbine*nTurbines) errStat errMsg 	<ul style="list-style-type: none"> Set parameters and initialize states at time zero Initial outputs are not currently used by FAST.Farm
sc_calcOutputs	<ul style="list-style-type: none"> nTurbines NumParamGlobal <ul style="list-style-type: none"> ParamGlobal(1:NumParamGlobal) NumParamTurbine <ul style="list-style-type: none"> ParamTurbine(1:NumParamTurbine*nTurbines) nInpGlobal <ul style="list-style-type: none"> to_SCglob(1:nInpGlobal) NumCtrl2SC <ul style="list-style-type: none"> to_SC(1:NumCtrl2SC*nTurbines) NumStatesGlobal <ul style="list-style-type: none"> StatesGlob(1:NumStatesGlobal) <ul style="list-style-type: none"> NumStatesTurbine <ul style="list-style-type: none"> StatesTurbine(1:NumStatesTurbine*nTurbines) 	<ul style="list-style-type: none"> <ul style="list-style-type: none"> from_SCglob(1:NumSC2CtrlGlob) <ul style="list-style-type: none"> from_SC(1:NumSC2Ctrl*nTurbines) errStat errMsg 	<ul style="list-style-type: none"> Calculate outputs at the current time step nInpGlobal is currently zero in FAST.Farm to_SCglob is currently null in FAST.Farm
4.2. Module Documentation	<ul style="list-style-type: none"> NumSC2CtrlGlob NumSC2Ctrl 		371
sc_updateStates			

To interact with the super controller, the individual turbine controllers within each instance of OpenFAST must also be compiled as a dynamic library. The single procedure, DISCON, is unchanged from the standard DISCON interface for the Bladed wind turbine software package, as defined by DNV GL, but with three extra arguments, as outlined in [Table 4.18](#).

Table 4.18: Arguments of the DISCON Procedure for Individual Turbine Controller Dynamic Library, Updated for the Super Controller

Procedure	Inputs	Outputs	Comments
DISCON	<ul style="list-style-type: none"> • avrSWAP(*) • from_SCglob(1:NumSC2CtrlGlob, 1:NumCtrl2SC) • from_SC(1:NumSC2Ctrl) • accInFILE • avcOUTNAME 	<ul style="list-style-type: none"> • avrSWAP(*) • to_SCglob • aviFAIL • avcMSG 	<ul style="list-style-type: none"> • New inputs: from_SCglob and from_SC • New output: to_SC

Note that at time zero, the super controller output calculation (sc_calcOutputs) is called before the call to the individual turbine controllers (DISCON). So, the initial outputs from the super controller (from_SC, from_SCglob) are sent as inputs to the individual turbine controllers, but the initial inputs to the super controller from the individual turbine controller outputs (to_SC) at time zero are always zero. At subsequent time steps, the individual turbine controllers (DISCON) are called before the output calculation of the super controller (sc_calcOutputs). As a result, at each time step other than time zero, the outputs from the super controller (from_SC, from_SCglob) are extrapolated in time based on past values within *OF* before being sent as input to the individual turbine controllers. Thus, care should be taken to ensure that the outputs from the super controller (from_SC, from_SCglob) vary smoothly over time (without steps). See [Figure Fig. 4.57](#) for more information.

Commonly Encountered Errors

This section covers errors that have been commonly encountered by users during the development, verification, and use of FAST.Farm. Submit any additional errors or questions to the [NWTC forum](#).

InflowWind Errors

InflowWind errors tend to be related to improperly setting the high- or low-resolution domain sizes. Two such common errors are detailed here.

Turbine Leaving the Domain

The following error is commonly encountered:

```
T<n_t>:<routine name>:FAST_Solution0:CalcOutputs_And_SolveForInputs:
SolveOption2:InflowWind_CalcOutput:CalcOutput:IfW_4Dext_CalcOutput
[position=(-1.8753, 0, 32.183) in wind-file coordinates]:Interp4D:Outside
the grid bounds.
```

This error occurs when a turbine leaves the specified high-resolution domain. This typically happens through improper domain specification or large blade deflections/structural motions. Note that coordinates in this error are in the local frame of reference of the turbine and are case dependent.

If the cause is improper domain specification, the error will trigger in the initialization stage of the simulation (*<routine name>=FARM_InitialCO:FWrap_t0*). In this case, a review of the primary FAST.Farm input file is suggested. In particular, the values of **NX_High**, **NY_High**, **NZ_High**, **X0_High**, **Y0_High**, **Z0_High**, **dX_High**, **dY_High**, and **dZ_High**, as these parameters define the size and location of the high-resolution domain. Note that the error specifies which turbine ($T<n_t>$) the error has occurred for, which will aid in debugging where the error is.

If the cause is large blade deflection or structural motion, the error will trigger at some point during the simulation (*<routine name>=FARM_UpdateStates:FWrap_t0*). In this case, increasing the overall size of the high-resolution domain could alleviate this problem. However, the user should first confirm that such large deflections/motions are expected and realistic and not due to a turbine modeling error.

Undefined Location

The following error is commonly encountered:

```
Farm_Initialize:InflowWind_CalcOutput:CalcOutput:IfW_TSFFWind_CalcOutput
[position=(5, 565, 5) in wind-file coordinates]: FF wind array boundaries
violated: Grid too small in Y direction. Y=565; Y boundaries =
[-555, 555]
```

This error occurs when FAST.Farm tries to access a point in the low-resolution domain that is not contained in the ambient wind file. Note that coordinates in this error are in the global frame of reference and are case dependent. For this error, a review of the primary FAST.Farm input file is suggested. In particular, the values of **NX_Low**, **NY_Low**, **NZ_Low**, **X0_Low**, **Y0_Low**, **Z0_Low**, **dX_Low**, **dY_Low**, and **dZ_Low**, as these parameters define the size and location of the low-resolution domain. The error specifies along which axis the error has occurred, aiding in debugging.

FAST.Farm Theory

FAST.Farm is a multiphysics engineering tool for predicting the performance and loads of wind turbines within a wind farm. FAST.Farm uses [OpenFAST](#) to solve the aero-hydro-servo-elastic dynamics of each individual turbine, but considers additional physics for wind-farm-wide ambient wind in the atmospheric boundary layer; a wind-farm super controller; and wake deficits, advection, deflection, meandering, and merging. FAST.Farm is based on the principles of the DWM model – including passive tracer modeling of wake meandering – but addresses many of the limitations of previous DWM implementations.

Dynamic Wake Meandering Principles and Limitations Addressed

The main idea behind the DWM model is to capture key wake features pertinent to accurate prediction of wind farm power performance and wind turbine loads, including the wake-deficit evolution (important for performance) and the wake meandering and wake-added turbulence (important for loads). Although fundamental laws of physics are applied, appropriate simplifications have been made to minimize the computational expense, and HFM solutions are used to inform and calibrate the submodels. In the DWM model, the wake-flow processes are treated via the “splitting of scales,” in which small turbulent eddies (less than two diameters) affect wake-deficit evolution and large turbulent eddies (greater than two diameters) affect wake meandering.

The presence of thrust from the wind turbine rotor causes the wind speed to decrease and the pressure to increase just upwind of the rotor. In the near-wake region just downwind of the rotor – illustrated in [Fig. 4.56](#) – coherent vortices break down, the pressure recovers to free stream, the wind speed decreases further, and the wake expands radially. In the far-wake region further downwind, the wake deficit is approximately Gaussian and recovers to free stream due to the turbulent transfer of momentum into the wake from the ambient wind across the wake shear layer. This flow-speed reduction and gradual recovery to free stream is known as the wake-deficit evolution. In most DWM implementations, the wake-deficit evolution is modeled via the thin shear-layer approximation of the Reynolds-averaged Navier-Stokes

equations under quasi-steady-state conditions in axisymmetric coordinates – illustrated in Fig. 4.48. The turbulence closure is captured by using an eddy-viscosity formulation, dependent on small turbulent eddies. This wake-deficit evolution solution is only valid in the far wake. This far wake is most important for wind farm analysis because wind turbines are not typically spaced closely. However, because the wake-deficit evolution solution begins at the rotor, a near-wake correction is applied at the inlet boundary condition to improve the accuracy of the far-wake solution.

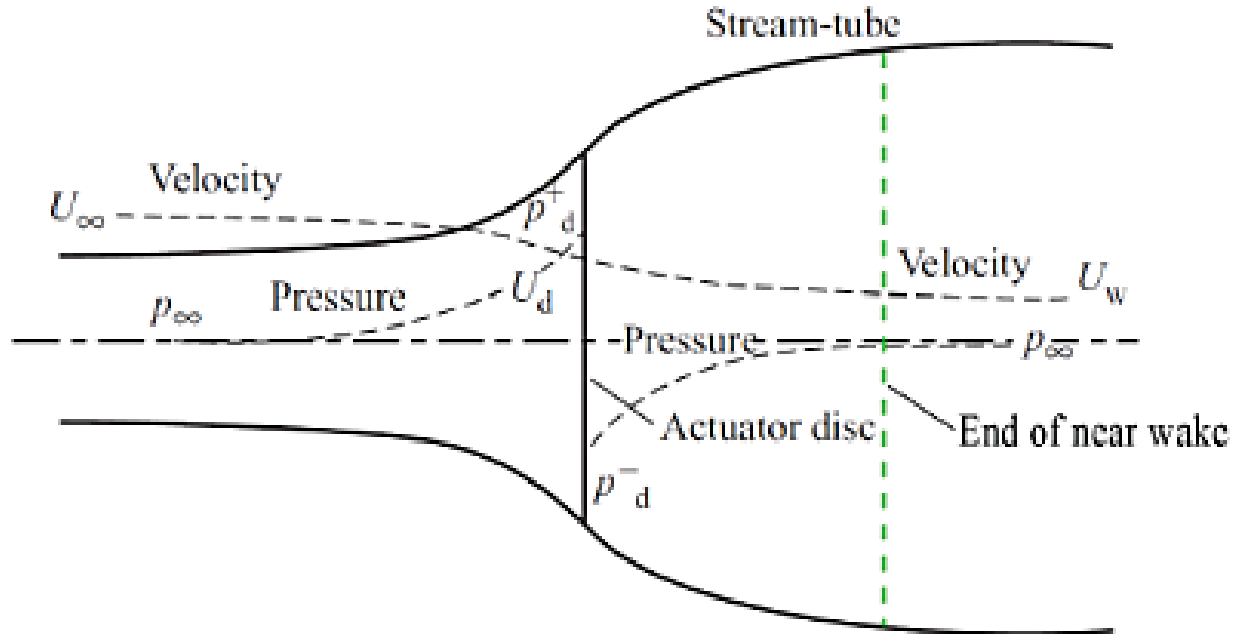


Fig. 4.56: Near-wake region.

Wake meandering is the large-scale movement of the wake deficit transported by large turbulent eddies. This wake-meandering process is treated pragmatically in DWM ([ff-Leal08]) by modeling the meandering as a passive tracer, which transfers the wake deficit transversely (horizontally and vertically) to a moving frame of reference (MFOR) – as illustrated in Fig. 4.45 – based on the ambient wind (including large turbulent eddies) spatially averaged across planes of the wake.

Wake-added turbulence is the additional small-scale turbulence generated from the turbulent mixing in the wake. It is often modeled in DWM by scaling up the background (undisturbed) turbulence.

Several variations of DWM have been implemented, e.g., by the Technical University of Denmark ([ff-Meal10, ff-Meal16]) and the University of Massachusetts ([ff-Ceal15, ff-Hao16, ff-Heal14]). Although the exact limitations of existing DWM implementations depend on the implementation, specific limitations that are addressed in developing FAST.Farm are summarized in Table 4.19 and are discussed where appropriate in the next section.

Table 4.19: Dynamic Wake Meandering Limitations Addressed by FAST.Farm

Limitation	Solution/Innovation
<ul style="list-style-type: none"> Ambient wind is solved per individual rotor and generated synthetically based on the Taylor’s frozen-turbulence assumption; not coherent across the wind farm or based on mesoscale conditions or local terrain. 	<ul style="list-style-type: none"> Optionally compute ambient wind-farm-wide from a high-fidelity precursor.
<ul style="list-style-type: none"> No treatment of a wind farm super controller. 	<ul style="list-style-type: none"> Optional inclusion of a wind farm super controller.
<ul style="list-style-type: none"> Wake advects at mean ambient wind speed, not accelerating from near wake to far wake or affected by local flow conditions. 	<ul style="list-style-type: none"> Wake advects based on the local spatially averaged ambient wind speed and wake deficit.
<ul style="list-style-type: none"> Wake deficit is not distorted by inflow skew (i.e., when looking downwind, the wake looks circular, not elliptical). Wake centerline is not deflected by inflow skew. 	<ul style="list-style-type: none"> Wake deficit solved in planes parallel to rotor disk. Wake centerline deflected based on inflow skew.
<ul style="list-style-type: none"> Wake deficit and centerline based only on mean conditions, not updated for transients in inflow, turbine control, or wind turbine motion (the latter is especially important for floating offshore wind turbines). 	<ul style="list-style-type: none"> Wake deficit and centerline updated based on low-pass-filtered inflow, wind turbine control, and wind turbine motion.
<ul style="list-style-type: none"> Individual wind turbine and wake dynamics solved individually or serially, not considering two-way wake-merging interactions. Wake impingement based only on the strongest wake deficit – not considering cumulative effects from multiple upwind wind turbines – and/or the wake impingement approach is treated differently below and above rated wind speed (i.e., a discrete change). No available method to calculate disturbed wind in zones of wake overlap. 	<ul style="list-style-type: none"> Individual wind turbine and wake dynamics solved in parallel on multiple cores. Wake merging allowed to influence wake dynamics. Wake deficits of downwind wind turbines dependent on impingement of wakes from upwind wind turbines. Wake deficits superimposed in the axial direction based on the RSS method.
<ul style="list-style-type: none"> Wake meandering velocity calculated with uniform spatial averaging, resulting in less meandering than expected and at improper frequencies. The wakes meander laterally, but not axially. 	<ul style="list-style-type: none"> Wake meandering velocity calculated with optional weighted spatial averaging based on the jinc function to result in closer-to-ideal low-pass filtering. Wakes meander both laterally and axially.

FAST.Farm Theory Basis

FAST.Farm is a nonlinear time-domain multiphysics engineering tool composed of multiple submodels, each representing different physics domains of the wind farm. FAST.Farm is implemented as open-source software that follows the programming requirements of the FAST modularization framework ([ff-Jon13]), whereby the submodels are implemented as modules interconnected through a driver code. The submodel hierarchy of FAST.Farm is illustrated in Fig. 4.46. Wake advection, deflection, and meandering; near-wake correction; and wake-deficit increment are submodels of the wake-dynamics (*WD*) model, implemented in a single module. Ambient wind and wake merging are submodels of the ambient wind and array effects (*AWAE*) model, implemented in a single module. Combined with the super controller (*SC*) and OpenFAST (*OF*) modules, FAST.Farm has four modules and one driver. There are multiple instances of the *OF* and *WD* modules – one instance for each wind turbine/rotor. Each submodel/module is described in the subsections below.

FAST.Farm can be compiled and run in serial or parallel mode. Parallelization has been implemented in FAST.Farm through OpenMP, which allows FAST.Farm to take advantage of multicore computers by dividing computational tasks among the cores/threads within a node (but not between nodes) to speed up a single simulation. This process is illustrated in Fig. 4.57 for a node where the number of threads (N_{Th}) is greater than the number of wind turbines (N_t). There is one instance of the *AWAE* and *SC* modules and N_t instances of the *OF* and *WD* modules. The initialization, update states, calculate output, and end calls to each module are shown. The output calculation of *AWAE* is parallelized across all threads. During time marching, each instance of *OF* is solved in parallel while the ambient wind data are read by *AWAE*.

The size of the wind farm and number of wind turbines is limited only by the available RAM. In parallel mode, each instance of the OpenFAST submodel can be run in parallel on separate threads. At the same time, the ambient wind within the *AWAE* module is being read into memory on another thread. Thus, the fastest simulations require at least one more core than the number of wind turbines in the wind farm. Furthermore, the output calculations within the *AWAE* module are parallelized into separate threads. To support the modeling of large wind farms, single simulations involving memory parallelization and parallelization between nodes of a multinode HPC through MPI is likely required. MPI has not yet been implemented within FAST.Farm. However, a multinode HPC can be used to run multiple serial or parallelized simulations in parallel (in batch mode) on separate nodes. In serial mode, multiple serial simulations can be run in parallel (in batch mode) on separate cores and/or nodes.

FAST.Farm Driver

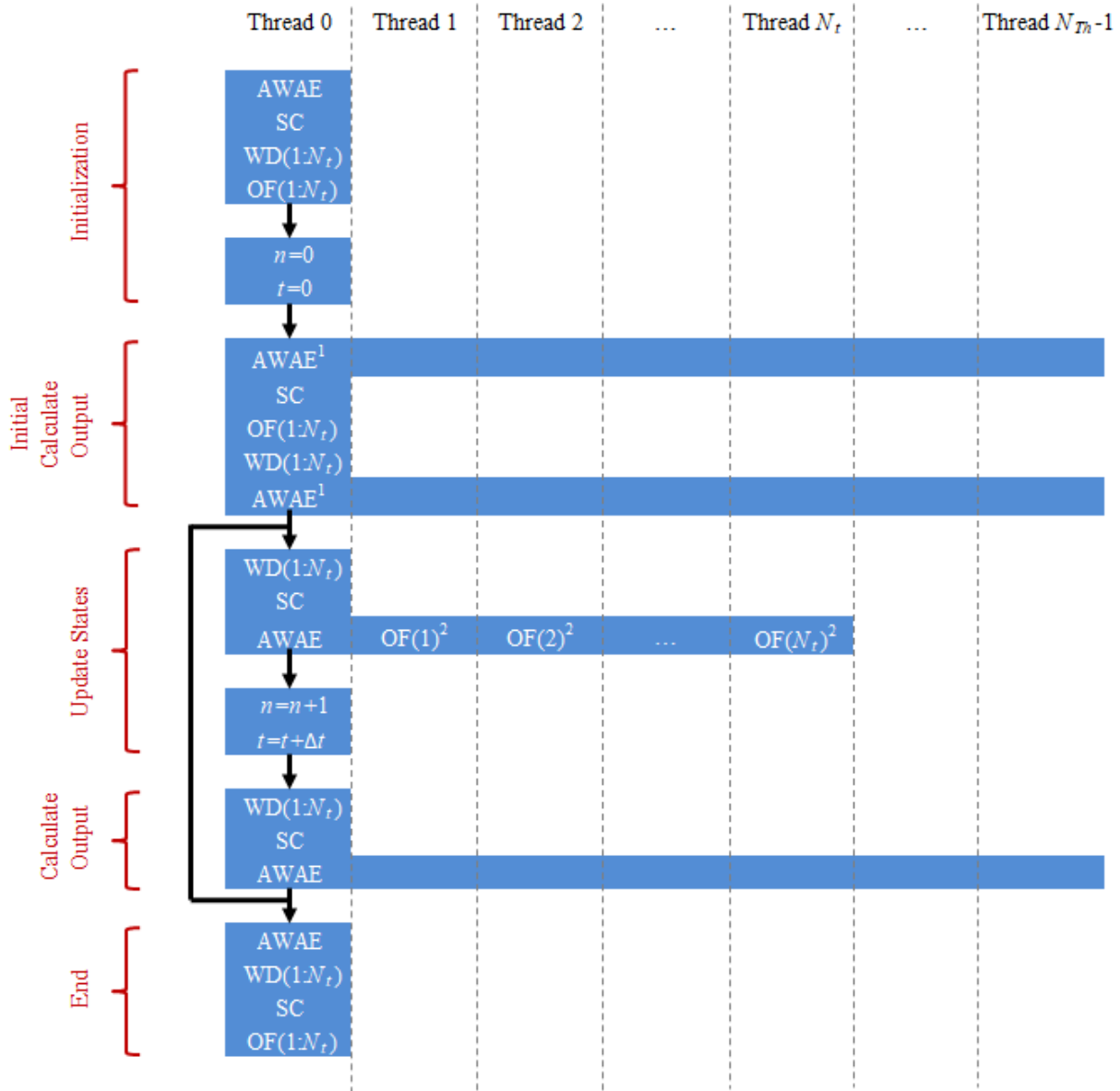
The FAST.Farm driver, also known as the “glue code,” is the code that couples individual modules together and drives the overall time-domain solution forward. Additionally, the FAST.Farm driver reads an input file of simulation parameters, checks the validity of these parameters, initializes the modules, writes results to a file, and releases memory at the end of the simulation.

To simplify the coupling algorithm in the FAST.Farm driver and ensure computational efficiency, all module states (x^d), inputs (u^d), outputs (y^d), and functions (X^d for state updates and Y^d for outputs) in FAST.Farm are expressed in discrete time, $t = n\Delta t$, where t is time, n is the discrete-time-step counter, and Δt is the user-specified discrete time step (increment). Thus, the most general form of a module in FAST.Farm is simpler than that permitted by the FAST modularization framework ([ff-Jon13]), represented mathematically as:¹

$$\begin{aligned} x^d[n+1] &= X^d(x^d[n], u^d[n], n) \\ y^d[n] &= Y^d(x^d[n], u^d[n], n) \end{aligned}$$

The *SC*, *OF*, and *WD* modules do not have direct feedthrough of input to output, meaning that the corresponding output functions simplify to $y^d[n] = Y^d(x^d[n], n)$. The ability of the *OF* module to be written in the above form is explained in Section 4.2.14. Additionally, the *AWAE* module does not have states, reducing the module to a feed-forward-only system and a module form that simplifies to $y^d[n] = Y^d(u^d[n], n)$. For functions in this manual, square brackets

¹ x^d and X^d are identical to what is described in [ff-Jon13]. u^d , y^d , and Y^d are identical to u , y , and Y from [ff-Jon13], but are only evaluated in discrete time, $t = n\Delta t$, and so, are marked here with superscript d .



¹The output calculation of AWAE is called twice at initialization--first to return the high-resolution ambient wind before initializing OF and second to fully initialize the output

²In OF, state update and output calculations are combined into a single routine

Fig. 4.57: FAST.Farm parallelization process.

[] denote discrete functions and round parentheses () denote continuous functions; the brackets/parentheses are dropped when implied. The states, inputs, and outputs of each of the FAST.Farm modules (*SC*, *OF*, *WD*, and *AWAE*) are listed in Table 4.20 and explained further in the sections below.

Table 4.20: Module States, Inputs, and Outputs in FAST.Farm

Module	States (Discrete Time)	Inputs	Outputs
<i>Super Controller (SC)</i>	<ul style="list-style-type: none"> User-defined 	<ul style="list-style-type: none"> Global measurements Commands/measurements from individual turbine controllers 	<ul style="list-style-type: none"> Global controller commands Commands to individual turbine controllers
<i>OpenFAST (OF)</i>	<ul style="list-style-type: none"> None in the OpenFAST wrapper, but there are many states internal to OpenFAST 	<ul style="list-style-type: none"> Global controller commands Commands to the individual turbine controller \vec{V}_{Dist}^{High} 	<ul style="list-style-type: none"> Commands/measurements from the individual turbine controller \hat{x}^{Disk} \vec{p}^{Hub} D^{Rotor} γ^{YawErr} DiskAvg V_x^{Rel} AzimAvg $C_t(r)$
<i>Wake Dynamics (WD)</i>	<ul style="list-style-type: none"> FiltDiskAvg V_x^{Rel} FiltAzimAvg $C_t(r)$ <p>For $0 \leq n_p \leq N_p - 1$:</p> <ul style="list-style-type: none"> Filt $D_{n_p}^{Rotor}$ Filt $\gamma_{n_p}^{YawErr}$ Filt $\vec{V}_{n_p}^{Plane}$ FiltDiskAvg $V_{x_{n_p}}^{Wind}$ Filt $TI_{Amb_{n_p}}$ $x_{n_p}^{Plane}$ $\hat{x}_{n_p}^{Plane}$ $\vec{p}_{n_p}^{Plane}$ $V_{x_{n_p}}^{Wake}(r)$ $V_{r_{n_p}}^{Wake}(r)$ 	<ul style="list-style-type: none"> \hat{x}^{Disk} \vec{p}^{Hub} D^{Rotor} γ^{YawErr} DiskAvg V_x^{Rel} AzimAvg $C_t(r)$ $\vec{V}_{n_p}^{Plane}$ for $0_p \leq N_p - 1$ DiskAvg V_x^{Wind} TI_{Amb} 	<p>For $0 \leq n_p \leq N_p - 1$:</p> <ul style="list-style-type: none"> $\hat{x}_{n_p}^{Plane}$ $\vec{p}_{n_p}^{Plane}$ $V_{x_{n_p}}^{Wake}(r)$ $V_{r_{n_p}}^{Wake}(r)$ $D_{n_p}^{Wake}$
<i>Ambient Wind and Array Effects (AWAE)</i>	<ul style="list-style-type: none"> None 	<p>For each turbine and $0 \leq n_p \leq N_p - 1$:</p> <ul style="list-style-type: none"> $\hat{x}_{n_p}^{Plane}$ $\vec{p}_{n_p}^{Plane}$ $V_{x_{n_p}}^{Wake}(r)$ $V_{r_{n_p}}^{Wake}(r)$ $D_{n_p}^{Wake}$ 	<p>For each turbine:</p> <ul style="list-style-type: none"> \vec{V}_{Dist}^{High} $\vec{V}_{n_p}^{Plane}$ for $0 \leq n_p \leq N_p - 1$ DiskAvg V_x^{Wind} TI_{Amb}

After initialization and within each time step, the states of each module (*SC*, *OF*, and *WD*) are updated (from time t to time $t + \Delta t$, or equivalently, n to $n + 1$); time is incremented; and the module outputs are calculated and transferred

as inputs to other modules. Because of the form simplifications, the state updates of each module can be solved in parallel; the output-to-input transfer does not require a large nonlinear solve; and overall correction steps of the solution are not needed. The lack of a correction step is a major simplification of the coupling algorithm used within OpenFAST ([ff-Seal14, ff-Seal15]). Furthermore, the output calculations of the *SC*, *OF*, and *WD* modules can be parallelized, followed then by the output calculation of the *AWAE* module.² In parallel mode, parallelization has been implemented in FAST.Farm through OpenMP.

Because of the small timescales and sophisticated physics, the OpenFAST submodel is the computationally slowest of the FAST.Farm modules. Additionally, the output calculation of the *AWAE* module is the only major calculation that cannot be solved in parallel to OpenFAST. Because of this, the parallelized FAST.Farm solution at its fastest may execute only slightly more slowly than stand-alone OpenFAST simulations. This results in simulations that are computationally inexpensive enough to run the many simulations necessary for wind turbine/farm design and analysis.

Super Controller (SC Module)

Wind-farm-wide super controllers have the potential to achieve the global benefit of improving overall power performance and reducing turbine loads, based on modifying wake deficits through variations in blade pitch or generator torque and/or redirecting (steering) wakes through variations in nacelle yaw or tilt, as illustrated in Fig. 4.58.

The *SC* module of FAST.Farm provides an interface to the super controller dynamic library – essentially identical to the super controller available in SOWFA – which allows the user of FAST.Farm to implement their own wind-farm-wide control logic in discrete time and without direct feedthrough of input to output – perhaps developed through the application of FLORIS. The inputs to the *SC* module are commands or measurements from individual turbine controllers (output from the *OF* module).³ The outputs of the *SC* module are the global controller commands and individual turbine controller commands (inputs to the *OF* module).

Note that at time zero, the *SC* module is called before the call to the *OF* module and the associated individual turbine controllers. So, the initial outputs from the super controller are sent as inputs to the individual turbine controllers, but the initial inputs to the super controller from the individual turbine controller outputs at time zero are always zero. At subsequent time steps, the *OF* module and the associated individual turbine controllers are called before the output calculation of the *SC* module. As a result, at each time step other than time zero, the outputs from the super controller are extrapolated in time based on past values within *OF* before being sent as input to the individual turbine controllers. Thus, care should be taken to ensure that the outputs from the super controller vary smoothly over time (without steps). See Fig. 4.57 for more information.

OpenFAST (OF Module)

FAST.Farm makes use of OpenFAST to model the dynamics (loads and motions) of distinct turbines in the wind farm. OpenFAST captures the environmental excitations (wind inflow; for offshore systems, waves, current, and ice) and coupled system response of the full system (the rotor, drivetrain, nacelle, tower, controller; for offshore systems, the substructure and station-keeping system). OpenFAST itself is an interconnection of various modules, each corresponding to different physical domains of the coupled aero-hydro-servo-elastic solution. The details of the OpenFAST solution are outside the scope of this document, but can be found in the hyperlink above and associated references.

The *OF* module of FAST.Farm is a wrapper that enables the coupling of OpenFAST to FAST.Farm – similar to the OpenFAST wrapper available in SOWFA, but with different inputs and outputs (described below). This wrapper also controls subcycling of the OpenFAST state updates. The timescales solved within OpenFAST are much smaller than those within FAST.Farm. Therefore, for accuracy and numerical stability reasons, the OpenFAST time step is typically much smaller than that required of FAST.Farm, as depicted in Fig. 4.59.

² Not all of these possible parallel tasks have been implemented within FAST.Farm because profiling did not show adequate computational speedup. However, to minimize the computational expense of the output calculation of the *AWAE* module, the ambient wind data files are read in parallel to the state updates of the *SC*, *OF*, and *WD* modules. See the introduction to Section 4.2.14 for more information.

³ The *SC* module also has as input a placeholder for future global (e.g., wind) measurements (output from the *AWAE* module) in addition to commands or measurements from the individual turbine controllers. But the global inputs are currently null.

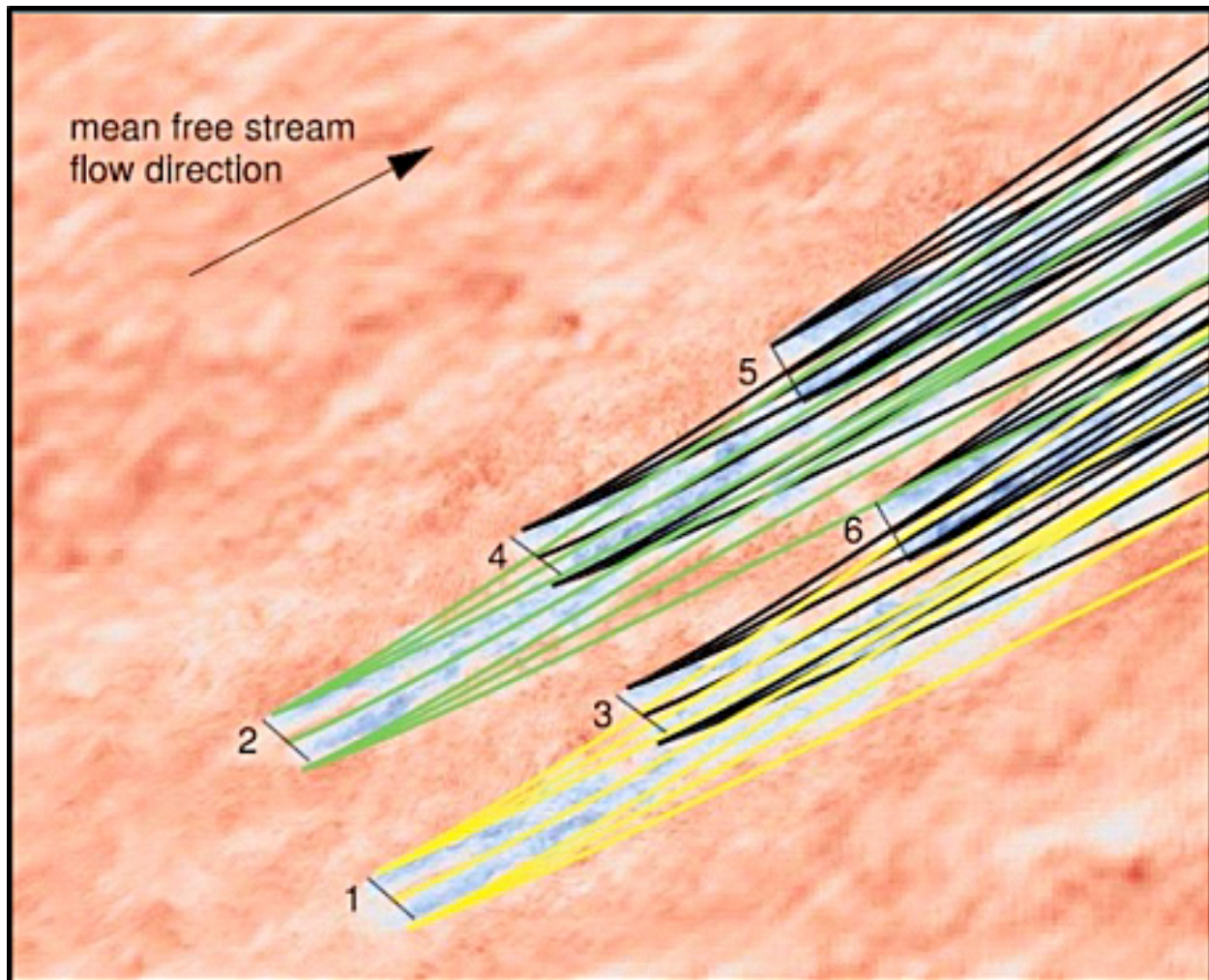


Fig. 4.58: Nacelle-yaw control used to redirect wakes away from downwind wind turbines. [ff-Geal16]

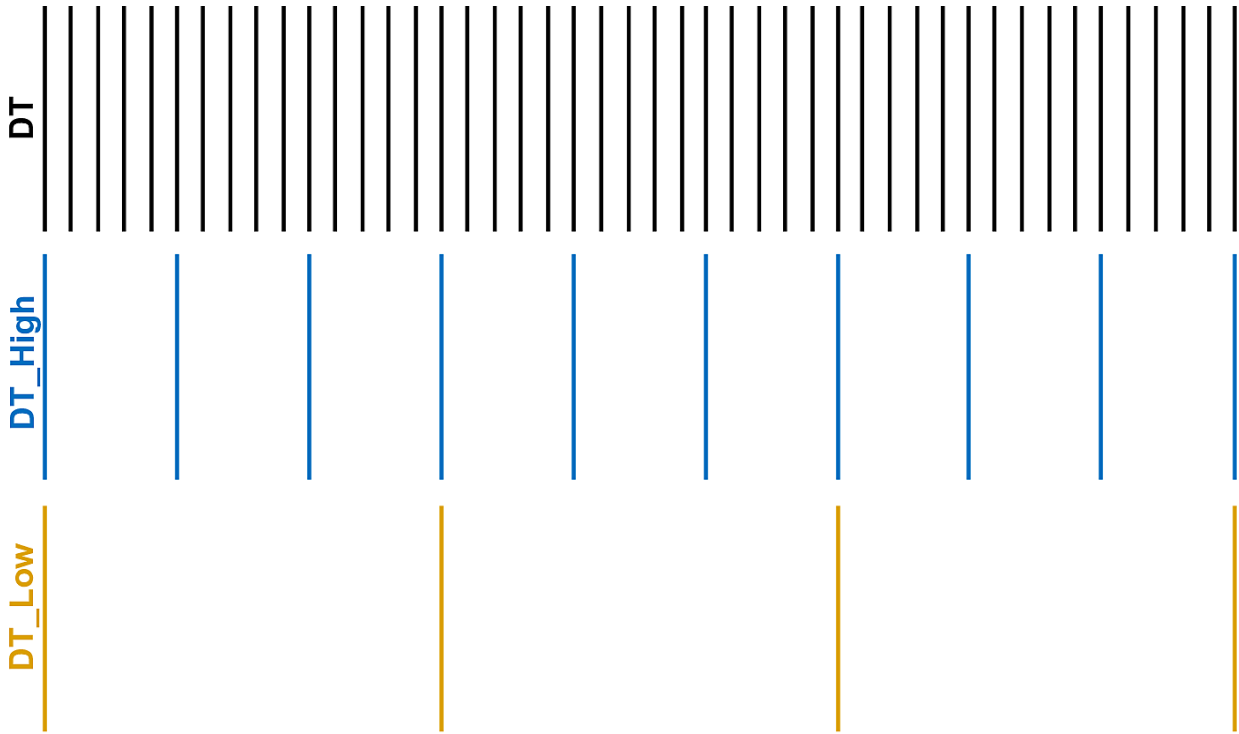


Fig. 4.59: Illustration of timescale ranges for OpenFAST (DT), the FAST.Farm high-resolution domain (DT_High), and the FAST.Farm low-resolution domain (DT_Low).

There is one instance of the *OF* module for each wind turbine. In parallel mode, these instances are parallelized through OpenMP. OpenFAST itself has various modules with different inputs, outputs, states, and parameters – including continuous-time, discrete-time, algebraic, and other (e.g., logical) states. However, for the purposes of coupling OpenFAST to FAST.Farm, the *OF* module functions in discrete time and without direct feedthrough of input to output. This is achieved by calling the *OF* module at the rate dictated by the FAST.Farm time step, Δt , and by introducing a one-time-step (Δt) delay of the output relative to the input; this one-time-step delay is not expected to be problematic because of the slow timescales solved within FAST.Farm.

At initialization, the number of wind turbines (N_t , with n_t the turbine counter such that $1 \leq n_t \leq N_t$), the corresponding OpenFAST primary input files, and turbine origins in the global *X-Y-Z* inertial-frame coordinate system are specified by the user. Turbine origins are defined as the intersection of the undeflected tower centerline and the ground or, for offshore systems, mean sea level. The global inertial-frame coordinate system is defined with *Z* directed vertically upward (opposite gravity), *X* directed horizontally nominally downwind (along the zero-degree wind direction), and *Y* directed horizontally transversely.

The global and turbine-dependent commands from the super controller (outputs from the *SC* module) are used as inputs to the *OF* module to enable the individual turbine controller to be guided by wind farm-level effects; likewise, the turbine-dependent commands or measurements are output from the *OF* module for access by the super controller (inputs to the *SC* module).

The *OF* module also uses the disturbed wind (ambient plus wakes of neighboring turbines) across a high-resolution wind domain (in both time and space) around the turbine (output from the *AWAE* module – see [Section 4.2.14](#) for more information), $\vec{V}_{\text{Dist}}^{\text{High}}$, as input, to ensure that the individual turbine loads and response calculated by OpenFAST are accurately driven by flow through the wind farm, including wake and array effects. Spatially, the high-resolution wind domain must be large enough to encompass yawing of the rotor, blade deflection, and motion of the support structure (the latter is especially important for floating offshore wind turbines). OpenFAST uses a four-dimensional (three space dimensions plus one time dimension) interpolation to determine the wind local to its analysis nodes.

The *OF* module computes several outputs needed for calculating wake dynamics (inputs to the *WD* module). These include:

- \hat{x}^{Disk} – the orientation of the rotor centerline
- \vec{p}^{Hub} – the global position of the rotor center
- D^{Rotor} – the rotor diameter
- γ^{YawErr} – the nacelle-yaw error of the rotor
- $\text{DiskAvg } V_x^{\text{Rel}}$ – the rotor-disk-averaged relative wind speed (ambient plus wakes of neighboring turbines plus turbine motion), normal to the disk
- $\text{AzimAvg } C_t(r)$ – the azimuthally averaged thrust-force coefficient (normal to the rotor disk), distributed radially, where r is the radius.

In this manual, an over arrow ($\vec{}$) denotes a three-component vector and a hat ($\hat{}$) denotes a three-component unit vector. For clarity in this manual, (r) is used to denote radial dependence as a continuous function, even though the radial dependence is stored/computed on a discrete radial finite-difference grid within FAST.Farm. Except for γ^{YawErr} and $\text{AzimAvg } C_t(r)$, all of the listed variables were computed within OpenFAST before the development of FAST.Farm. γ^{YawErr} is defined as the angle about global Z from the rotor centerline to the rotor-disk-averaged relative wind velocity (ambient plus wakes of neighboring turbines plus turbine motion), both projected onto the horizontal global X-Y plane – see Fig. 4.47 for an illustration. $\text{AzimAvg } C_t(r)$ is computed by Equation (4.204)

$$\text{AzimAvg } C_t(r) = \frac{\sum_{n_b=1}^{N_b} \{ \hat{x}^{\text{Disk}} \}^T \vec{f}_{n_b}(r)}{\frac{1}{2} \rho 2\pi r (\text{DiskAvg } V_x^{\text{Rel}})^2} \quad (4.204)$$

where:

- N_b – number of rotor blades, with n_b as the blade counter such that $1 \leq n_b \leq N_b$
- $\{ \}^T$ – vector transpose
- ρ – air density
- $\vec{f}_{n_b}(r)$ – aerodynamic applied loads⁴ distributed per unit length along a line extending radially outward in the plane of the rotor disk for blade n_b .

The numerator of Equation (4.204) is the aerodynamic applied loads distributed per unit length projected normal to the rotor disk, i.e., the radially dependent thrust force. The denominator is the normalizing factor for the radially dependent thrust coefficient, composed of the circumference at the given radius, $2\pi r$, and the dynamic pressure of the rotor-disk-averaged relative wind speed, $\frac{1}{2} \rho (\text{DiskAvg } V_x^{\text{Rel}})^2$.

Wake Dynamics (WD Module)

The *WD* module of FAST.Farm calculates wake dynamics for an individual rotor, including wake advection, deflection, and meandering; a near-wake correction; and a wake-deficit increment. The near-wake correction treats the near-wake (pressure-gradient zone) expansion of the wake deficit. The wake-deficit increment shifts the quasi-steady-state axisymmetric wake deficit nominally downwind. Each submodel is described in the subsections below. There is one instance of the *WD* module for each rotor.

The wake-dynamics calculations involve many user-specified parameters that may depend, e.g., on turbine operation or atmospheric conditions that can be calibrated to better match experimental data or HFM, e.g., by running [SOWFA](#) (or equivalent) as a benchmark. Default values have been derived for each calibrated parameter based on [SOWFA](#) simulations ([\[ff-Deal18\]](#)), but these can be overwritten by the user of FAST.Farm.

⁴ Derived using the Line2-to-Line2 mesh-mapping algorithm of FAST ([\[ff-Seal14, ff-Seal15\]](#)) to transfer the aerodynamic applied loads distributed per unit length along the deflected/curved blade as calculated within FAST.

The wake-deficit evolution is solved in discrete time on an axisymmetric finite-difference grid consisting of a fixed number of wake planes, N_p (with n_p the wake-plane counter such that $0 \leq n_p \leq N_p - 1$), each with a fixed radial grid of nodes. Because the wake deficit is assumed to be axisymmetric, the radial finite-difference grid can be considered a plane. A wake plane can be thought of as a cross section of the wake wherein the wake deficit is calculated.

Inputs to the *WD* module include \hat{x}^{Disk} , \vec{p}^{Hub} , D^{Rotor} , γ^{YawErr} , $\text{DiskAvg } V_x^{\text{Rel}}$, and $\text{AzimAvg } C_t(r)$. Additional inputs are the advection, deflection, and meandering velocity of the wake planes for the rotor ($\vec{V}_{n_p}^{\text{Plane}}$); the rotor-disk-averaged ambient wind speed, normal to the disk ($\text{DiskAvg } V_x^{\text{Wind}}$); and the ambient turbulence intensity of the wind at the rotor (TI_{Amb}) (output from the *AWAE* module – see [Section 4.2.14](#) for more information). $\vec{V}_{n_p}^{\text{Plane}}$ is computed for $0 \leq n_p \leq N_p - 1$ by spatial averaging of the disturbed wind.

The *WD* module computes several outputs needed for the calculation of disturbed wind, to be used as input to the *AWAE* module. These outputs include:

- $\hat{x}_{n_p}^{\text{Plane}}$ – the orientations of the wake planes defined using the unit vectors normal to each plane, i.e., the orientation of the wake-plane centerline
- $\vec{p}_{n_p}^{\text{Plane}}$ – the global positions of the centers of the wake planes
- $V_{x_{n_p}}^{\text{Wake}}(r)$ and $V_{r_{n_p}}^{\text{Wake}}(r)$ – the axial and radial wake-velocity deficits, respectively, at the wake planes, distributed radially
- $D_{n_p}^{\text{Wake}}$ – the wake diameters at the wake planes, each for $0 \leq n_p \leq N_p - 1$.

Though the details are left out of this manual, during start-up – whereby a wake has not yet propagated through all of the wake planes – the number of wake planes is limited by the elapsed time to avoid having to set inputs, outputs, and states in the *WD* and *AWAE* modules beyond where the wake has propagated.

Wake Advection, Deflection, and Meandering

By simple extensions to the passive tracer solution for transverse (horizontal and vertical) wake meandering, the wake-dynamics solution in FAST.Farm is extended to account for wake deflection – as illustrated in [Fig. 4.47](#) – and wake advection – as illustrated in [Fig. 4.48](#) – among other physical improvements. The following extensions are introduced:

1. Calculating the wake plane velocities, $\vec{V}_{n_p}^{\text{Plane}}$ for $0 \leq n_p \leq N_p - 1$, by spatially averaging the disturbed wind instead of the ambient wind (see [Section 4.2.14](#))
2. Orientating the wake planes with the rotor centerline instead of the wind direction
3. Low-pass filtering the local conditions at the rotor, as input to the *WD* module, to account for transients in inflow, turbine control, and/or turbine motion instead of considering time-averaged conditions.

With these extensions, the passive tracer solution enables:

1. The wake centerline to deflect based on inflow skew. This is achieved because in skewed inflow, the wake deficit normal to the disk introduces a velocity component that is not parallel to the ambient flow.
2. The wake to accelerate from near wake to far wake because the wake deficits are stronger in the near wake and weaken downwind.
3. The wake-deficit evolution to change based on conditions at the rotor because low-pass time filtered conditions are used instead of time-averaged.
4. The wake to meander axially in addition to transversely because local axial winds are considered.
5. The wake shape to be elliptical instead of circular in skewed flow when looking downwind (the wake shape remains circular when looking down the rotor centerline).

For item 3, low-pass time filtering is important because the wake reacts slowly to changes in local conditions at the rotor and because the wake evolution is treated in a quasi-steady-state fashion. Furthermore, a correction to the wake deflection resulting from item 1 is needed to account for the physical combination of wake rotation and shear, which

is not modeled directly in the *WD* module. This is achieved through a horizontally asymmetric correction to the wake deflection from item 1 (see Fig. 4.47 for an illustration). This horizontal wake-deflection correction is a simple linear correction with slope and offset, similar to the correction implemented in the wake model of FLORIS. It is important for accurate modeling of nacelle-yaw-based wake-redirection (wake-steering) wind farm control.

Mathematically, the low-pass time filter is implemented using a recursive, single-pole filter with exponential smoothing ([ff-Smi06]). The discrete-time recursion (difference) equation for this filter is ([ff-Jeal09]):

$$x_{n_p}^d [n + 1] = x_{n_p}^d [n] \alpha + u^d [n] (1 - \alpha) \quad \text{for } n_p = 0 \quad (4.205)$$

where

- x^d – discrete-time state storing the low-pass time-filtered value of input u^d
- $\alpha = e^{-2\pi\Delta t f_c}$ – low-pass time-filter parameter, with a value between 0 (minimum filtering) and 1 (maximum filtering) (exclusive)
- f_c – user-specified cutoff (corner) frequency (the time constant of the low-pass time filter is $\frac{1}{f_c}$).

Subscript n_p is used to denote the state associated with wake-plane n_p ; Equation (4.205) applies at the rotor disk, where $n_p = 0$.

To be consistent with the quasi-steady-state treatment of the wake-deficit evolution (see Section 4.2.14), the conditions at the rotor are maintained as fixed states of a wake plane as the plane propagates downstream

$$x_{n_p}^d [n + 1] = x_{n_p-1}^d [n] \quad \text{for } 1 \leq n_p \leq N_p - 1 \quad (4.206)$$

Equations (4.205) and (4.206) apply directly to the *WD* module inputs $D_{n_p}^{\text{Rotor5}}$, $\gamma_{n_p}^{\text{YawErr}}$, $\text{DiskAvg } V_x^{\text{Rel}}$, and TI_{Amb} . The associated states are $\text{Filt } D_{n_p}^{\text{Rotor}}$, $\text{Filt } \gamma_{n_p}^{\text{YawErr}}$, $\text{FiltDiskAvg } V_{x_{n_p}}^{\text{Wind}}$, and $\text{Filt } TI_{\text{Amb}_{n_p}}$ respectively (each for $0 \leq n_p \leq N_p - 1$). The *WD* module inputs $\text{DiskAvg } V_x^{\text{Rel}}$ and $\text{AzimAvg } C_t(r)$ are needed for the boundary condition at the rotor, but are not otherwise needed in the wake-deficit evolution calculation and are therefore not propagated downstream with the wake planes. Therefore, Equation (4.205) applies to these inputs but Equation (4.206) does not. The associated states are $\text{FiltDiskAvg } V_x^{\text{Rel}}$ and $\text{FiltAzimAvg } C_t(r)$. Likewise, only Equation (4.205) is used to low-pass time filter the *WD* module input $\vec{V}_{n_p}^{\text{Plane}}$ with state $\text{Filt } \vec{V}_{n_p}^{\text{Plane}}$ (for $0 \leq n_p \leq N_p - 1$). Equations (4.205) and (4.206) apply in a modified form to the *WD* module inputs \hat{x}^{Disk} and \vec{p}^{Hub} to derive the state associated with the downwind distance from the rotor to each wake plane in the axisymmetric coordinate system ($x_{n_p}^{\text{Plane}}$), and the states and outputs associated with the orientations of the wake planes, normal to the planes, ($\hat{x}_{n_p}^{\text{Plane}}$), and the global center positions of the wake planes, ($\vec{p}_{n_p}^{\text{Plane}}$) as follows:

$$\hat{x}_{n_p}^{\text{Plane}} [n + 1] = \begin{cases} \frac{\hat{x}_{n_p}^{\text{Plane}} [n] \alpha + \hat{x}^{\text{Disk}} (1 - \alpha)}{\|\hat{x}_{n_p}^{\text{Plane}} [n] \alpha + \hat{x}^{\text{Disk}} (1 - \alpha)\|_2} & \text{for } n_p = 0 \\ \hat{x}_{n_p-1}^{\text{Plane}} [n] & \text{for } 1 \leq n_p \leq N_p - 1 \end{cases} \quad (4.207)$$

$$x_{n_p}^{\text{Plane}} [n + 1] = \begin{cases} 0 & \text{for } n_p = 0 \\ x_{n_p-1}^{\text{Plane}} [n] + |d\hat{x}_{n_p-1}| & \text{for } 1 \leq n_p \leq N_p - 1 \end{cases} \quad (4.208)$$

$$\vec{p}_{n_p}^{\text{Plane}} [n + 1] = \begin{cases} \vec{p}_{n_p}^{\text{Plane}} [n] \alpha + \{\vec{p}^{\text{Hub}} [n] + (C_{\text{HWKDF}}^{\text{O}} + C_{\text{HWKDF}}^{\text{OY}} \text{Filt } \gamma_{n_p}^{\text{YawErr}} [n + 1]) \widehat{XY}_{n_p}\} (1 - \alpha) & \text{for } n_p = 0 \\ \vec{p}_{n_p-1}^{\text{Plane}} [n] + \hat{x}_{n_p-1}^{\text{Plane}} [n] d\hat{x}_{n_p-1} + \left[I - \hat{x}_{n_p-1}^{\text{Plane}} [n] \left\{ \hat{x}_{n_p-1}^{\text{Plane}} [n] \right\}^T \right] \vec{V}_{n_p-1}^{\text{Plane}} \Delta t + \left((C_{\text{HWKDF}}^{\text{X}} + C_{\text{HWKDF}}^{\text{XY}} \text{Filt } \gamma_{n_p-1}^{\text{YawErr}} [n]) d\hat{x}_{n_p-1} \right) \widehat{XY}_{n_p-1} & \text{for } 1 \leq n_p \leq N_p - 1 \end{cases} \quad (4.209)$$

⁵ Variations in the rotor diameter, D^{Rotor} , are possible as a result of blade deflection. These variations are likely small, but this variable is treated the same as other inputs for consistency.

where:

$$d\hat{x}_{n_p-1} = \left\{ \hat{x}_{n_p-1}^{\text{Plane}} [n] \right\}^T \text{Filt} \vec{V}_{n_p-1}^{\text{Plane}} [n+1] \Delta t \quad (4.210)$$

$$\widehat{XY}_{n_p} = \left\{ \frac{\left(\left\{ \hat{x}_{n_p}^{\text{Plane}} [n+1] \right\}^T \hat{X} \right) \hat{Y} - \left(\left\{ \hat{x}_{n_p}^{\text{Plane}} [n+1] \right\}^T \hat{Y} \right) \hat{X}}{\left\| \left(\left\{ \hat{x}_{n_p}^{\text{Plane}} [n+1] \right\}^T \hat{X} \right) \hat{X} + \left(\left\{ \hat{x}_{n_p}^{\text{Plane}} [n+1] \right\}^T \hat{Y} \right) \hat{Y} \right\|_2} \right\} \quad (4.211)$$

Equation (4.207) differs from Equations (4.205) and (4.206) in that after applying Equation (4.205) to low-pass time-filter input \hat{x}^{Disk} , the state is renormalized to ensure that the vector remains unit length; Equation (4.207) ensures that the wake-plane orientation is maintained as the planes propagate nominally downwind. Equation (4.208) expresses that each wake plane propagates downwind in the axisymmetric coordinate system by a distance equal to that traveled by the low-pass time-filtered wake-plane velocity projected along the plane orientation over the time step;⁶ the initial wake plane ($n_p = 0$) is always at the rotor disk. Equation (4.209) expresses the global center positions of the wake plane following the passive tracer concept, similar to Equation (4.208), but considering the full three-component movement of the wake plane, including deflection and meandering. The last term on the right-hand side of Equation (4.209) for each wake plane is the horizontal wake-deflection correction, where:

- C_{HWkDfl}^O – user-specified parameter defining the horizontal offset at the rotor
- C_{HWkDfl}^{OY} – user-specified parameter defining the horizontal offset at the rotor scaled with nacelle-yaw error
- C_{HWkDfl}^x – user-specified parameter defining the horizontal offset scaled with downstream distance
- C_{HWkDfl}^{xY} – user-specified parameter defining the horizontal offset scaled with downstream distance and nacelle-yaw error
- \hat{X} , \hat{Y} , and \hat{Z} – unit vectors parallel to the inertial-frame coordinates X , Y and, Z respectively
- \widehat{XY}_{n_p} – three-component unit vector in the horizontal global X - Y plane orthogonal to $\hat{x}_{n_p}^{\text{Plane}} [n+1]$
- $C_{HWkDfl}^O + C_{HWkDfl}^{OY} \text{Filt} \gamma_{n_p}^{\text{YawErr}} [n+1]$ – offset at the rotor
- $C_{HWkDfl}^x + C_{HWkDfl}^{xY} \text{Filt} \gamma_{n_p}^{\text{YawErr}} [n+1]$ – slope
- $d\hat{x}_{n_p-1}$ – nominally downwind increment of the wake plane (from Equation (4.208))
- I – three-by-three identity matrix
- $\left[I - \hat{x}_{n_p-1}^{\text{Plane}} [n] \left\{ \hat{x}_{n_p-1}^{\text{Plane}} [n] \right\}^T \right]$ – used to calculate the transverse component of $V_{n_p-1}^{\text{Plane}}$ normal to $\hat{x}_{n_p-1}^{\text{Plane}} [n]$.

It is noted that the advection, deflection, and meandering velocity of the wake planes, $\vec{V}_{n_p-1}^{\text{Plane}}$, is low-pass time filtered in the axial direction, but not in the transverse direction. Low-pass time filtering in the axial direction is useful for minimizing how often wake planes get close to or pass each other while they travel axially; this filtering is not needed transversely because an appropriate transverse meandering velocity is achieved through spatially averaging the disturbed wind (see Section 4.2.14).

The consistent output equation corresponding to the low-pass time filter of Equation (4.205) is $y^d [n] = x^d [n] \alpha + u^d [n] (1 - \alpha)$, i.e., $Y^d(\cdot) = X^d(\cdot)$, or equivalently, $y^d [n] = x^d [n+1]$ ([ff-Jeal09]). However, the output is delayed by one time step (Δt) to avoid having direct feedthrough of input to output within the *WD* module, yielding $y^d [n] = x^d [n]$. This one-time-step delay is applied to all outputs of the *WD* module and is not expected to be problematic because of the slow timescales solved within FAST.Farm.

⁶ The absolute value is added because, as far as wake evolution is concerned, if a wake plane travels opposite of its original propagation direction (e.g., due to a localized wind gust), the total downwind distance traveled is used rather than the instantaneous downwind distance from the rotor.

Near-Wake Correction

The near-wake correction submodel of the *WD* module computes the axial and radial wake-velocity deficits at the rotor disk as an inlet boundary condition for the wake-deficit evolution described in [Section 4.2.14](#). To improve the accuracy of the far-wake solution, the near-wake correction accounts for the drop in wind speed and radial expansion of the wake in the pressure-gradient zone behind the rotor that is not otherwise accounted for in the solution for the wake-deficit evolution. For clarity, the equations in this section are expressed using continuous variables, but within FAST.Farm the equations are solved discretely on an axisymmetric finite-difference grid.

The near-wake correction is computed differently for low thrust conditions ($C_T < \frac{24}{25}$), momentum theory is valid, and high thrust conditions ($1.1 < C_T \leq 2$), where C_T is the rotor disk-averaged thrust coefficient, derived from the low-pass time-filtered azimuthally averaged thrust-force coefficient (normal to the rotor disk), ${}^{\text{FiltAzimAvg}}C_t(r)$, evaluated at $n + 1$. The propeller brake region occurs for very high thrust-force coefficients ($C_T \geq 2$) and is not considered. Between the low and high thrust regions, a linear blending of the two solutions, based on C_T , is implemented.

At low thrust ($C_T < \frac{24}{25}$) conditions, the axial induction at the rotor disk, distributed radially, $a(r)$, is derived from the low-pass time-filtered azimuthally averaged thrust-force coefficient (normal to the rotor disk), ${}^{\text{FiltAzimAvg}}C_t(r)$, evaluated at $n + 1$ using Equation (4.212), which follows from the momentum region of blade-element momentum (BEM) theory.

$$a(r) = \frac{1}{2} \left(1 - \sqrt{1 - \text{MIN} \left[{}^{\text{FiltAzimAvg}}C_t(r), \frac{24}{25} \right]} \right) \quad (4.212)$$

To avoid unrealistically high induction at the ends of a blade, Equation (4.212) does not directly consider hub- or tip-loss corrections, but these may be accounted for in the calculation of the applied aerodynamic loads within OpenFAST (depending on the aerodynamic options enabled within OpenFAST), which have an effect on ${}^{\text{FiltAzimAvg}}C_t(r)$. Moreover, ${}^{\text{FiltAzimAvg}}C_t(r)$ is capped at $\frac{24}{25}$ to avoid ill-conditioning of the radial wake expansion discussed next.

The states and outputs associated with the axial and radial wake-velocity deficits, distributed radially ($V_{x_{n_p}}^{\text{Wake}}(r)$ and $V_{r_{n_p}}^{\text{Wake}}(r)$), are derived at the rotor disk ($n_p = 0$) from $a(r)$ and the low-pass time-filtered rotor-disk-averaged relative wind speed (ambient plus wakes of neighboring turbines plus turbine motion), normal to the disk (${}^{\text{FiltDiskAvg}}V_x^{\text{Rel}}$), evaluated at $n + 1$ using Equations (4.213) and (4.214).

$$V_{x_{n_p}}^{\text{Wake}}(r^{\text{Plane}})|_{n_p=0} = -{}^{\text{FiltDiskAvg}}V_x^{\text{Rel}} C_{\text{NearWake}} a(r) \quad (4.213)$$

$$V_{r_{n_p}}^{\text{Wake}}(r^{\text{Plane}})|_{n_p=0} = 0 \quad (4.214)$$

where

$$r^{\text{Plane}} = \sqrt{2 \int_0^r \frac{1 - a(r')}{1 - C_{\text{NearWake}} a(r')} r' dr'}$$

In Equation (4.213):

- r^{Plane} – radial expansion of the wake associated with r
- r' – dummy variable of r
- C_{NearWake} – user-specified calibration parameter greater than unity and less than 2.5 which determines how far the wind speed drops and wake expands radially in the pressure-gradient zone before recovering in the far wake.⁷

The right-hand side of Equation (4.213) represents the axial-induced velocity at the end of the pressure-gradient zone; the negative sign appears because the axial wake deficit is in the opposite direction of the free stream axial wind – see [Section 4.2.14](#) for more information. The radial expansion of the wake in the left-hand side of Equation (4.213) results

⁷ A value of $C_{\text{NearWake}} = 2$ is expected from first principles, but can be calibrated by the user of FAST.Farm to better match the far wake to known solutions.

from the application of the conservation of mass within an incremental annulus in the pressure-gradient zone.⁸ The radial wake deficit is initialized to zero, as given in Equations (4.214). Because the near-wake correction is applied directly at the rotor disk, the solution to the wake-deficit evolution for downwind distances within the first few diameters of the rotor, i.e., in the near wake, is not expected to be accurate; as a result, modifications to FAST.Farm would be needed to accurately model closely spaced wind farms.

At high thrust ($1.1 < C_T \leq 2$) conditions, the axial wake-velocity deficit, distributed radially ($V_{x_{n_p}}^{\text{Wake}}(r)$), is derived at the rotor disk ($n_p = 0$) by a Gaussian fit to LES solutions at high thrust per Equation (4.215), as derived by [ff-MT21]. The radial wake deficit is again initialized to zero.

$$V_{x_{n_p}}^{\text{Wake}}(r)|_{n_p=0} = -\mu(C_T)^{\text{FiltDiskAvg}} V_x^{\text{Rel}} e^{-\left(\frac{r}{\sigma(C_T)^{\text{Filt}} D_{n_p}^{\text{Rotor}}|_{n_p=0}}\right)^2} \quad (4.215)$$

where

$$\mu(C_T) = \frac{0.3}{2C_T^2 - 1} + \frac{1}{5}$$

$$\sigma(C_T) = \frac{C_T}{2} + \frac{4}{25}$$

Wake-Deficit Increment

As with most DWM implementations, the *WD* module of FAST.Farm models the wake-deficit evolution via the thin shear-layer approximation of the Reynolds-averaged Navier-Stokes equations under quasi-steady-state conditions in axisymmetric coordinates, with turbulence closure captured by using an eddy-viscosity formulation ([ff-Ain88]). The thin shear-layer approximation drops the pressure term and assumes that the velocity gradients are much bigger in the radial direction than in the axial direction. With these simplifications, analytical expressions for the conservation of momentum (Equation (4.216)) and conservation of mass (continuity, Equation (4.217)) are as follows:

$$V_x \frac{\partial V_x}{\partial x} + V_r \frac{\partial V_x}{\partial r} = \frac{1}{r} \frac{\partial}{\partial r} \left(r \nu_T \frac{\partial V_x}{\partial r} \right),$$

or equivalently,

$$(4.216)$$

$$r V_x \frac{\partial V_x}{\partial x} + r V_r \frac{\partial V_x}{\partial r} = \nu_T \frac{\partial V_x}{\partial r} + r \nu_T \frac{\partial^2 V_x}{\partial r^2} + r \frac{\partial \nu_T}{\partial r} \frac{\partial V_x}{\partial r}$$

$$\frac{\partial V_x}{\partial x} + \frac{1}{r} \frac{\partial}{\partial r} (r V_r) = 0 \quad , \text{ or equivalently, } V_r + r \frac{\partial V_r}{\partial r} + r \frac{\partial V_x}{\partial x} = 0 \quad (4.217)$$

where V_x and V_r are the axial and radial velocities in the axisymmetric coordinate system, respectively, and ν_T is the eddy viscosity (all dependent on x and r). The equations on the left are written in a form common in literature. The equivalent equations on the right are written in the form implemented within FAST.Farm. For clarity, the equations in this section are first expressed using continuous variables, but within FAST.Farm the equations are solved discretely on an axisymmetric finite-difference grid consisting of a fixed number of wake planes, as summarized at the end of this section. For the continuous variables, subscript n_p , corresponding to wake plane n_p , is replaced with (x) . The subscript is altogether dropped for variables that remain constant as the wake propagates downstream, following Equation (4.206). For example, $\text{Filt} D_{n_p}^{\text{Rotor}}$, $\text{FiltDiskAvg} V_{x_{n_p}}^{\text{Wind}}$, and $\text{Filt} T I_{\text{Amb} n_p}$ are written as $\text{Filt} D^{\text{Rotor}}$, $\text{FiltDiskAvg} V_x^{\text{Wind}}$, and $\text{Filt} T I_{\text{Amb}}$, respectively.

V_x and V_r are related to the low-pass time-filtered rotor-disk-averaged ambient wind speed, normal to the disk ($\text{FiltDiskAvg} V_x^{\text{Wind}}$), and the states and outputs associated with radially distributed axial and radial wake-velocity deficits,

⁸ The incremental mass flow is given by:

$$dm = 2\pi r dr \rho^{\text{FiltDiskAvg}} V_x^{\text{Rel}} (1 - a(r)) = 2\pi r^{\text{Plane}} dr^{\text{Plane}} \rho^{\text{FiltDiskAvg}} V_x^{\text{Rel}} (1 - C_{\text{NearWake}} a(r))$$

Following from this, $r^{\text{Plane}} dr^{\text{Plane}} = \frac{1-a(r)}{1-C_{\text{NearWake}} a(r)} r dr$, which can then be integrated along the radius.

$V_x^{\text{Wake}}(x, r)$ and $V_r^{\text{Wake}}(x, r)$, respectively, by Equations (4.218) and (4.219).

$$V_x(x, r) = \text{FiltDiskAvg} V_x^{\text{Wind}} + V_x^{\text{Wake}}(x, r) \quad (4.218)$$

$$V_r(x, r) = V_r^{\text{Wake}}(x, r) \quad (4.219)$$

$V_x(x, r)$ and $V_r(x, r)$ can be thought of as the change in wind velocity in the wake relative to free stream; therefore, $V_x^{\text{Wake}}(x, r)$ usually has a negative value. Several variations of the eddy-viscosity formulation have been used in prior implementations of DWM. The eddy-viscosity formulation currently implemented within FAST.Farm is given by Equation (4.220).

$$\begin{aligned} \nu_T(x, r) = & F_{\nu\text{Amb}}(x) k_{\nu\text{Amb}} \text{Filt} T I_{\text{Amb}} \text{FiltDiskAvg} V_x^{\text{Wind}} \frac{\text{Filt} D^{\text{Rotor}}}{2} \\ & + F_{\nu\text{Shr}}(x) k_{\nu\text{Shr}} \text{MAX} \left[\left(\frac{D^{\text{Wake}}(x)}{2} \right)^2 \left| \frac{\partial V_x}{\partial r}(x, r) \right|, \frac{D^{\text{Wake}}(x)}{2} \text{MIN}_{|r} \{V_x(x, r)\} \right] \end{aligned} \quad (4.220)$$

where:

- $F_{\nu\text{Amb}}(x)$ – filter function associated with ambient turbulence
- $F_{\nu\text{Shr}}(x)$ – filter function associated with the wake shear layer
- $k_{\nu\text{Amb}}$ – user-specified calibration parameters weighting the influence of ambient turbulence on the eddy viscosity
- $k_{\nu\text{Shr}}$ – user-specified calibration parameters weighting the influence of the wake shear layer on the eddy viscosity
- $\frac{D^{\text{Wake}}(x)}{2}$ – wake half-width
- $\left| \frac{\partial V_x}{\partial r} \right|$ – absolute value of the radial gradient of the axial velocity
- $\text{MIN}_{|r}(V_x(x, r))$ – used to denote the minimum value of V_x along the radius for a given downstream distance.

Although not matching any specific eddy-viscosity formulation found in prior implementations of DWM, the chosen implementation within FAST.Farm is simple to apply and inherently tailorable, allowing the user to properly calibrate the wake evolution to known solutions. The eddy-viscosity formulation expresses the influence of the ambient turbulence (first term on the right-hand side) and wake shear layer (second term) on the turbulent stresses in the wake. The dependence of the eddy viscosity on x and r is explicitly given in Equations (4.220) to make it clear which terms depend on the downwind distance and/or radius. The first term on the right-hand side of Equations (4.220) is similar to that given by [ff-Meal10] with a characteristic length taken to be the rotor radius, $\frac{\text{Filt} D^{\text{Rotor}}}{2}$. The second term is similar to that given by [ff-Keal13], but without consideration of atmospheric shear, which is considered by the *AWAE* module in the definition of ambient turbulence – see Section 4.2.14 for more information. In this second term, the characteristic length is taken to be the wake half-width and the $\text{MAX}(\)$ operator is used to denote the maximum of the two wake shear-layer methods. The second shear-layer method is needed to avoid underpredicting the turbulent stresses from the first method at radii where the radial gradient of the axial velocity approaches zero.

The filter functions currently implemented within FAST.Farm are given by Equations (4.221) and (4.222), where $C_{\nu\text{Amb}}^{\text{DMax}}$, $C_{\nu\text{Amb}}^{\text{DMin}}$, $C_{\nu\text{Amb}}^{\text{Exp}}$, $C_{\nu\text{Amb}}^{\text{FMin}}$, $C_{\nu\text{Shr}}^{\text{DMax}}$, $C_{\nu\text{Shr}}^{\text{DMin}}$, $C_{\nu\text{Shr}}^{\text{Exp}}$, and $C_{\nu\text{Shr}}^{\text{FMin}}$ are user-specified calibration parameters for the functions associated with ambient turbulence and the wake shear layer, respectively.

$$F_{\nu\text{Amb}}(x) = \begin{cases} C_{\nu\text{Amb}}^{\text{FMin}} & \text{for } x \leq C_{\nu\text{Amb}}^{\text{DMin}} \text{Filt} D^{\text{Rotor}} \\ C_{\nu\text{Amb}}^{\text{FMin}} + (1 - C_{\nu\text{Amb}}^{\text{FMin}}) \left[\frac{\frac{x}{\text{Filt} D^{\text{Rotor}}} - C_{\nu\text{Amb}}^{\text{DMin}}}{C_{\nu\text{Amb}}^{\text{DMax}} - C_{\nu\text{Amb}}^{\text{DMin}}} \right] C_{\nu\text{Amb}}^{\text{Exp}} & \text{for } C_{\nu\text{Amb}}^{\text{DMin}} \text{Filt} D^{\text{Rotor}} < x < C_{\nu\text{Amb}}^{\text{DMax}} \text{Filt} D^{\text{Rotor}} \\ 1 & \text{for } x \geq C_{\nu\text{Amb}}^{\text{DMax}} \text{Filt} D^{\text{Rotor}} \end{cases} \quad (4.221)$$

$$F_{\nu\text{Shr}}(x) = \begin{cases} C_{\nu\text{Shr}}^{\text{FMin}} & \text{for } x \leq C_{\nu\text{Shr}}^{\text{DMin Filt}} D^{\text{Rotor}} \\ C_{\nu\text{Shr}}^{\text{FMin}} + (1 - C_{\nu\text{Shr}}^{\text{FMin}}) \left[\frac{\frac{x}{C_{\nu\text{Shr}}^{\text{DMax Filt}} D^{\text{Rotor}}} - C_{\nu\text{Shr}}^{\text{DMin}}}{C_{\nu\text{Shr}}^{\text{DMax}} - C_{\nu\text{Shr}}^{\text{DMin}}} \right] C_{\nu\text{Shr}}^{\text{Exp}} & \text{for } C_{\nu\text{Shr}}^{\text{DMin Filt}} D^{\text{Rotor}} < x < C_{\nu\text{Shr}}^{\text{DMax Filt}} D^{\text{Rotor}} \\ 1 & \text{for } x \geq C_{\nu\text{Shr}}^{\text{DMax Filt}} D^{\text{Rotor}} \end{cases} \quad (4.222)$$

The filter functions of Equations (4.221) and (4.222) represent the delay in the turbulent stress generated by ambient turbulence and the development of turbulent stresses generated by the wake shear layer, respectively, and are made general in FAST.Farm. Each filter function is split into three regions of downstream distance, including:

1. A fixed minimum value (between zero and unity, inclusive) near the rotor
2. A fixed value of unity far downstream from the rotor
3. A transition region for intermediate distances, where the value can transition linearly or via any rational exponent of the normalized downstream distance within the transition region.

The definition of wake diameter is somewhat ambiguous and not defined consistently in DWM literature. FAST.Farm allows the user to choose one of several methods to calculate the wake diameter, $D^{\text{Wake}}(x)$, including taking the wake diameter to be:

1. The rotor diameter
2. The diameter at which the axial velocity of the wake is the C_{WakeDiam} fraction of the ambient wind speed, where C_{WakeDiam} is a user-specified calibration parameter between zero and 0.99 (exclusive)
3. The diameter that captures the C_{WakeDiam} fraction of the mass flux of the axial wake deficit across the wake plane
4. The diameter that captures the C_{WakeDiam} fraction of the momentum flux of the axial wake deficit across the wake plane.

Through the use of a $\text{MAX}(\)$ operator, models 2 through 4 have a lower bound set equal to the rotor diameter when the wake-diameter calculation otherwise returns smaller values. This is done to avoid numerical problems resulting from too few wind data points in the spatial averaging used to compute the wake-meandering velocity – see [Section 4.2.14](#) for more information. Although the implementation in FAST.Farm is numerical, analytical expressions for these four methods are given in Equation (4.223). Here, $|x$ means the mean conditioned on x .

$$D^{\text{Wake}}(x) = \begin{cases} \text{Filt } D^{\text{Rotor}} & \text{for method 1-rotor diameter} \\ \text{MAX}(\text{Filt } D^{\text{Rotor}}, \{2r | (V_x(x, r) = C_{\text{WakeDiam}} \text{FiltDiskAvg } V_x^{\text{Wind}})\}) & \text{for method 2-velocity based} \\ \text{MAX}\left(\text{Filt } D^{\text{Rotor}}, \left\{D^{\text{Wake}}(x) \mid \int_0^{\frac{D^{\text{Wake}}(x)}{2}} V_x^{\text{Wake}}(x, r) 2\pi r dr = C_{\text{WakeDiam}} \int_0^\infty V_x^{\text{Wake}}(x, r) 2\pi r dr\right\}\right) & \text{for method 3-mass-flux based} \\ \text{MAX}\left(\text{Filt } D^{\text{Rotor}}, \left\{D^{\text{Wake}}(x) \mid \int_0^{\frac{D^{\text{Wake}}(x)}{2}} (V_x^{\text{Wake}}(x, r))^2 2\pi r dr = C_{\text{WakeDiam}} \int_0^\infty (V_x^{\text{Wake}}(x, r))^2 2\pi r dr\right\}\right) & \text{for method 4-momentum-flux based} \end{cases} \quad (4.223)$$

The momentum and continuity equations are solved numerically in the wake-deficit-increment submodel of the *WD* module using a second-order accurate finite-difference method at $n + \frac{1}{2}$, following the implicit Crank-Nicolson

method ([ff-CN96]). Following this method, central differences are used for all derivatives, e.g., Equation (4.224) for the momentum equation.

$$\frac{\partial V_x}{\partial x} = \frac{V_{x_{n_p}}^{\text{Wake}}(r)[n+1] - V_{x_{n_p-1}}^{\text{Wake}}(r)[n]}{\Delta x} \quad (4.224)$$

Here,

$$\Delta x = |x_{n_p}^{\text{Plane}}[n+1] - x_{n_p-1}^{\text{Plane}}[n]|$$

or equivalently from Equation (4.210)

$$\Delta x = \left| \left\{ \hat{x}_{n_p-1}^{\text{Plane}}[n] \right\}^T \text{Filt} \vec{V}_{n_p-1}^{\text{Plane}}[n+1] \Delta t \right| \quad \text{for } 1 \leq n_p \leq N_p - 1$$

For the momentum equation, for each wake plane downstream of the rotor ($1 \leq n_p \leq N_p - 1$), the terms V_x , V_r , ν_T , and $\frac{\partial \nu_T}{\partial r}$ are calculated at n (or equivalently $x = x_{n_p-1}^{\text{Plane}}[n]$), e.g., $V_x = \text{FiltDiskAvg } V_{x_{n_p-1}}^{\text{Wind}}[n] + V_{x_{n_p-1}}^{\text{Wake}}(r)[n]$ and $V_r = V_{r_{n_p-1}}^{\text{Wake}}(r)[n]$, to avoid nonlinearities in the solution for $n+1$. This will prevent the solution from achieving second-order convergence, but has been shown to remain numerically stable. Although the definition of each central difference is outside the scope of this document, the end result is that for each wake plane downstream of the rotor, $V_{x_{n_p}}^{\text{Wake}}(r)[n+1]$ can be solved via a linear tridiagonal matrix system of equations in terms of known solutions of $V_{x_{n_p-1}}^{\text{Wake}}(r)[n]$, $V_{r_{n_p-1}}^{\text{Wake}}(r)[n]$, and other previously calculated states, e.g., $\text{FiltDiskAvg } V_{x_{n_p-1}}^{\text{Wind}}[n]$. The linear tridiagonal matrix system of equations is solved efficiently in FAST.Farm via the Thomas algorithm ([ff-Tho49]).

For the continuity equation, a different finite-difference scheme is needed because the resulting tridiagonal matrix is not diagonally dominant when the same finite-difference scheme used for the momentum equation is used for the continuity equation, resulting in a numerically unstable solution. Instead, the finite-difference scheme used for the continuity equation is based on a second-order accurate scheme at $n + \frac{1}{2}$ and $n_r - \frac{1}{2}$. However, the terms involving V_r and $\frac{\partial V_r}{\partial r}$ are calculated at $n+1$, e.g., $V_r = \frac{1}{2} \left(V_{r_{n_p, n_r}}^{\text{Wake}}[n+1] + V_{r_{n_p, n_r-1}}^{\text{Wake}}[n+1] \right)$, where n_r is the radii counter for N_r radial nodes ($0 \leq n_r \leq N_r - 1$).⁹ Although the definition of each central difference is outside the scope of this document, the end result is that for each wake plane downstream of the rotor, $V_{r_{n_p, n_r}}^{\text{Wake}}[n+1]$ can be solved explicitly sequentially from known solutions of $V_{x_{n_p}}^{\text{Wake}}(r)[n+1]$ (from the solution of the momentum equation), $V_{x_{n_p-1}}^{\text{Wake}}(r)[n]$, and $V_{r_{n_p, n_r-1}}^{\text{Wake}}[n+1]$ for $1 \leq n_r \leq N_r - 1$.¹⁰

Ambient Wind and Array Effects (AWAE Module)

The AWAE module of FAST.Farm processes ambient wind and wake interactions across the wind farm, including the ambient wind and wake-merging submodels. The ambient wind submodule processes ambient wind across the wind farm from either a high-fidelity precursor simulation or an interface to the *InflowWind* module in OpenFAST. The wake-merging submodule identifies zones of overlap between all wakes across the wind farm and merges their wake deficits. Both submodels are described in the subsections below.

The calculations in the AWAE module make use of wake volumes, which are volumes formed by a (possibly curved) cylinder starting at a wake plane and extending to the next adjacent wake plane along a line connecting the centers of the two wake planes. If the adjacent wake planes (top and bottom of the cylinder) are not parallel, e.g., for transient simulations involving variations in nacelle-yaw angle, the centerline will be curved instead of straight. Fig. 4.49 illustrates some of the concepts that will be detailed in the subsections below. The calculations in the AWAE module also require looping through all wind data points, turbines, and wake planes; these loops have been sped up in the parallel mode of FAST.Farm by implementation of OpenMP parallelization.

The AWAE module does not have states, reducing the module to a feed-forward-only system whereby the module outputs are computed directly from the module inputs (with direct feedthrough of input to output). The AWAE module uses as

⁹ Subscript n_r has been used here in place of (r)

¹⁰ Note that the radial wake-velocity deficit at the centerline of the axisymmetric coordinate system ($n_r = 0$) is always zero ($V_{r_{n_p}}^{\text{Wake}}(r)|_{r=0} = 0$).

input $\hat{x}_{n_p}^{\text{Plane}}$, $\bar{p}_{n_p}^{\text{Plane}}$, $V_{x_{n_p}}^{\text{Wake}}(r)$, $V_{r_{n_p}}^{\text{Wake}}(r)$, and $D_{n_p}^{\text{Wake}}$ (each for $0 \leq n_p \leq N_p - 1$) as computed by the wake-dynamics model for each individual wind turbine (output by the *WD* module). The *AWAE* module computes output $\vec{V}_{\text{Dist}}^{\text{High}}$ needed for the calculation of OpenFAST for each individual wind turbine (input to the *OF* module) as well as outputs for $\vec{V}_{n_p}^{\text{Plane}}$ for $0 \leq n_p \leq N_p - 1$, $\text{DiskAvg } V_x^{\text{Wind}}$, and TI_{Amb} needed for the calculation of wake dynamics for each individual wind turbine (input to the *WD* module).

Ambient Wind

The ambient wind data used by FAST.Farm can be generated in one of two ways. The use of the *InflowWind* module in OpenFAST enables the use of simple ambient wind, e.g., uniform wind, discrete wind events, or synthetically generated turbulent wind data. Synthetically generated turbulence can be from, e.g., TurbSim or the Mann model, in which the wind is propagated through the wind farm using Taylor’s frozen-turbulence assumption. This method is most applicable to small wind farms or a subset of wind turbines within a larger wind farm. FAST.Farm can also use ambient wind generated by a high-fidelity precursor LES simulation of the entire wind farm (without wind turbines present), such as the ABLSolver preprocessor of SOWFA. This atmospheric precursor simulation captures more physics than synthetic turbulence – as illustrated in Fig. 4.50 – including atmospheric stability, wind-farm-wide turbulent length scales, and complex terrain effects. It is more computationally expensive than using the ambient wind modeling options of *InflowWind*, but it is much less computationally expensive than a SOWFA simulation with multiple wind turbines present.

FAST.Farm requires ambient wind to be available in two different resolutions. Because wind will be spatially averaged across wake planes within the *AWAE* module, FAST.Farm needs a low-resolution wind domain (in both space and time) throughout the wind farm. The spatial resolution of the low-resolution domain – consisting of a structured 3D grid of wind data points – should be sufficient so that the spatial averaging is accurate, e.g., on the order of tens of meters for utility-scale wind turbines. The time step of the low-resolution domain dictates that of the FAST.Farm driver (Δt) and all FAST.Farm modules. It should therefore be consistent with the timescales of wake dynamics, e.g., on the order of seconds and smaller for higher mean wind speeds. Note that OpenFAST is subcycled within the *OF* module with a smaller time step. For accurate load calculation by OpenFAST, FAST.Farm also needs high-resolution wind domains (in both space and time) around each wind turbine and encompassing any turbine displacement. The spatial and time resolution of each high-resolution domain should be sufficient for accurate aerodynamic load calculations, e.g., on the order of the blade chord length and fractions of a second ([ff-Seal19b]). The high-resolution domains overlap portions of the low-resolution domain. For simplicity of and to minimize computational expense within FAST.Farm, the time step of the high-resolution domain must be an integer divisor of the low-resolution domain time step.

When using ambient wind generated by a high-fidelity precursor simulation, the *AWAE* module reads in the three-component wind-velocity data across the high- and low-resolution domains – $\vec{V}_{\text{Amb}}^{\text{High}}$ for each turbine and $\vec{V}_{\text{Amb}}^{\text{Low}}$, respectively – that were computed by the high-fidelity solver within each time step. These values are stored in files for use in a given driver time step. The wind data files, including spatial discretizations, must be in VTK format and are specified by users of FAST.Farm at initialization. When using the *InflowWind* inflow option, the ambient wind across the high- and low-resolution domains are computed by calling the *InflowWind* module. In this case, the spatial discretizations of these domains are specified directly within the FAST.Farm primary input file. These wind data from the combined low- and high-resolution domains within a given driver time step represent the largest memory requirement of FAST.Farm.

After the ambient wind is processed at a given time step, the ambient wind submodel computes as output the rotor-disk-averaged ambient wind speed, normal to the disk, $\text{DiskAvg } V_x^{\text{Wind}}$, for each turbine using Equation (4.225).

$$\text{DiskAvg } V_x^{\text{Wind}} = \left(\left\{ \hat{x}_{n_p}^{\text{Plane}} \right\}^T \left\{ \frac{1}{N_{n_p}^{\text{Polar}}} \sum_{n^{\text{Polar}}=1}^{N_{n_p}^{\text{Polar}}} \vec{V}_{\text{Amb}, n^{\text{Polar}}}^{\text{Low}} \right\} \right) \Big|_{n_p=0} \quad (4.225)$$

In Equation (4.225), $N_{n_p}^{\text{Polar}}$ is the number of points in a polar grid on wake plane n_p of the given wind turbine, n^{Polar} is the point counter such that $1 \leq n^{\text{Polar}} \leq N_{n_p}^{\text{Polar}}$ for wake plane n_p , and the equation is evaluated for the wake plane at the rotor disk ($n_p = 0$). The polar grid on wake plane n_p has a uniform radial and azimuthal discretization equal to the average X-Y-Z spatial discretization of the low-resolution domain (independent from the radial finite-difference grid

used within the *WD* module) and a diameter of $C_{Meander} D_{n_p}^{Wake}$; $C_{Meander}$ is discussed further in [Section 4.2.14](#) below. Subscript n^{Polar} is appended to \vec{V}_{Amb}^{Low} in Equation (4.225) to identify wind data that have been trilinearly interpolated from the low-resolution domain to the polar grid on the wake plane. Intuitively, Equation (4.225) states that the rotor-disk-averaged ambient wind speed, normal to the disk, for each turbine is calculated as the uniform spatial average of the ambient wind velocity on the wake plane at the rotor disk projected along the low-pass time-filtered rotor centerline.

The ambient wind submodel of the *AWAE* module also calculates as output the ambient turbulence intensity around each rotor, TI_{Amb} , using Equation (4.226):

$$TI_{Amb} = \left(\frac{\sqrt{\frac{1}{3N_{n_p}^{Polar}} \sum_{n^{Polar}=1}^{N_{n_p}^{Polar}} \left\| \vec{V}_{Amb, n^{Polar}}^{Low} - \left\{ \frac{1}{N_{n_p}^{Polar}} \sum_{n^{Polar}=1}^{N_{n_p}^{Polar}} \vec{V}_{Amb, n^{Polar}}^{Low} \right\} \right\|_2^2}}{\left\| \left\{ \frac{1}{N_{n_p}^{Polar}} \sum_{n^{Polar}=1}^{N_{n_p}^{Polar}} \vec{V}_{Amb, n^{Polar}}^{Low} \right\} \right\|_2} \right) \bigg|_{n_p=0} \quad (4.226)$$

The bracketed term in Equation (4.226) is the same as in Equation (4.225), representing the uniform spatial average of the ambient wind velocity on the wake plane at the rotor disk. In contrast to the common definition of turbulence intensity used in the wind industry, which consists of a time-averaged quantity of the axial wind component, the turbulence intensity calculated in the ambient wind submodel of the *AWAE* module is based on a uniform spatial average of the three vector components. Not using time averaging ensures that only ambient wind at the current time step needs to be processed, which decreases memory requirements. Moreover, any time variation in the spatial average is moderated by the low-pass time filter in the *WD* module. Using spatial averaging and the three vector components allows for atmospheric shear, wind veer, and other ambient wind characteristics to influence the eddy viscosity and wake-deficit evolution in the *WD* module. The incorporation of wake-added turbulence is left for future work. Note that Equation (4.226) uses the eight wind data points from the low-resolution domain surrounding each point in the polar grid rather than interpolation. This is because calculating wind data in the polar grid on the wake plane via trilinear interpolation from the low-resolution domain would smooth out spatial variations and artificially reduce the calculated turbulence intensity.

Wake Merging

In previous implementations of DWM, the wind turbine and wake dynamics were solved individually or serially, not considering two-way wake-merging interactions. Additionally, there was no method available to calculate the disturbed wind in zones of wake overlap. Wake merging is illustrated by the *SOWFA* simulation of [Fig. 4.51](#). In *FAST.Farm*, the wake-merging submodel of the *AWAE* module identifies zones of wake overlap between all wakes across the wind farm by finding wake volumes that overlap in space. Wake deficits are superimposed in the axial direction based on the RSS method ([ff-Katic2018]); transverse components (radial wake deficits) are superimposed by vector sum. In Katic et al. ([ff-Katic2018]), the RSS method is applied to wakes with axial deficits that are uniform across the wake diameter and radial deficits are not considered. In contrast, the RSS method in *FAST.Farm* is applied locally at a given wind data point. The RSS method assumes that the local kinetic energy of the axial deficit in a merged wake equals the sum of the local energies of the axial deficits for each wake at the given wind data point. The RSS method only applies to an array of scalars, which works well for axial deficits because overlapping wakes likely have similar axial directions. This means, however, that only the magnitude of the vector is important in the superposition. A vector sum is applied to the transverse components (radial wake deficits) because any given radial direction is dependent on the azimuth angle in the axisymmetric coordinate system.

The disturbed (ambient plus wakes) wind velocities across the high- and low-resolution domains – \vec{V}_{Dist}^{High} for each

turbine and $\vec{V}_{\text{Dist}}^{\text{Low}}$, respectively – are computed using Equations (4.227) and (4.228), respectively.

$$\begin{aligned} \vec{V}_{\text{Dist}}^{\text{High}} &= \vec{V}_{\text{Amb}}^{\text{High}} \\ &- \left\{ \sqrt{\sum_{n_{\text{Wake}}=1}^{N_{\text{Wake}}} \left(\left\{ \hat{\bar{x}}^{\text{Plane}} \right\}^T \left\{ V_{x_{n_{\text{Wake}}}^{\text{Wake}}} \hat{x}_{n_{\text{Wake}}}^{\text{Plane}} + V_{r_{n_{\text{Wake}}}^{\text{Wake}}} \hat{r}_{n_{\text{Wake}}}^{\text{Plane}} \right\} \right)^2} \right\} \hat{\bar{x}}^{\text{Plane}} \quad \text{for } (n_{t_{n_{\text{Wake}}}} \neq n_t) \\ &\quad 0 \quad \text{otherwise} \end{aligned} \quad (4.227)$$

$$+ \sum_{n_{\text{Wake}}=1}^{N_{\text{Wake}}} \left\{ \begin{aligned} &\left[I - \hat{\bar{x}}^{\text{Plane}} \left\{ \hat{\bar{x}}^{\text{Plane}} \right\}^T \right] \left\{ V_{x_{n_{\text{Wake}}}^{\text{Wake}}} \hat{x}_{n_{\text{Wake}}}^{\text{Plane}} + V_{r_{n_{\text{Wake}}}^{\text{Wake}}} \hat{r}_{n_{\text{Wake}}}^{\text{Plane}} \right\} \quad \text{for } (n_{t_{n_{\text{Wake}}}} \neq n_t) \\ &\vec{0} \quad \text{otherwise} \end{aligned} \right.$$

$$\begin{aligned} \vec{V}_{\text{Dist}}^{\text{Low}} &= \vec{V}_{\text{Amb}}^{\text{Low}} \\ &- \left\{ \sqrt{\sum_{n_{\text{Wake}}=1}^{N_{\text{Wake}}} \left(\left\{ \hat{\bar{x}}^{\text{Plane}} \right\}^T \left\{ V_{x_{n_{\text{Wake}}}^{\text{Wake}}} \hat{x}_{n_{\text{Wake}}}^{\text{Plane}} + V_{r_{n_{\text{Wake}}}^{\text{Wake}}} \hat{r}_{n_{\text{Wake}}}^{\text{Plane}} \right\} \right)^2} \right\} \hat{\bar{x}}^{\text{Plane}} \\ &+ \sum_{n_{\text{Wake}}=1}^{N_{\text{Wake}}} \left[I - \hat{\bar{x}}^{\text{Plane}} \left\{ \hat{\bar{x}}^{\text{Plane}} \right\}^T \right] \left\{ V_{x_{n_{\text{Wake}}}^{\text{Wake}}} \hat{x}_{n_{\text{Wake}}}^{\text{Plane}} + V_{r_{n_{\text{Wake}}}^{\text{Wake}}} \hat{r}_{n_{\text{Wake}}}^{\text{Plane}} \right\} \end{aligned} \quad (4.228)$$

Here, $(n_{t_{n_{\text{Wake}}}} \neq n_t)$ signifies that wake n^{Wake} is not associated with the given turbine n_t . The first, second, and third terms on the right-hand side of Equations (4.227) and (4.228) represent the ambient wind velocity, the RSS superposition of the axial wake-velocity deficits, and the vector sum of the transverse wake-velocity deficits, respectively. Although many mathematical details are outside the scope of this document, the nomenclature of Equations (4.227) and (4.228) is as follows:

- N^{Wake} – number of wake volumes overlapping a given wind data point in the wind domain
- n^{Wake} – wake counter such that $1 \leq n^{\text{Wake}} \leq N^{\text{Wake}}$ which, when used as a subscript, is used to identify the specific point in a wake plane in place of (r) and subscript n_p
- $V_{x_{n_{\text{Wake}}}^{\text{Wake}}}^{\text{Wake}}$ – axial wake-velocity deficit associated with where the given wind data point lies within the specific wake volume and corresponding wake plane
- $V_{r_{n_{\text{Wake}}}^{\text{Wake}}}^{\text{Wake}}$ – radial wake-velocity deficit associated with where the given wind data point lies within the specific wake volume and corresponding wake plane
- $\hat{x}_{n_{\text{Wake}}}^{\text{Plane}}$ – axial orientation associated with where the given wind data point lies within the specific wake volume and corresponding wake plane
- $\hat{r}_{n_{\text{Wake}}}^{\text{Plane}}$ – radial unit vector associated with where the given wind data point lies within the specific wake volume and corresponding wake plane
- $\hat{\bar{x}}^{\text{Plane}}$ – weighted-average axial orientation associated with a given point in the wind spatial domain
- $\left\{ \hat{\bar{x}}^{\text{Plane}} \right\}^T$ – projects $\left\{ V_{x_{n_{\text{Wake}}}^{\text{Wake}}} \hat{x}_{n_{\text{Wake}}}^{\text{Plane}} + V_{r_{n_{\text{Wake}}}^{\text{Wake}}} \hat{r}_{n_{\text{Wake}}}^{\text{Plane}} \right\}$ along $\hat{r}_{n_{\text{Wake}}}^{\text{Plane}}$
- $\left[I - \hat{x}_{n_{\text{Wake}}}^{\text{Plane}} \left\{ \hat{\bar{x}}^{\text{Plane}} \right\}^T \right]$ – calculates the transverse component of $\left\{ V_{x_{n_{\text{Wake}}}^{\text{Wake}}} \hat{x}_{n_{\text{Wake}}}^{\text{Plane}} + V_{r_{n_{\text{Wake}}}^{\text{Wake}}} \hat{r}_{n_{\text{Wake}}}^{\text{Plane}} \right\}$ normal to $\hat{\bar{x}}^{\text{Plane}}$.

Wake volumes are found by looping through all points, turbines, and wake planes and spatially determining if the given point resides in a wake volume that has a diameter equal to the radial extent of the wake planes. Wake volume n_p (for $0 \leq n_p \leq N_p - 2$) starts at wake plane n_p and extends to wake plane $n_p + 1$. Wake volumes have a centerline determined by $\hat{p}_{n_p}^{\text{Plane}}$, $\hat{x}_{n_p}^{\text{Plane}}$, $\hat{p}_{n_p+1}^{\text{Plane}}$, and $\hat{x}_{n_p+1}^{\text{Plane}}$ – this centerline is curved if $\hat{x}_{n_p}^{\text{Plane}}$ and $\hat{x}_{n_p+1}^{\text{Plane}}$ are not parallel. The calculations of $V_{x_{n_{\text{Wake}}}^{\text{Wake}}}^{\text{Wake}}$ and $V_{r_{n_{\text{Wake}}}^{\text{Wake}}}^{\text{Wake}}$ involve bilinear interpolation of the wake deficits in the axial and radial directions. The axial interpolation is complicated when the adjacent wake planes are not parallel. The vector quantity $\left\{ V_{x_{n_{\text{Wake}}}^{\text{Wake}}} \hat{x}_{n_{\text{Wake}}}^{\text{Plane}} + V_{r_{n_{\text{Wake}}}^{\text{Wake}}} \hat{r}_{n_{\text{Wake}}}^{\text{Plane}} \right\}$ represents the total wake-velocity deficit associated with where the given wind data point lies within the specific wake

volume and corresponding wake plane. Because each wake plane may have a unique orientation, what constitutes “axial” and “radial” in the superposition at a given wind data point is determined by weighted-averaging the orientations of each wake volume overlapping that point (weighted by the magnitude of each axial wake deficit). A similar equation is used to calculate the distributed wind velocities across the high-resolution domain ($\vec{V}_{\text{Dist}}^{\text{High}}$) for each turbine, which is needed to calculate the disturbed wind inflow to a turbine. Note that for the high-resolution domain, a turbine is prevented from interacting with its own wake.

Once the distributed wind velocities across the low-resolution domain have been found, the wake merging submodel of the *AWAE* module computes as output the advection, deflection, and meandering velocity of each wake plane, $\vec{V}_{n_p}^{\text{Plane}}$ for $0 \leq n_p \leq N_p - 1$, for each turbine as the weighted spatial average of the disturbed wind velocity across the wake plane, using Equation (4.229).

$$\vec{V}_{n_p}^{\text{Plane}} = \frac{\sum_{n^{\text{Polar}}=1}^{N_{n_p}^{\text{Polar}}} w_{n^{\text{Polar}}} \vec{V}_{\text{Dist}, n^{\text{Polar}}}^{\text{Low}}}{\sum_{n^{\text{Polar}}=1}^{N_{n_p}^{\text{Polar}}} w_{n^{\text{Polar}}}} \quad (4.229)$$

The polar grid on wake plane n_p has a uniform radial and azimuthal discretization equal to the average *X-Y-Z* spatial discretization of the low-resolution domain (independent from the radial finite-difference grid used within the *WD* module) and a local diameter described below. Subscript n^{Polar} is appended to $\vec{V}_{\text{Dist}}^{\text{Low}}$ in Equation (4.229) to identify wind data that have been trilinearly interpolated from the low-resolution domain to the polar grid on the wake plane. Unlike Equation (4.225), Equation (4.229) includes a spatial weighting factor, $w_{n^{\text{Polar}}}$, dependent on the radial distance of point n^{Polar} from the center of the wake plane (discussed below). FAST.Farm will issue a warning if the center of any wake plane has left the boundaries of the low-resolution domain and set the meandering velocity of each wake plane, $\vec{V}_{n_p}^{\text{Plane}}$, to zero for any wake plane that has entirely left the boundaries of the low-resolution domain. Qualitatively, Equation (4.229) states that the advection, deflection, and meandering velocity of each wake plane for each turbine is calculated as the weighted spatial average of the disturbed wind velocity on the wake plane. Larsen et al. ([ff-Leal08]) proposed a uniform spatial average where all points within a circle of diameter $2D_{n_p}^{\text{Wake}}$ are given equal weight. However, the Fourier transform of the circular function in a polar spatial domain results in a *jinc* function in the polar wave-number domain,¹¹ implying a gentle roll-off of energy below the cutoff wave number and pockets of energy at distinct wave numbers above the cutoff wave number. Experience with FAST.Farm development has shown that this approach results in less overall wake meandering and at improper frequencies. As such, three weighted spatial averaging methods have been implemented in FAST.Farm, as defined in Equation (4.230).

$$w_{n^{\text{Polar}}} = \begin{cases} 1 & \text{for method 1-uniform} \\ jinc\left(\frac{r_{n^{\text{Polar}}}}{C_{\text{Meander}} D_{n_p}^{\text{Wake}}}\right) & \text{for method 2-truncated jinc} \\ jinc\left(\frac{r_{n^{\text{Polar}}}}{C_{\text{Meander}} D_{n_p}^{\text{Wake}}}\right) jinc\left(\frac{r_{n^{\text{Polar}}}}{2C_{\text{Meander}} D_{n_p}^{\text{Wake}}}\right) & \text{for method 3-windowed jinc} \end{cases} \quad (4.230)$$

The first method is a spatial average with a uniform weighting with a local polar-grid diameter of $C_{\text{Meander}} D_{n_p}^{\text{Wake}}$ at wake plane n_p , resulting in a cutoff wave number of $\frac{1}{C_{\text{Meander}} D_{n_p}^{\text{Wake}}}$. The second and third methods weight each point in the spatial average by a form of the *jinc* function dependent on the radius of the point from the wake centerline, $r_{n^{\text{Polar}}}$, normalized by $C_{\text{Meander}} D_{n_p}^{\text{Wake}}$. This results in a more ideal low-pass filter with a sharper cutoff of energy in the polar wave-number domain with a cutoff wave number of $\frac{1}{C_{\text{Meander}} D_{n_p}^{\text{Wake}}}$. However, because the *jinc* function decays slowly with increasing argument, the *jinc* function must be windowed to be applied in practice. The second method truncates the

¹¹ In this context, the *jinc* function is defined as $jinc(r) = \frac{J_1(2\pi r)}{r}$ (with the limiting value at the origin of $jinc(0) = \pi$), where $J_1(r)$ is the Bessel function of the first kind and order one. The *jinc* function is normalized such that $\int_0^\infty jinc(r) 2\pi r dr = 1$. The *jinc* function is the polar-equivalent of the one-dimensional sinc function defined as $\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$ (with the limiting value at the origin of $\text{sinc}(0) = 1$, which is the Fourier transform of a rectangular function, i.e., an ideal low-pass filter, and normalized such that $\int_{-\infty}^\infty \text{sinc}(x) dx = 1$).

jinc function at its first zero crossing, corresponding to a local polar-grid diameter of $1.21967C_{\text{Meander}}D_{n_p}^{\text{Wake}}$ at wake plane n_p . The third method windows the *jinc* function by multiplying it with a *jinc* function of half the argument (the polar-domain equivalent of a one-dimensional Lanczos/sinc window), which tapers the weighting to zero at its second zero crossing (the weighting is positive below the first zero crossing and negative past the first zero crossing until it tapers to zero). This corresponds to a local polar-grid diameter of $2.23313C_{\text{Meander}}D_{n_p}^{\text{Wake}}$ at wake plane n_p . These weighted spatial averaging methods improve the overall level and frequency content of the wake meandering at the expense of a bit heavier computations due to the larger polar-grid diameters (i.e., the truncated *jinc* method has roughly 50% more points within the polar grid than the uniform method, and the windowed *jinc* method has roughly five times more points than the uniform method). A value of $C_{\text{Meander}} = 2$, resulting in a polar-grid diameter of $2D^{\text{Wake}}$ and cutoff wave number of $\frac{1}{2D^{\text{Wake}}}$, follows the characteristic dimension important to transverse wake meandering proposed by Larsen et al. ([ff-Leal08]) C_{Meander} is included in all methods to enable the user of FAST.Farm to better match the meandering to known solutions. Note that the lower the value of C_{Meander} , the more the wake will meander.

Future Work

This list contains features that could be implemented in future releases:

- Develop more efficient methods of generating/processing ambient wind from a high-fidelity precursor simulation, including:
 - Propagate 2D planes of ambient wind data using Taylor’s frozen turbulence hypothesis as an alternative to 3D volumes
 - Allow for nonuniform grids in Turbsim
 - Use Dynamic Mode Decomposition to compress the file size of the low-resolution domains
 - Implement Gabor mode enrichment to replace the high-resolution domains
 - Develop a more efficient ABLSolver based on a simple rectangular (rather than a generally unstructured) grid.
- Improve the eddy-viscosity formulation with additional physics.
- Pursue additional wake-modeling approaches, including:
 - Introduce simpler wake-deficit models, e.g., the Gaussian wake model by Bastankhah and Porté-Agel and the super-Gaussian model by Blondel and Cathelain
 - Introduce simpler wake-deflection models, e.g., the model by Jiménez or the model by Qian and Ishihara
 - Apply a free-vortex method for the near wake
 - Incorporate a kidney-shaped wake under skewed-flow conditions, e.g., by incorporating opposing vortices from the skew-induced horseshoe vortex
 - Deform the base-wake deficit (introduce asymmetry) as a result of background turbulence (in addition to wake meandering)
 - Incorporate wake-added turbulence
 - Improve the treatment of complex terrain (beyond specifying ambient wind data as NaN in VTK format)
 - Include wakes from the nacelle and support structure
 - Reflect wakes off of the ground.
- Address deep-array effects for large wind farms and account for flow speedup around the edges of the wind farm – i.e., account for the wind-farm blockage effect – e.g., by mimicking the wind farm-induced boundary layer with surface roughness in the LES ambient wind precursor.
- Implement a model to mimic the measurements taken from a LIDAR and other remote sensing technologies.

- Incorporate MPI to support the modeling of large wind farms by taking advantage of memory parallelization and parallelization between nodes of an HPC.
- Allow for a more general module form, e.g.:
 - Support continuous states
 - Support direct feedthrough of input to output
 - Support full-system linearization.
- Support an interface to Simulink for super and individual wind turbine controllers.
- Implement checkpoint-restart capability.
- Enable binary wind data input and output formats and binary time-series results output format.
- Add ability to output disturbed wind in VTK format on 2D slices that need not be parallel to the X-Y, Y-Z and/or X-Z planes of the global inertial-frame coordinate system.
- Rename the ambient wind data input files in VTK format following the naming convention used for the FAST.Farm-generated visualization output files in VTK format (with leading zeros and without the t).
- Support super controller-, inflow-, and wake-related output channels for more than the first 9 wind turbines in the wind farm.
- Interface FAST.Farm to the Wind-Plant Integrated System Design & Engineering Model ([WISDEM™](#)) for systems-engineering applications (multidisciplinary design, analysis, and optimization; uncertainty quantification; and so on).
- Develop a wrapper for stand-alone AeroDyn – the aerodynamics module of OpenFAST (or an equivalent BEM tool) – as an alternative to OpenFAST to support advanced performance-only wind-farm analysis that is much more computationally efficient than FAST.Farm analysis using OpenFAST.
- Address unique offshore wind energy challenges, e.g.:
 - Ensure consistent waves across an offshore wind farm
 - Support the air-water interface
 - Consider shared mooring and anchoring arrangements (for floating offshore wind farms).
- Adopt the capability to support undersea marine turbine arrays (which may require supporting direct feedthrough of input to output to handle the added-mass effects).

FAST.Farm Primary Input File

When a default value is available, DEFAULT may be used instead of the value.

Check the regression test cases for updates to this input file.

```

1 FAST.Farm v1.00.* INPUT FILE
2 Sample FAST.Farm input file
3 --- SIMULATION CONTROL ---
4 False      Echo      Echo input data to <RootName>.ech? (flag)
5 FATAL      AbortLevel Error level when simulation should abort (string) {"WARNING",
6 ↪ "SEVERE", "FATAL"}
7 2000.0     TMax      Total run time (s) [>=0.0]
8 False      UseSC     Use a super controller? (flag)
9 1          Mod_AmbWind Ambient wind model (-) (switch) {1: high-fidelity precursor in
↪ VTK format, 2: one InflowWind module, 3: multiple instances of InflowWind module}
--- SUPER CONTROLLER --- [used only for UseSC=True]
```

(continues on next page)

(continued from previous page)

```

10 "SC_DLL.dll" SC_FileName Name/location of the dynamic library {.dll [Windows] or .so
    ↳[Linux]} containing the Super Controller algorithms (quoted string)
11 --- AMBIENT WIND: PRECURSOR IN VTK FORMAT --- [used only for Mod_AmbWind=1]
12 2.0      DT_Low-VTK      Time step for low-resolution wind data input files; will be
    ↳used as the global FAST.Farm time step (s) [>0.0]
13 0.5      DT_High-VTK     Time step for high-resolution wind data input files (s) [>0.0]
    ↳"/AmbWind/steady" WindFilePath Path name to wind data files from precursor (string)
14 False    ChkWndFiles     Check all the ambient wind files for data consistency (flag)
15 --- AMBIENT WIND: INFLOWWIND MODULE --- [used only for Mod_AmbWind=2 or 3]
16 2.0      DT_Low          Time step for low-resolution wind data interpolation; will be
    ↳used as the global FAST.Farm time step (s) [>0.0]
17 0.5      DT_High         Time step for high-resolution wind data interpolation (s) [>0.0]
18 300      NX_Low          Number of low-resolution spatial nodes in X direction for wind
    ↳data interpolation (-) [>=2]
19 300      NY_Low          Number of low-resolution spatial nodes in Y direction for wind
    ↳data interpolation (-) [>=2]
20 35       NZ_Low          Number of low-resolution spatial nodes in Z direction for wind
    ↳data interpolation (-) [>=2]
21 5.0      X0_Low          Origin of low-resolution spatial nodes in X direction for wind
    ↳data interpolation (m)
22 5.0      Y0_Low          Origin of low-resolution spatial nodes in Y direction for wind
    ↳data interpolation (m)
23 5.0      Z0_Low          Origin of low-resolution spatial nodes in Z direction for wind
    ↳data interpolation (m)
24 10.0     dX_Low          Spacing of low-resolution spatial nodes in X direction for wind
    ↳data interpolation (m) [>0.0]
25 10.0     dY_Low          Spacing of low-resolution spatial nodes in Y direction for wind
    ↳data interpolation (m) [>0.0]
26 10.0     dZ_Low          Spacing of low-resolution spatial nodes in Z direction for wind
    ↳data interpolation (m) [>0.0]
27 16       NX_High         Number of high-resolution spatial nodes in X direction for wind
    ↳data interpolation (-) [>=2]
28 16       NY_High         Number of high-resolution spatial nodes in Y direction for wind
    ↳data interpolation (-) [>=2]
29 17       NZ_High         Number of high-resolution spatial nodes in Z direction for wind
    ↳data interpolation (-) [>=2]
30 "InflowWind.dat" InflowFile Name of file containing InflowWind module input parameters
    ↳(quoted string)
31 --- WIND TURBINES ---
32 1         NumTurbines     Number of wind turbines (-) [>=1] [last 6 columns below used
    ↳only for Mod_AmbWind=2 or 3]
33 WT_X     WT_Y     WT_Z     WT_FASTInFile      X0_High Y0_High  Z0_High dX_High dY_High dZ_
    ↳High
34 (m)      (m)      (m)      (string)          (m)      (m)      (m)      (m)      (m)     
    ↳(m)
35 605.0    1500.0    0.0      "/FAST/Test18.fst"  525.0    1425.0    5.0      10.0    10.0    10.
    ↳0
36 --- WAKE DYNAMICS ---
37 5.0      dr             Radial increment of radial finite-difference grid (m) [>0.0]
38 40       NumRadii       Number of radii in the radial finite-difference grid (-) [>=2]
39 140      NumPlanes      Number of wake planes (-) [>=2]
40 DEFAULT  f_c           Cutoff (corner) frequency of the low-pass time-filter for the

```

(continues on next page)

(continued from previous page)

```

41  →wake advection, deflection, and meandering model (Hz) [ $>0.0$ ] or DEFAULT [=0.0007]
    DEFAULT C_Hwkdfl_0 Calibrated parameter in the correction for wake deflection,
    →defining the horizontal offset at the rotor (m) or DEFAULT [=0.0]
42  DEFAULT C_Hwkdfl_OY Calibrated parameter in the correction for wake deflection,
    →defining the horizontal offset at the rotor scaled with yaw error (m/deg) or DEFAULT
    →[=0.3]
43  DEFAULT C_Hwkdfl_x Calibrated parameter in the correction for wake deflection,
    →defining the horizontal offset scaled with downstream distance (-) or DEFAULT [=0.0]
44  DEFAULT C_Hwkdfl_xY Calibrated parameter in the correction for wake deflection,
    →defining the horizontal offset scaled with downstream distance and yaw error (1/deg),
    →or DEFAULT [=0.004]
45  DEFAULT C_NearWake Calibrated parameter for the near-wake correction (-) [ $>1.$  and
    → $<2.5$ ] or DEFAULT [=1.8]
46  DEFAULT k_vAmb Calibrated parameter for the influence of ambient turbulence in,
    →the eddy viscosity (-) [ $\geq 0.0$ ] or DEFAULT [=0.05]
47  DEFAULT k_vShr Calibrated parameter for the influence of the shear layer in,
    →the eddy viscosity (-) [ $\geq 0.0$ ] or DEFAULT [=0.016]
48  DEFAULT C_vAmb_DMin Calibrated parameter in the eddy viscosity filter function for,
    →ambient turbulence defining the transitional diameter fraction between the minimum and,
    →exponential regions (-) [ $\geq 0.0$ ] or DEFAULT [=0.0]
49  DEFAULT C_vAmb_DMax Calibrated parameter in the eddy viscosity filter function for,
    →ambient turbulence defining the transitional diameter fraction between the exponential,
    →and maximum regions (-) [ $> C\_vAmb\_DMin$ ] or DEFAULT [=1.0]
50  DEFAULT C_vAmb_FMin Calibrated parameter in the eddy viscosity filter function for,
    →ambient turbulence defining the value in the minimum region (-) [ $\geq 0.0$  and  $\leq 1.0$ ] or
    →DEFAULT [=1.0]
51  DEFAULT C_vAmb_Exp Calibrated parameter in the eddy viscosity filter function for,
    →ambient turbulence defining the exponent in the exponential region (-) [ $> 0.0$ ] or
    →DEFAULT [=0.01]
52  DEFAULT C_vShr_DMin Calibrated parameter in the eddy viscosity filter function for,
    →the shear layer defining the transitional diameter fraction between the minimum and,
    →exponential regions (-) [ $\geq 0.0$ ] or DEFAULT [=3.0]
53  DEFAULT C_vShr_DMax Calibrated parameter in the eddy viscosity filter function for,
    →the shear layer defining the transitional diameter fraction between the exponential,
    →and maximum regions (-) [ $> C\_vShr\_DMin$ ] or DEFAULT [=25.0]
54  DEFAULT C_vShr_FMin Calibrated parameter in the eddy viscosity filter function for,
    →the shear layer defining the value in the minimum region (-) [ $\geq 0.0$  and  $\leq 1.0$ ] or
    →DEFAULT [=0.2]
55  DEFAULT C_vShr_Exp Calibrated parameter in the eddy viscosity filter function for,
    →the shear layer defining the exponent in the exponential region (-) [ $> 0.0$ ] or DEFAULT
    →[=0.1]
56  DEFAULT Mod_WakeDiam Wake diameter calculation model (-) (switch) {1: rotor diameter,
    →2: velocity based, 3: mass-flux based, 4: momentum-flux based} or DEFAULT [=1]
57  DEFAULT C_WakeDiam Calibrated parameter for wake diameter calculation (-) [ $>0.0$ ,
    →and  $<0.99$ ] or DEFAULT [=0.95] [unused for Mod_WakeDiam=1]
58  DEFAULT Mod_Meander Spatial filter model for wake meandering (-) (switch) {1:
    →uniform, 2: truncated jinc, 3: windowed jinc} or DEFAULT [=3]
59  DEFAULT C_Meander Calibrated parameter for wake meandering (-) [ $\geq 1.0$ ] or DEFAULT,
    →[=1.9]
60  --- VISUALIZATION ---
61  False WrDisWind Write low- and high-resolution disturbed wind data to <RootName>
    →.Low.Dis.t<n>.vtk etc. (flag)

```

(continues on next page)

(continued from previous page)

```

62 1      NOutDisWindXY Number of XY planes for output of disturbed wind data across
    ↳ the low-resolution domain to <RootName>.Low.DisXY<n_out>.t<n>.vtk (-) [0 to 9]
63 900.0   OutDisWindZ   Z coordinates of XY planes for output of disturbed wind data
    ↳ across the low-resolution domain (m) [1 to NOutDisWindXY] [unused for NOutDisWindXY=0]
64 2      NOutDisWindYZ Number of YZ planes for output of disturbed wind data across
    ↳ the low-resolution domain to <RootName>/Low.DisYZ<n_out>.t<n>.vtk (-) [0 to 9]
65 600.0,978.0 OutDisWindX X coordinates of YZ planes for output of disturbed wind data
    ↳ across the low-resolution domain (m) [1 to NOutDisWindYZ] [unused for NOutDisWindYZ=0]
66 1      NOutDisWindXZ Number of XZ planes for output of disturbed wind data across
    ↳ the low-resolution domain to <RootName>/Low.DisXZ<n_out>.t<n>.vtk (-) [0 to 9]
67 1500.0   OutDisWindY   Y coordinates of XZ planes for output of disturbed wind data
    ↳ across the low-resolution domain (m) [1 to NOutDisWindXZ] [unused for NOutDisWindXZ=0]
68 4.0      WrDisDT       Time step for disturbed wind visualization output (s) [>0.0] or
    ↳ DEFAULT [=DT_Low or DT_Low-VTK] [unused for WrDisWind=False and
    ↳ NOutDisWindXY=NOutDisWindYZ=NOutDisWindXZ=0]
69 --- OUTPUT ---
70 True      SumPrint      Print summary data to <RootName>.sum? (flag)
71 99999.9   ChkptTime     Amount of time between creating checkpoint files for potential
    ↳ restart (s) [>0.0]
72 200.0     TStart        Time to begin tabular output (s) [>=0.0]
73 1         OutFileFmt     Format for tabular (time-marching) output file (-) (switch) {1:
    ↳ text file [<RootName>.out], 2: binary file [<RootName>.outb], 3: both}
74 True      TabDelim      Use tab delimiters in text tabular output file? (flag) {uses
    ↳ spaces if False}
75 "ES10.3E2" OutFmt        Format used for text tabular output, excluding the time channel.
    ↳ Resulting field should be 10 characters. (quoted string)
76 3         NOutRadII     Number of radial nodes for wake output for an individual rotor
    ↳ (-) [0 to 20]
77 0, 15, 39 OutRadII     List of radial nodes for wake output for an individual rotor (-)
78 2         NOutDist      Number of downstream distances for wake output for an
    ↳ individual rotor (-) [1 to NOutRadII] [unused for NOutRadII=0] rotor (-) [0 to 9]
79 0.0, 378.0 OutDist      List of downstream distances for wake output for an individual
    ↳ rotor (m) [1 to NOutDist] [unused for NOutDist =0]
80 1         NWindVel      Number of points for wind output (-) [0 to 9]
81 600.0     WindVelX      List of coordinates in the X direction for wind output (m) [1
    ↳ to NWindVel] [unused for NWindVel=0]
82 1500.0    WindVelY      List of coordinates in the Y direction for wind output (m) [1
    ↳ to NWindVel] [unused for NWindVel=0]
83 90.0      WindVelZ      List of coordinates in the Z direction for wind output (m) [1
    ↳ to NWindVel] [unused for NWindVel=0]
84 OutList The next line(s) contains a list of output parameters. (quoted string)
85 "RtAxsXT1, RtAxsYT1, RtAxsZT1"
86 "RtPosXT1, RtPosYT1, RtPosZT1"
87 "YawErrT1"
88 "TIAmbT1"
89 "CtT1N01, CtT1N02, CtT1N03, CtT1N04, CtT1N05"
90 "WkAxsXT1D1, WkAxsXT1D2, WkAxsXT1D3"
91 "WkAxsYT1D1, WkAxsYT1D2, WkAxsYT1D3"
92 "WkAxsZT1D1, WkAxsZT1D2, WkAxsZT1D3"
93 "WkPosXT1D1, WkPosXT1D2, WkPosXT1D3"
94 "WkPosYT1D1, WkPosYT1D2, WkPosYT1D3"
95 "WkPosZT1D1, WkPosZT1D2, WkPosZT1D3"

```

(continues on next page)

(continued from previous page)

```

96 "WkDfVxT1N01D1, WkDfVxT1N02D1, WkDfVxT1N03D1, WkDfVxT1N04D1, WkDfVxT1N05D1"
97 "WkDfVxT1N01D2, WkDfVxT1N02D2, WkDfVxT1N03D2, WkDfVxT1N04D2, WkDfVxT1N05D2"
98 "WkDfVxT1N01D3, WkDfVxT1N02D3, WkDfVxT1N03D3, WkDfVxT1N04D3, WkDfVxT1N05D3"
99 "WkDfVrT1N01D1, WkDfVrT1N02D1, WkDfVrT1N03D1, WkDfVrT1N04D1, WkDfVrT1N05D1"
100 "WkDfVrT1N01D2, WkDfVrT1N02D2, WkDfVrT1N03D2, WkDfVrT1N04D2, WkDfVrT1N05D2"
101 "WkDfVrT1N01D3, WkDfVrT1N02D3, WkDfVrT1N03D3, WkDfVrT1N04D3, WkDfVrT1N05D3"
102 END of input file (the word "END" must appear in the first 3 columns of this last
    ↳OutList line)

```

Ambient Wind File

```

# vtk DataFile Version 3.0
Amb.coarse
ASCII
DATASET STRUCTURED_POINTS
DIMENSIONS 300 300 35
ORIGIN 5 5 5
SPACING 10 10 10
POINT_DATA 3150000
VECTORS Amb FLOAT
 4.852      0.217      -0.009
 5.092      0.213      -0.077
 5.248      0.418      -0.060
 5.280      0.794      -0.045
 5.125      0.993      -0.046
 5.067      0.799      -0.118
 5.272      0.481      -0.242
 5.624      0.410      -0.312
 5.724      0.531      -0.266
 5.267      0.430      -0.167
 4.597      0.055      -0.109
 4.204      -0.322      -0.069
 4.248      -0.441      -0.055
 4.569      -0.159      -0.171
 4.793      0.259      -0.423
 4.658      0.431      -0.647
 4.287      0.395      -0.572
 3.955      0.355      -0.208
 3.849      0.252       0.233
 3.938      0.099       0.534
...
[3,149,970 lines removed]
...
10.576      -0.273       0.244
10.910      -0.287      -0.051
11.207      -0.241      -0.349
11.393      -0.172      -0.524
11.513      -0.139      -0.528
11.581      -0.210      -0.411
11.647      -0.315      -0.219

```

(continues on next page)

(continued from previous page)

11.516	-0.348	-0.039
11.514	-0.252	0.185
10.977	0.058	0.245

List of Output Channels

This is a list of all possible output parameters available within FAST.Farm (except those that are available from OpenFAST, which are specified within the OpenFAST input file(s) and output separately for each turbine). The names are grouped by meaning, but can be ordered in the OUTPUTS section of the FAST.Farm primary input file as you see fit.

$T\alpha$ refers to turbine α , where α is a one-digit number in the range [1,9], corresponding to row α in the wind turbine input table. If **NumTurbines** > 9, only values for the first 9 turbines can be output. Setting $\alpha > \text{NumTurbines}$ yields invalid output.

$In\zeta$ and $Ot\zeta$ refer to super-controller input and output ζ , respectively, where ζ is a one-digit number in the range [1,9], corresponding to element ζ in the input and output arguments of the super-controller source code. If there are more than 9 elements, only values for the first 9 inputs and outputs can be output. Setting ζ greater than the number of elements yields invalid output.

$N\beta$ refers to radial output node β , where β is a two-digit number in the range [01,20], corresponding to entry β in the **OutRadii** list, where node β is at radius $\mathbf{dr} \times \text{OutRadii}[\beta]$. Setting $\beta > \text{NOutRadii}$ yields invalid output.

$W\eta$ refers to wind point η , where η is a one-digit number in the range [1,9], corresponding to entry η in the **WindVelX**, **WindVelY**, and **WindVelZ** lists. Setting $\eta > \text{NWindVel}$ yields invalid output. Setting **WindVelX**, **WindVelY**, and **WindVelZ** outside the low-resolution wind domain also yields invalid output.

δ refers to the X, Y, or Z coordinate axis.

$D\gamma$ refers to downstream distance γ , where γ is a one-digit number in the range [1,9], corresponding to entry γ in the **OutDist** list. Setting $\gamma > \text{NOutDist}$ yields invalid output. The output is also invalid if **OutDist** is a distance further downstream than the wake has been calculated or for any distance where the wake from the turbine has overlapped itself.

Table 4.21: List of Available FAST.Farm Output Channels

Channel Name	Units	Description
<i>Super Controller</i>		
SCGblIn ζ	(user)	Global (turbine independent) super controller input ζ
SCT α In ζ	(user)	Turbine-dependent super controller input ζ for turbine α
SCGblOut ζ	(user)	Global (turbine independent) super controller output ζ
SCT α Out ζ	(user)	Turbine-dependent super controller output ζ for turbine α
<i>Wind Turbine and Inflow</i>		
RtAxs δT_α	(-)	Orientation of the rotor centerline for turbine α in the global coordinate system
RtPos δT_α	(m)	Position of the rotor (hub) center for turbine α in the global coordinate system
RtDiam T_α	(m)	Rotor diameter for turbine α
YawErr T_α	(deg)	Nacelle-yaw error for turbine α
TIAmb T_α	(%)	Ambient turbulence intensity of the wind at the the rotor disk for turbine α . The ambient turbulence intensity is based on a spatial-average of the three vector components, instead of just the axial component.
RtVAmb T_α	(m/s)	Rotor-disk-averaged ambient wind speed (normal to disk, not including structural motion, local induction or wakes from upstream turbines) for turbine α
RtVRel T_α	(m/s)	Rotor-disk-averaged relative wind speed (normal to disk, including structural motion and wakes from upstream turbines, but not including local induction) for turbine α
Ct $T_\alpha N_\beta$	(-)	Azimutally averaged thrust force coefficient (normal to disk) for radial output node β of turbine α
<i>Wake (for an Individual Rotor)</i>		
WkAxs $\delta T_\alpha \delta \gamma$	(-)	Orientation of the wake centerline for downstream distance γ of turbine α in the global coordinate system
WkPos $\delta T_\alpha \delta \gamma$	(m)	Center position of the wake centerline for downstream distance γ of turbine α in the global coordinate system
WkVel $\delta T_\alpha \delta \gamma$	(m/s)	Advection, deflection, and meandering velocity (not including the horizontal wake-deflection correction or low-pass time-filtering) of the wake for downstream distance γ of turbine α in the global coordinate system
WkDiam $T_\alpha \delta \gamma$	(m)	Wake diameter for downstream distance γ of turbine α
WkDFVx $T_\alpha N_\beta \delta \gamma$	(m/s)	Axial wake velocity deficits for radial output node β and downstream distance γ of turbine α
WkDFVr $T_\alpha N_\beta \delta \gamma$	(m/s)	Radial wake velocity deficits for radial output node β and downstream distance γ of turbine α
EddVis $T_\alpha N_\beta \delta \gamma$	(m ² /s)	Total eddy viscosity for radial output node β and downstream distance γ of turbine α
EddAmb $T_\alpha N_\beta \delta \gamma$	(m ² /s)	Individual contribution to the eddy viscosity from ambient turbulence for radial output node β and downstream distance γ of turbine α
EddShr $T_\alpha N_\beta \delta \gamma$	(m ² /s)	Individual contributions to the eddy viscosity from the shear layer for radial output node β and downstream distance γ of turbine α
<i>Ambient Wind and Array Effects</i>		
W η VAmb δ	(m/s)	Ambient wind velocity (not including wakes) for point η in the global coordinate system (from the low-resolution domain)
W η VDis δ	(m/s)	Disturbed wind velocity (including wakes) for point η in the global coordinate system (from the low-resolution domain)

The following modules do not currently have formal documentation or are contributed to OpenFAST from organizations external to NREL and the core OpenFAST team. As documentation is added, these resources will be moved to their appropriate location. If newer versions of the external resources are available, please open a [GitHub Issue](#) with the information for the new documentation.

- MAP++
 - [Official MAP++ documentation](#)

- Implementation of a Multi-Segmented, Quasi-Static Cable Model
- FEAMooring
 - Theory Manual
 - User's Guide
- MoorDyn
 - [Official User's Guide](#)
- OrcaFlex Interface:
 - User's Guide
- IceFloe
 - Ice Load Project Final Technical Report
- IceDyn
 - Draft: FAST Ice Module Manual
- TurbSim
 - User's Guide

4.3 Modularization Framework

Information specific to the modularization framework of OpenFAST is provided here. These are a collection of publications, presentations, and past studies on the subject.

- [The New Modularization Framework for the FAST Wind Turbine CAE Tool](#)
- [Example Module Implementation Plans](#)
- [Module and Mesh-Mapping Linearization Implementation Plan](#)
- [Interpolation of DCMs](#) - A summary of the mathematics used in the interpolation of DCM (direction cosine matrices) using logarithmic mapping and matrix exponentials.
- [Set-point Linearization Development Plan](#)

4.4 Glue Code and Mesh Mapping

- [FAST Modular Wind Turbine CAE Tool: Nonmatching Spatial and Temporal Meshes](#)
- [FAST Modular Framework for Wind Turbine Simulation: New Algorithms and Numerical Examples](#)
- [OpenFAST Algorithms](#) - A summary of the solve method used in the glue code.
- [Predictor-Corrector Approach](#)

4.5 NWTC Subroutine Library

- NWTC Library - short overview of subroutines and functions

DEVELOPER DOCUMENTATION

Our goal as developers of OpenFAST is to ensure that it is well tested, well documented, and self-sustaining software. To that end, we continually work to improve the documentation and test coverage along with feature additions and improvements. This section of the documentation outlines the processes and procedures we have established for external developers to work with the NREL OpenFAST team on code development.

If you'd like to help with general OpenFAST development or work on a particular feature, then first install OpenFAST following the *installation instructions* for your machine. Next, verify that your installation is valid by running the test suite following the *testing instructions*. While OpenFAST is compiling, we encourage reading through the *Development Philosophy and Guidelines* section to understand the general workflow for individual and coordinated development. Finally, be sure to review the *GitHub workflow* to avoid any merge or code conflicts.

With development happening in parallel between NREL, industry partners, and universities, NREL relies on GitHub to coordinate efforts:

- [GitHub Issues](#) is the place to ask usage or development questions, report bugs, and suggest code enhancements
- [GitHub Pull Requests](#) is the place for engaging with the OpenFAST team to have your new code merged into the main repository.

For other questions regarding OpenFAST, please contact [Mike Sprague](#).

Tip: The following sections provide valuable guidance on workflow and development tips which make the process more efficient and effective:

- *[Working with OpenFAST on GitHub](#)*
 - *[Code Style](#)*
 - *[Debugging OpenFAST](#)*
-

5.1 Development Philosophy and Guidelines

OpenFAST is intended to be a self-sustaining, community developed software. While the NREL OpenFAST team serves as the gatekeeper of the repository, we actively encourage the community to share new ideas and contribute code. Considerations for contributing code are outlined here.

5.1.1 Engagement with NREL

The process for community code contribution starts with engaging directly with the NREL OpenFAST team to define the scope of the work and coordinate development efforts. This is particularly important since many groups work on OpenFAST simultaneously. By engaging early, all developers can stay up to date and minimize conflicts during the code merge. The preferred method of communication is [GitHub Issues](#). An initial post should contain all relevant information about the planned development work, the areas of the software that will be impacted, and any model validation materials. See [Development Plan / Implementation Plan](#) for more information on describing the planned work.

The NREL OpenFAST team is always working on internal projects that require the majority of our attention, but we will make every effort to engage with the community and support development efforts in a reasonable time frame. After posting an Issue, the NREL OpenFAST team may reach out to schedule a meeting to talk through the details.

5.1.2 Development Plan / Implementation Plan

Significant code development efforts at NREL begin with the development of a detailed implementation plan, and a few such plans are available to download for reference:

- [Development Plan for the Aerodynamic Linearization of OpenFAST](#)
- [FAST.Farm Development Plan](#).
- [Implementation Plan - 2nd-order Forces Within HydroDyn](#)
- [Implementation Plan - 2nd-order Wave Kinematics Within HydroDyn](#)

A good plan within the modularization framework of OpenFAST will follow the definitions and nomenclature used by the [NWTC Programmer's Handbook](#). It should communicate the following information:

- State whether the module is intended for loose coupling, tight coupling for time marching, and/or linearization.
- Define the module's inputs (including initialization), outputs (including initialization), states (continuous, discrete, and constraint), and parameters, including units.
- Lay out an example input file for the module.
- Explain the module's mathematical formulation, including Jacobians (for tight coupling and linearization), in the form required of the framework.
- Prescribe how the module's inputs are derived from the outputs of other specific modules
- Identify any potential numerical problems and how to avoid them in the code.
- Lay out the module's subroutines using pseudocode (as opposed to actual code), including identifying which mathematical formulas are used by which subroutines, and describing the algorithms used in the solution process.

This information is very helpful since it is easier to review, iterate, and agree on a plan before making changes to source code. Additionally, an implementation plan will greatly aid in the programming effort and is a useful starting point for writing the user and developer documentation.

5.1.3 Qualities of a good submission

Development efforts should include adequate testing throughout the development process. When possible, new subroutines should include unit-level tests, and the existing regression tests should be run periodically to ensure that the full system behavior has not changed unintentionally. For new features, additional regression tests should be added to cover the new code. If the regression test results change in an expected manner, the baseline results should be updated locally and in the [openfast/r-test](#) repository. The [r-test README](#) describes updating the baselines and the [Testing OpenFAST](#) section in this documentation contains additional details on testing.

New code should consider robustness from both the developer and user perspectives. Here are some questions to consider during code development:

- Is it clear to other developers how to use your subroutine?
- Does your new code exhibit clear and predictable behavior?
- How will your code perform under different qualities of data?
- How does your code impact the performance of the simulation?

Additionally, user and developer documentation should be included with new code. User documentation includes theory, modeling guidance, and a description of any inputs and outputs. User documentation should be included as part of the online documentation described in [Developing Documentation](#). Developer documentation is typically included in comments in the source code. This should describe subroutine API's (inputs and outputs) as well as any algorithms or lines of code that are unclear. Ask yourself what you would need to know to fully understand your code if you don't see it again for two years.

5.1.4 Submit for review and NREL feedback

New code can be submitted for review from the NREL OpenFAST team by opening a [pull request](#) as described in [Working with OpenFAST on GitHub](#). We will review the code for accuracy, validity, quality, and robustness. Reviewing open source code contributions can be difficult, so it is worthwhile to review your own code and consider what information would help you to determine whether it is ready to merge.

The review process begins with simply ensuring that the automated tests pass in [GitHub Actions](#). **Please ensure that all automated tests pass prior to requesting a review.** After that, the process will involve some communication between the reviewer and the submitter, possibly requests for more information on the background or validation, and comments in the pull request to gain additional insight into specific lines of code.

After a consensus is reached between the submitter and reviewer, the pull request will be merged into the target branch (typically *dev*) and the pull request will be closed. You're done! This change will be included in the subsequent release of OpenFAST when the *dev* branch is merged into *main*.

5.1.5 Bug fixes

If you've found a bug in the code, it is important to fully describe it both in a [GitHub Issue](#) and through a minimal test. Before making a commit with the bug fix, commit the new test that exposes the bug. This test should fail. Then, commit the bug fix and show that the test passes. The git-commit history should look something like this (progresses bottom to top):

See [Testing OpenFAST](#) and [Working with OpenFAST on GitHub](#) for more information.

5.1.6 Additional guidance

The following sections provide extended guidance on developing source code, interacting with the NREL OpenFAST team and other community contributors, and generally debugging and building out features.

Working with OpenFAST on GitHub

The majority of the collaboration and development for OpenFAST takes place on the [GitHub repository](#). There, [issues](#) and [pull requests](#) are discussed and new versions are released. It is the best mechanism for engaging with the NREL OpenFAST team and other developers throughout the OpenFAST community.

Issues and work assignment

Issues should be opened with proper documentation and data to fully describe the problem or feature gap. It is here that communication and coordination should happen regarding ongoing work for new development, and developers should make clear any intention to complete a task.

Pull Requests

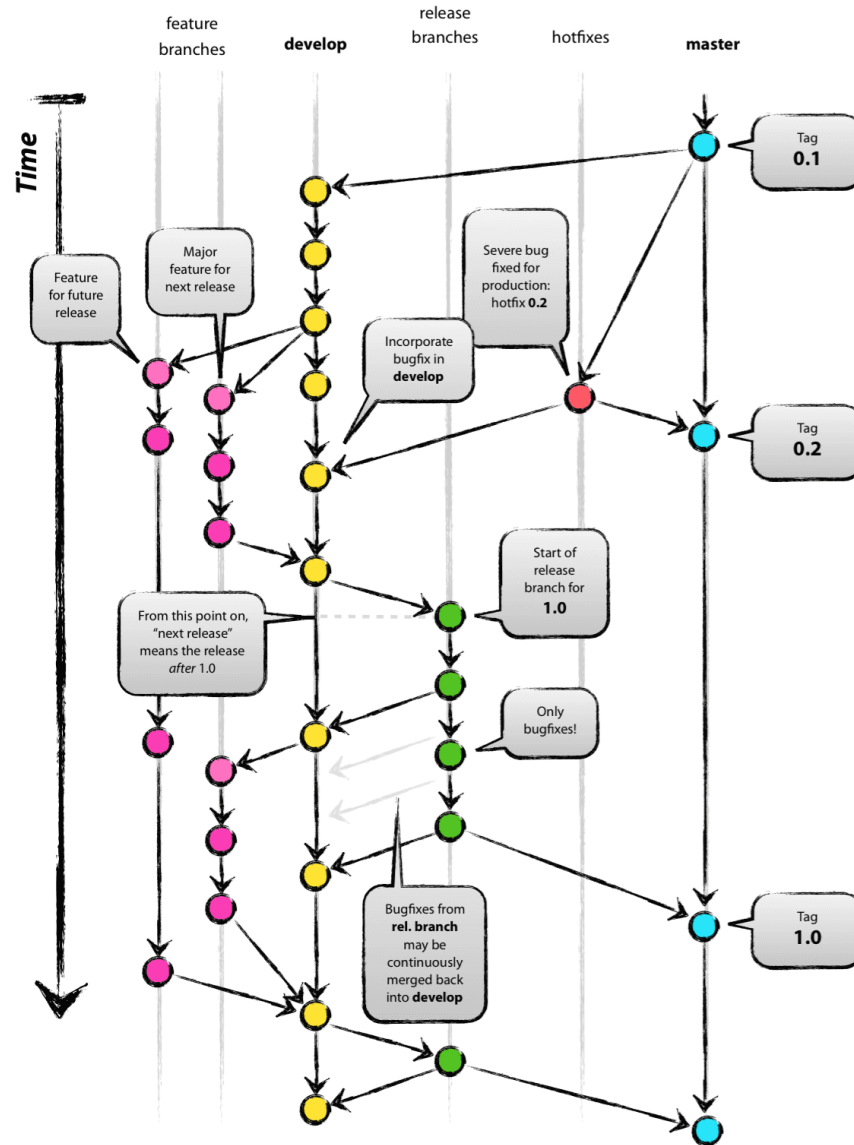
When a code modification is ready for review, a pull request should be submitted along with all appropriate documentation and tests. An NREL OpenFAST team member will assign a reviewer and work with the developer to have the code merged into the main repository.

New pull requests should contain the following:

- A description of the need for modifications
 - If the pull request fixes a bug, the accompanying GitHub issue should be referenced
- A summary of the work implemented
- Regression test results
 - If all tests pass, the summary print out should be provided
 - If any tests fail, an explanation of the failing cases and supporting data like plots should be included
- Updated unit tests, if applicable
- Updated documentation in applicable sections ready for compilation and deployment to [readthedocs](#).

Git workflow and interacting with the main repository

OpenFAST development should follow “Git Flow” when interacting with the github repository. Git Flow is a well-defined and widely adopted workflow for using git that outlines safe methods of pushing and pulling commits to a shared repository. Maintaining Git Flow is critical to prevent remote changes from blocking your local development. This workflow is detailed nicely [here](#) and the chart below provides a high level perspective.



Reference: <http://nvie.com/posts/a-successful-git-branching-model>

OpenFAST Specific Git Flow

It is important to consider how your current work will be affected by other developer's commits and how your commits will affect other developers. On public branches, avoid using `git rebase` and never `force push`.

In OpenFAST development, the typical workflow follows this procedure:

1. Fork the OpenFAST repository on GitHub
2. Clone your new fork

```
git clone https://github.com/<youruser>/OpenFAST
```

3. Add OpenFAST/OpenFAST as a remote named `upstream`

```
# This adds the remote
git remote add upstream https://github.com/OpenFAST/OpenFAST

# This downloads all the info in the remote, but it doesn't change
# the local source code
git fetch --all
```

4. Create a feature branch for active development starting from the OpenFAST dev branch and check it out

```
git branch feature/a_great_feature upstream/dev
git checkout feature/a_great_feature
```

5. Add new development on feature/a_great_feature

```
git add a_file.f90
git commit -m "A message"
git push origin feature/a_great_feature
```

6. Update your feature branch with upstream

```
git pull upstream dev
git push origin feature/a_great_feature
```

7. Open a new [pull request](#) to merge <youruser>/OpenFAST/feature/a_great_feature into OpenFAST/OpenFAST/dev

Code Style

OpenFAST and its underlying modules are mostly written in Fortran adhering to the 2003 standard, but modules can be written in C or C++. The NWTC Programmer's Handbook is the definitive reference for all questions related to working with the FAST Framework and adding code to OpenFAST.

Generally, code should be written such that it is straightforward to read. Syntactic sugar or brevity should not detract from readability. The exception to this is in situations where performance requires poorly readable code. Here, comment blocks should be used to describe what is not readily apparent in the code. Indentation is typically three spaces and no tabs.

Developing Documentation

OpenFAST documentation is hosted on [readthedocs](#). It is automatically generated through the `readthedocs` build system from both the `main` and `dev` branches whenever new commits are added. This documentation uses the [restructured text](#) markup language.

Building this documentation locally

The documentation is compiled with [Sphinx](#), which is a Python based tool. Install it and the other required Python packages listed in `openfast/docs/requirements.txt` with `pip` or another Python package manager.

These additional packages are optional and are not included in the requirements file:

- [Doxygen](#)
- [Doxylink](#)
- [Graphviz](#)
- [LaTeX](#)

Doxygen and Graphviz can be installed directly from their website or with a package manager like `brew`, `yum`, or `apt`.

The result of building the documentation locally will be a set of HTML files and their accompanying required files. The main HTML file will exist `openfast/build/docs/html/index.html`. This file can be opened with any browser to view and navigate the locally-generated documentation as if it were any other web site.

Pure python build

If CMake and Make are not available on your system, the documentation can be generated directly with *sphinx*.

Note: This method does not generate the API documentation through Doxygen.

First, align your directory structure to the standard OpenFAST build by creating a directory at `openfast/build`. Then, move into `openfast/build` and run this command:

```
# sphinx-build -b <builder-name> <source-directory> <output-directory>
sphinx-build -b html ../docs ../docs/html
```

If this completes successfully, an html file will be created at `build/docs/html/index.html` which can be opened with any web browser.

Building with CMake and Make

In the OpenFAST directory, create a `build` directory and move into it. Then, run CMake with this flag: `-DBUILD_DOCUMENTATION=ON`. CMake will configure the build system with the necessary files for building the documentation.

Next, run the command to compile the docs:

```
make docs
```

This will first build the Doxygen API documentation and then the Sphinx documentation. If this completes successfully, a html file will be created at `build/docs/html/index.html` which can be opened with any web browser.

The full procedure for configuring and building the documentation is:

```
mkdir build
cd build
cmake .. -DBUILD_DOCUMENTATION=ON
make docs
```

If any modifications are made to the source files in `openfast/docs/source`, you can simply update the html files by executing the `make` command again.

The table below lists make-targets related to the documentation.

Target	Command	Output location
Full docs	<code>make docs</code>	<code>openfast/build/docs/html/index.html</code>
Full docs	<code>make sphinx</code>	<code>openfast/build/docs/html/index.html</code>
Doxygen API Reference	<code>make doxygen</code>	
HTML only	<code>make sphinx-html</code>	<code>openfast/build/docs/html/index.html</code>
PDF only	<code>make sphinx-pdf</code>	<code>openfast/build/docs/latex/Openfast.pdf</code>

Adding documentation

Coming soon. Feel like contributing? Start here!

Types Files and the OpenFAST Registry

Being a modern software project, OpenFAST has a complex system of custom data types. In Fortran, these are known as “derived data types.” Each module contains a unique collection of derived types which may add on to but must comply with the OpenFAST Framework. The module types are generally auto-generated by an included program called *OpenFAST Registry*. The OpenFAST Registry is written in C and adapted from a similar utility used in [WRF](#). Visit the [OpenFAST Registry README](#) for more information.

The OpenFAST Registry requires an input file to describe the necessary types for a given module. Generally, all module use a similar naming convention, `<Module>_Registry.txt`, and resulting Fortran code will be in a file called `<Module>_Types.f90`. For example, the AeroDyn OpenFAST Registry input file is located at `openfast/modules/aerodyn/src/AeroDyn_Registry.txt` and the resulting auto-generated Fortran source code is at `openfast/modules/aerodyn/src/AeroDyn_Types.f90`.

Since the types-modules are autogenerated, any changes to the data types directly should be expressed in the OpenFAST Registry input files so that the changes are not subsequently overwritten.

Compiling the OpenFAST Registry

The OpenFAST Registry is included in the OpenFAST build system through CMake. However, the default is to **not** compile the OpenFAST Registry executable and instead use the types modules that are included in *git* while compiling OpenFAST. To include the OpenFAST Registry in the build process and compile the Registry program, configure CMake with the `GENERATE_TYPES` flag:

```
cmake .. -DGENERATE_TYPES=ON
```

With `GENERATE_TYPES` enabled, CMake will configure the *openfast-registry* target to compile as a dependency of all other targets. The OpenFAST Registry executable will be found in `openfast/build/modules/openfast-registry/openfast-registry`.

Regenerating a types module

With the `GENERATE_TYPES` flag enabled, an additional step will be added to modules that are configured can make use of the OpenFAST Registry. The additional step will execute the OpenFAST Registry and regenerate the types module overwriting the existing modules. Any changes to the types module will be evident in *git*. For modules where the registry input file has not changed, the resulting types module will not change. However, for registry input files that have been modified, the output types module will be recompiled.

Adding a new types module

The process for adding a new types module follows *Regenerating a types module* closely. Here, an additional step is required to configure CMake to execute the Registry on the new input file and include the resulting types module in the compile step.

First, a new OpenFAST Registry input file must be created. Then, it must be configured to pass through the Registry in the corresponding module's `CMakeLists.txt`:

```
# This is the control statement for allowing the Registry to execute
if (GENERATE_TYPES)

    # Here is the CMake wrapper-function to execute the Registry
    # syntax: generate_f90_types(<Registry input file> <output file location>)
    generate_f90_types(src/AeroDyn_Registry.txt ${CMAKE_CURRENT_LIST_DIR}/src/AeroDyn_
↳Types.f90)
    generate_f90_types(src/New_Registry.txt ${CMAKE_CURRENT_LIST_DIR}/src/New_Types.f90)
endif()
```

Finally, the resulting types module must be added to the source files for the corresponding module:

```
# AeroDyn lib
set(AD_LIBS_SOURCES
    src/AeroDyn.f90
    src/AeroDyn_IO.f90
    src/AirfoilInfo.f90
    src/BEMT.f90
    src/DBEMT.f90
    src/BEMTUncoupled.f90
    src/UnsteadyAero.f90
    src/fmin_fcn.f90
    src/mod_rootldim.f90
    src/AeroDyn_Types.f90
    src/AirfoilInfo_Types.f90
    src/BEMT_Types.f90
    src/DBEMT_Types.f90
    src/UnsteadyAero_Types.f90

    # Add the new types module here
```

(continues on next page)

(continued from previous page)

```
src/New_Types.f90
)
```

With CMake properly configured, a message will display during the build process indicating that the OpenFAST Registry is executing:

```
[ 64%] Generating ../../modules/aerodyn/src/New_Types.f90

----- FAST Registry -----
-----
input file: /Users/rmudafor/Development/openfast/modules/aerodyn/src/New_Registry.txt
# more build process output will follow
```

And finally there should be an indication that the resulting types module is compiled:

```
Scanning dependencies of target aerodynlib
[ 70%] Building Fortran object modules/aerodyn/CMakeFiles/aerodynlib.dir/src/New_Types.
↪f90.o
```

Debugging OpenFAST

Being a Fortran project, OpenFAST can be challenging to debug and the process is unique for each system and environment. Keep in mind that some OpenFAST cases can be quite large in their memory footprint and may take a long time to reach the point of interest in the code. Choosing a test case carefully could save a significant amount of time.

It may be helpful to write a small Fortran program to verify that all debugging tools are set up properly before diving in to OpenFAST. Be sure to simulate a bug by doing something like accessing an array element that is not allocated and verify that you can catch the bug with a given set of tools.

Note: A requirement for all systems is to compile OpenFAST in **debug** mode.

Debugging on Windows

Windows developers using Intel tools can use the Visual Studio solution included in the OpenFAST repository for debugging. This is a straightforward process with lots of support from Intel.

Otherwise, Windows developers compiling in Unix-style environments should proceed to *Debugging on Linux and macOS*.

Debugging on Linux and macOS

First, compile OpenFAST in debug mode by setting `CMAKE_BUILD_TYPE` to “Debug”. This can be done on the command line with:

```
cmake .. -D CMAKE_BUILD_TYPE=Debug
```

or by using `ccmake` to open the command line `cmake` GUI to change it.

The GNU debugger, `gdb`, works well for debugging compiled code. It has a comprehensive command line interface which enables developers to add breakpoints and inspect variables.

Driving the debugger through an IDE can make inspecting the code much more efficient. One IDE known to work well is [Visual Studio Code](#) with the [Native Debug](#) extension. You can set up a [launch configuration](#) so that you can debug a particular OpenFAST case through the IDE. To do this, open the launch configuration and add a block similar to this:

```
{
  "name": "AOC_WSt",
  "type": "gdb",
  "request": "launch",
  "printCalls": false,
  "showDevDebugOutput": false,
  "valuesFormatting": "prettyPrinters",
  "gdbpath": "gdb",
  "target": "${workspaceRoot}/build/glue-codes/openfast/openfast",
  "cwd": "${workspaceRoot}/build/reg_tests/glue-codes/openfast/AOC_WSt/",
  "arguments": "${workspaceRoot}/build/reg_tests/glue-codes/openfast/AOC_WSt/AOC_WSt.
↪fst",
}
```

macOS-specific configuration

GDB on macOS needs some configuration before the system allows it to take over a process. It is recommended that gdb be installed with homebrew

```
brew info gdb
brew install gdb
```

After that completes, be sure to follow the caveats to finish the installation. For `gdb 8.2.1`, it looks like this:

```
==> Caveats
gdb requires special privileges to access Mach ports.
You will need to codesign the binary. For instructions, see:

https://sourceware.org/gdb/wiki/BuildingOnDarwin

On 10.12 (Sierra) or later with SIP, you need to run this:

echo "set startup-with-shell off" >> ~/.gdbinit
```

For Native Debug on macOS, you have to sort of hack the extension to allow breakpoints in fortran files by adding this line to `.vscode/settings.json`:

```
{
  "debug.allowBreakpointsEverywhere": true
}
```

Performance Profiling and Tuning

A major focus of the OpenFAST team is performance-profiling and optimization of the OpenFAST software with the goal of improving time-to-solution performance for the most computationally expensive use cases. The process generally involves initial profiling and hotspot analysis, then identifying specific subroutines to target for optimization in the physics modules and glue-codes.

A portion of this work was supported by Intel® through its designation of NREL as an [Intel® Parallel Computing Center \(IPCC\)](#).

The procedures, findings, and recommended programming practices are presented here. This is a working document and will be updating as additional studies are completed.

High-performance programming techniques

Being a compiled language designed for scientific and engineering applications, Fortran is well suited for producing very efficient code. However, the process of tuning code requires developers to understand the language as well as the tools available (compilers, performance libraries, etc) in order to generate the highest performance. This section identifies programming patterns to use Fortran and the Intel® Fortran compiler effectively. Developers should also reference the Intel® Fortran compiler documentation regarding [optimization](#), in general, and especially the sections on [automatic vectorization](#) and [coarrays](#).

Optimization report

When evaluating compiler optimization performance in a complex software, timing tests alone are not a good indication for the compiler's ability to optimize particular lines of code. For low-level information on the compilers attempts in optimization, developers should generate optimization reports to get line-by-line reporting on various metrics such as vectorization, parallelization, memory and cache usage, threading, and others. Developers should refer to the Intel® Fortran compiler documentation on [optimization reports](#).

For Linux and macOS, the OpenFAST CMake configuration has compiler flags for generating optimization reports available but commented in the `set_fast_intel_fortran_posix` macro in `openfast/cmake/OpenfastFortranOptions.cmake`. Primarily, the `qopt-report-phase` and `qopt-report` flags should be used. See the optimization report options [documentation](#) for more information on additional flags and configurations.

With compiler flags correctly configured, the compiler will output files with the extension `.optrpt` alongside the intermediate compile artifacts like `.o` files. The compile process will state that additional files are being generated:

```
ifort: remark #10397: optimization reports are generated in *.optrpt files in the output_
↳location
```

And the additional files should be located in the corresponding CMakeFiles directory for each compile target. For example, the optimization report for the VersionInfo module in OpenFAST are at:

```
>> ls -l openfast/build/modules/version/CMakeFiles/versioninfo.dir/src/
-rw-r--r-- 2740 May 12 23:10 VersionInfo.f90.o
-rw-r--r-- 0 May 12 23:10 VersionInfo.f90.o.provides.build
-rw-r--r-- 668 May 12 23:10 VersionInfo.f90.optrpt
```


Operator Strength Reduction

Each mathematical operation has an effective “strength”, and some operations can be equivalently represented as a combination of multiple reduced-strength operations that have better performance than the original. As part of the code optimization step, compilers may be able to identify areas where a mathematical operation’s strength can be reduced. Compilers are not able to optimize all expensive operations. For example, cases where a portion of an expensive mathematical operation is a variable contained in a derived data type are frequently skipped. Therefore, it is recommended that expensive subroutines be profiled and searched for possible strength reduction opportunities.

A concrete example of operator strength reduction is in dividing many elements of an array by a constant.

```
module_type%scale_factor = 10.0

do i = 1
  if array(i) < 30.0
    array(i) = array(i) / module_type%scale_factor
  end if
end do
```

In this case, a multiplication of real numbers is less expensive than a division of real numbers. The code can be factored so that the inverse of the scale factor is computed outside of the loop and the mathematical operation in the loop is converted to a multiplication.

```
module_type%scale_factor = 10.0
inverse_scale_factor = 1.0 / module_type%scale_factor

do i = 1
  if array(i) < 30.0
    array(i) = array(i) * inverse_scale_factor
  end if
end do
```

Coarrays

Coarrays are a feature introduced to the Fortran language in the 2008 standard to provide language-level parallelization for array operations in a Single Program, Multiple Data (SPMD) programming paradigm. The Fortran standard leaves the method of parallelization up to the compiler, and the Intel® Fortran compiler uses MPI.

Coarrays are used to split an array operation across multiple copies of the program. The copies are called “images”. Each image has its own local variables, plus a portion of any coarrays as shared variables. A coarray can be thought of as having extra dimensions, referred to as “codimensions”. A single image (typically the 1-th index) controls the I/O and problem setup, and images can read from memory in other images.

For operations on large arrays such as constructing a super-array from many sub-arrays (as in the construction of a Jacobian matrix), the coarray feature of Fortran 08 can parallelize the procedure improving overall computational efficiency.

Data modeling and access rules

Fortran represents arrays in column-major order. This means that a multidimensional array is represented in memory with column elements being adjacent. If a given element in an array is at a location in memory, one element before in memory corresponds to the element above it in its column.

In order to make use of the single instruction, multiple data features of modern processors, array construction and access should happen in column-major order. That is, loops should loop over the left-most index quickest. Slicing should occur with the `:` also on the left-most index when possible.

With this in mind, data should be represented as structures of arrays rather than arrays of structures. Concretely, this means that data types within OpenFAST should contain the underlying arrays and arrays should generally contain only numeric types.

The short program below displays the distance in memory in units of bytes between elements of an array and neighboring elements.

```

program memloc

implicit none

integer(kind=8), dimension(3, 3) :: r, distance_up, distance_left

! Take the element values as their "ID"
! r(row, column)
r(1,:) = (/ 1, 2, 3 /)
r(2,:) = (/ 4, 5, 6 /)
r(3,:) = (/ 7, 8, 9 /)
print *, "Reference array:"
call pretty_print_array(r)

! Compute the distance between matrix elements. Inputs to the `calculate_distance`
↳function
! are indices for elements in the equation loc(this_element) - loc(other_element)
distance_up(1,:) = (/ calculate_distance( 1,1 , 1,1), calculate_distance( 1,2 , 1,2),
↳calculate_distance( 1,3 , 1,3) /)
distance_up(2,:) = (/ calculate_distance( 2,1 , 1,1), calculate_distance( 2,2 , 1,2),
↳calculate_distance( 2,3 , 1,3) /)
distance_up(3,:) = (/ calculate_distance( 3,1 , 2,1), calculate_distance( 3,2 , 2,2),
↳calculate_distance( 3,3 , 2,3) /)
print *, "Distance in memory (bytes) for between an element and the one above it (top
↳row zeroed):"
call pretty_print_array(distance_up)

distance_left(1,:) = (/ calculate_distance( 1,1 , 1,1), calculate_distance( 1,2 , 1,1),
↳calculate_distance( 1,3 , 1,2) /)
distance_left(2,:) = (/ calculate_distance( 2,1 , 2,1), calculate_distance( 2,2 , 2,1),
↳calculate_distance( 2,3 , 2,2) /)
distance_left(3,:) = (/ calculate_distance( 3,1 , 3,1), calculate_distance( 3,2 , 3,1),
↳calculate_distance( 3,3 , 3,2) /)
print *, "Distance in memory (bytes) for between an element and the one to the its left
↳(first column zeroed):"
call pretty_print_array(distance_left)

```

(continues on next page)

(continued from previous page)

```
contains

integer(8) function calculate_distance(c1, r1, c2, r2)

    integer, intent(in) :: c1, r1, c2, r2
    calculate_distance = loc(r(c1, r1)) - loc(r(c2, r2))

end function

subroutine pretty_print_array(array)

    integer(8), dimension(3,3), intent(in) :: array
    print *, "["
    print '(I4, I4, I4)', array(1,1), array(1,2), array(1,3)
    print '(I4, I4, I4)', array(2,1), array(2,2), array(2,3)
    print '(I4, I4, I4)', array(3,1), array(3,2), array(3,3)
    print *, "]"

end subroutine

end program
```

Optimization Studies

This section describes specific efforts to profile sections of OpenFAST and improve performance with the Intel® compiler suite.

BeamDyn Performance Profiling and Optimization (IPCC Year 1 and 2)

The general mechanisms identified for performance improvements in OpenFAST were:

- Intel® compiler suite and Intel® Math Kernel Library (Intel® MKL)
- Algorithmic improvements
- Memory-access optimization enabling more efficient cache usage
- Data type alignment allowing for SIMD vectorization
- Multithreading with OpenMP

To establish a path forward with these options, OpenFAST was first profiled with Intel® VTune™ Amplifier to get a clear breakdown of time spent in the simulation. Then, the optimization report generated from the Intel® Fortran compiler was analyzed to determine areas that were not autovectorized. Finally, Intel® Advisor was used to highlight areas of the code that the compiler identified as potentially improved with multithreading.

Two OpenFAST test cases have been chosen to provide meaningful and realistic timing benchmarks. In addition to real-world turbine and atmospheric models, these cases are computationally expensive and expose the areas where performance improvements would make a difference.

5MW_Land_BD_DLL_WTurb

Download case files [here](#).

The physics modules used in this case are:

- BeamDyn
- InflowWind
- AeroDyn 15
- ServoDyn

This is a land based NREL 5-MW turbine simulation using BeamDyn as the structural module. It simulates 20 seconds with a time step size of 0.001 seconds and executes in **3m 55s** on NREL's [Peregrine](#) supercomputer.

5MW_OC4Jckt_DLL_WTurb_WavesIrr_MGrowth

Download case files [here](#).

This is an offshore, fixed-bottom NREL 5-MW turbine simulation with the majority of the computational expense occurring in the HydroDyn wave-dynamics calculation.

The physics modules used in this case are:

- ElastoDyn
- InflowWind
- AeroDyn 15
- ServoDyn
- HydroDyn
- SubDyn

It simulates 60 seconds with a time step size of 0.01 seconds and executes in **20m 27s** on NREL's [Peregrine](#) supercomputer.

Profiling

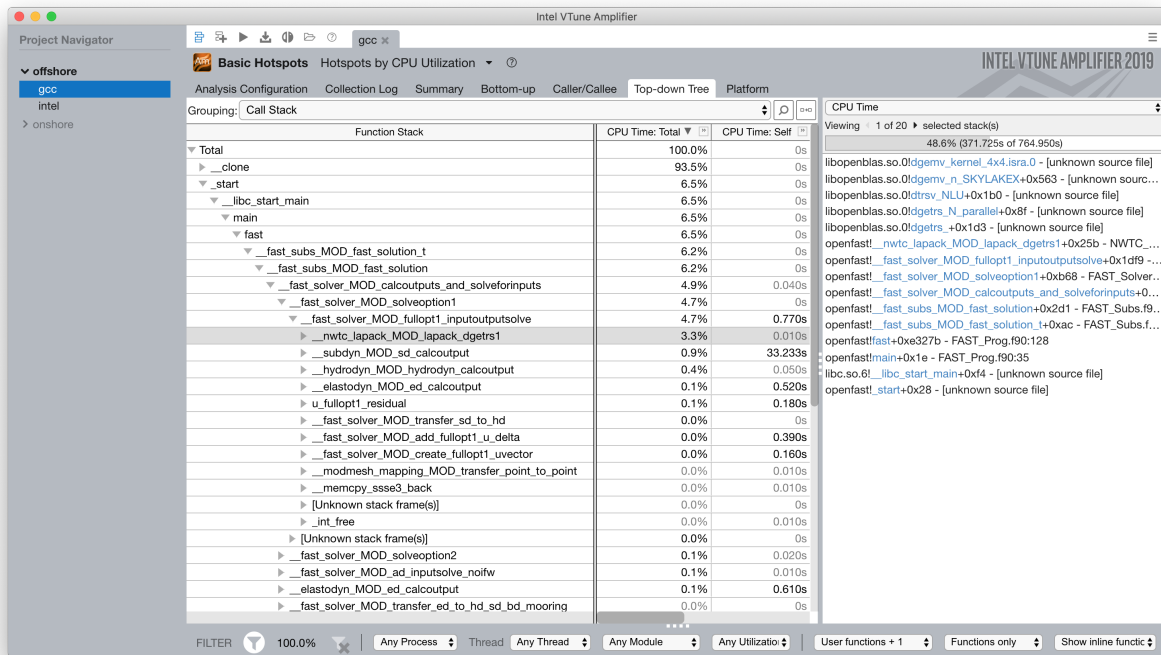
The OpenFAST test cases were profiled with Intel® VTune™ Amplifier to identify performance hotspots. Being that the two test cases exercise different portions of the OpenFAST software, different hotspots were identified. In all cases and environment settings, the majority of the CPU time was spent in `fast_solution` loop which is a high-level subroutine that coordinates the solution calculation from each physics module.

LAPACK

In the offshore case, the LAPACK usage was identified as a performance load. Within the `fast_solution` loop, the calls to the LAPACK function `dgetrs` consume 3.3% of the total CPU time.

BeamDyn

While BeamDyn provides a high-fidelity blade-response calculation, it is a computationally expensive module. Initial profiling highlighted the `bd_elementmatrixga2` subroutine as a hotspot. However, initial attempts to improve performance in BeamDyn revealed needs for algorithmic improvements and refinements to the module's data structures.



Results

Though work is ongoing, OpenFAST time-to-solution performance has improved and the performance potential is better understood.

Some keys outcomes from the first year of the IPCC project are as follows:

- Use of Intel® compiler and MKL library provides dramatic speedup over GCC and LAPACK
 - Additional significant gains are possible through MKL threading for offshore simulations
- Offshore-wind-turbine simulations are poorly load balanced across modules
 - Land-based-turbine configuration better balanced
 - OpenMP Tasks are employed to achieve better load-balancing
- OpenMP module-level parallelism provides significant, but limited speed up due to imbalance across different module tasks
- Core algorithms need significant modification to enable OpenMP and SIMD benefits

Tuning the Intel® tools to perform best on NREL's hardware and adding high level multithreading yielded a maximum 3.8x time-to-solution improvement for one of the benchmark cases.

Speedup - Intel® Compiler and MKL

By employing the standard Intel® developer tools tech stack, a performance improvement over GNU tools was demonstrated:

Com- piler	Math Library	5MW_Land_BD_DLL_WTurb	5MW_OC4Jckt_DLL_WTurb_WavesIrr_MGrowth
GNU	LAPACK	2265 s (1.0x)	673 s (1.0x)
Intel® 17	LAPACK	1650 s (1.4x)	251 s (2.7x)
Intel® 17	MKL	1235 s (1.8x)	—
Intel® 17	MKL Multi- threaded	722 s (3.1x)	—

Speedup - OpenMP at FAST_Solver

A performance improvement was demonstrated by adding OpenMP directives to the FAST_Solver module. Although the solution scheme is not well balanced, parallelizing mesh mapping and calculation routines resulted in the following speedup:

Com- piler	Math Library	5MW_Land_BD_DLL_WTurb	5MW_OC4Jckt_DLL_WTurb_WavesIrr_MGrowth
Intel® 17	MKL - 1 thread	1073 s (2.1x)	100 s (6.7x)
Intel® 17	MKL - 8 threads	597 s (3.8x)	—

Ongoing Work

The next phase of the OpenFAST performance improvements are focused in two key areas:





















1. Implementing the outcomes from previous work throughout OpenFAST modules and glue codes
2. Preparing OpenFAST for efficient execution on Intel®'s next generation platforms

Linearization routine profiling

As a portion of the [ARPA-E WEIS](#) project, the linearization capability within OpenFAST has been profiled in an effort to characterize the performance and current bottlenecks. This work specifically targetted the linearization routines within the FAST Library, primarily in [FAST_Line.f90](#), as well as the routines constructing the Jacobian matrices within individual physics modules. Because these routines require constructing large matrices, this is a computationally intensive process with a high rate of memory access.

A high-level flow of data in the linearization algorithm in the FAST_Linearize_OP subroutine is given below.

Each enabled physics module constructs module-level matrices in their respective <Module>_Jacobian and <Module>_GetOP routines, and the collection of these are assembled into global matrices in Glue_Jacobians and Glue_StateMatrices. In a top-down comparison of total CPU time in FAST_Linearize_OP, we see that the construction of the glue-code state matrices is the most expensive step. The HydroDyn Jacobian computation also stands out relative to other module Jacobian computations.

Function Call Sites and Loops	Location	CPU Time		
		Total Time %	Total Time	Total Elaps...
Total		100.0% 	211.170s	226.285s
[-] RtlUserThreadStart		100.0% 	211.170s	226.285s
[-] BaseThreadInitThunk		100.0% 	211.170s	226.285s
[-] _scrt_common_main_seh	exe_common.inl:236	100.0% 	211.170s	226.285s
[-] main		100.0% 	211.170s	226.285s
[-] FAST	FAST_Prog.f90:23	100.0% 	211.170s	225.967s
[-] [loop in FAST at FAST_Prog.f90:86]	FAST_Prog.f90:86	100.0% 	211.170s	225.967s
[-] FAST_SUBS_mp_FAST_LINEARIZE_T	FAST_Sub.f90:5857	100.0% 	211.129s	225.159s
[-] FAST_LINEAR_mp_FAST_LINEARIZE_OP	FAST_Lin.f90:563	100.0% 	211.129s	225.159s
[-] FAST_LINEAR_mp_GLUE_STATEMATRICES	FAST_Lin.f90:3519	93.8% 	198.110s	198.268s
[-] HYDRODYN_mp_HD_JACOBIANPINPUT	HydroDyn.f90:2431	5.2% 	11.041s	23.521s
[-] AERODYN_mp_AD_JACOBIANPINPUT	AeroDyn.f90:3287	0.6% 	1.191s	2.162s
[-] FAST_LINEAR_mp_GLUE_JACOBIANS	FAST_Lin.f90:1530	0.2% 	0.393s	0.393s
[-] ELASTODYN_mp_ED_JACOBIANPINPUT	ElastoDyn.f90:10481	0.2% 	0.333s	0.349s
[-] HYDRODYN_mp_HD_JACOBIANPCONTSTATE	HydroDyn.f90:2689	0.0% 	0.032s	0.222s
[-] ELASTODYN_mp_ED_JACOBIANPCONTSTATE	ElastoDyn.f90:10704	0.0% 	0.030s	0.030s
[-] FAST_LINEAR_mp_WRLINFILE_TXT_HEAD	FAST_Lin.f90:1167	0.0% 	0s	0.015s
[-] FAST_LINEAR_mp_WRLINFILE_TXT_END	FAST_Lin.f90:1313	0.0% 	0s	0.002s
[-] FAST_SUBS_mp_FAST_INITIALIZEALL_T	FAST_Sub.f90:34	0.0% 	0.027s	0.794s
[-] FAST_SUBS_mp_FAST_SOLUTION0_T	FAST_Sub.f90:3736	0.0% 	0.014s	0.014s

The Jacobian and state matrices are sized based on the total number of inputs, outputs, and continuous states. Though the size varies, these matrices generally contain thousands of elements in each dimension and are mostly zeroes. That is to say, the Jacobian and state matrices are large and sparse. To reduce the overhead of memory allocation and access, a sparse matrix representation is recommended.

Versioning

OpenFAST follows [semantic versioning](#). In summary, this means that with a version number as MAJOR.MINOR.PATCH, the components will be incremented as follows:

- MAJOR version when introducing incompatible API changes,
- MINOR version when adding functionality in a backwards-compatible manner, and
- PATCH version when making backwards-compatible bug fixes.

For example, OpenFAST-v1.0.0-123-gabcd1234-dirty describes OpenFAST as:

Version Component	Explanation
v1.0.0	MAJOR.MINOR.PATCH numbering system; corresponds to a tagged commit made by NREL on GitHub
123-g	Number of additional commits after the most recent tag for a build (the -g is for git)
abcd1234	First 8 characters of the current commit hash
dirty	Denotes that local changes have been made but not committed; omitted if there are no local changes

5.2 API Reference

Some subroutines and derived types throughout the source code have in-source documentation which is compiled with Doxygen. Though this portion of the documentation is always under development, the existing API reference can be found in the following pages:

- [Main Page](#)
- [Index of Types](#)
- [Source Files](#)

5.3 Other Documentation

Additional documentation exists that may be useful for developers seeking deeper understanding of the solver and mathematics.

- **NWTC Programmer's Handbook**

This is an overview of programming guidelines for FAST 8. While some syntax and minor details have changed in OpenFAST, most of this guide is still relevant.

- **OutListParameters.xlsx**

This Excel file contains the full list of outputs for each module. It is used to generate the Fortran code for the output channel list handling for each module (this code is generally in the `_IO.f90` files). The MATLAB script available in the [matlab-toolbox](#) repository at *Utilities/GetOutListParameters.m*.

LICENSING

The OpenFAST software, including its underlying modules, are licensed under [Apache License Version 2.0](#) open-source license.

GETTING HELP

For possible bugs, enhancement requests, or code questions, please submit an issue at the [OpenFAST Github repository](#).

For OpenFAST usage questions, users should consider the [FAST Forum](#), which provides a large 10+ year legacy of FAST-related Q&A; the forum's search functionality should be used before posting questions to either github issues or the forum.

Users may find the established FAST v8 through the NWTC Information Portal: <https://nwtc.nrel.gov/>

Please contact Michael.A.Sprague@NREL.gov with questions regarding the OpenFAST development plan or how to contribute.

ACKNOWLEDGEMENTS

This software is developed and maintained by researchers at the [National Renewable Energy Laboratory](#) with funding from U.S. Department of Energy Wind Energy Technology Office through the [Atmosphere to electrons \(A2e\)](#) research initiative.

NREL gratefully acknowledges development contributions from the following organizations:

- Envision Energy USA, Ltd
- Brigham Young University

NREL gratefully acknowledges additional development support through designation as an [Intel® Parallel Computing Center \(IPCC\)](#).

BIBLIOGRAPHY

- [ad-Bra17] E. Branlard. *Wind Turbine Aerodynamics and Vorticity-Based Methods: Fundamentals and Recent Applications*. Springer International Publishing, 2017. ISBN 978-3-319-55163-0. doi:10.1007/978-3-319-55164-7.
- [ad-DH19] R. Damiani and G. Hayman. The unsteady aerodynamics module for fast 8. Technical Report, National Renewable Energy Laboratory, 2019. NREL/TP-5000-66347.
- [ad-HGAM04] M.H. Hansen, Mac Gaunaa, and Helge Aagaard Madsen. A beddoes-leishman type dynamic stall model in state-space and indicial formulations. Technical Report, Risø National Laboratory, Roskilde, Denmark, 2004.
- [ad-LB89] J. G. Leishman and T.S. Beddoes. A semi-empirical model for dynamic stall. *Journal of the American Helicopter Society*, 34(3):p3–17, 1989.
- [ad-MH05] P.J. Moriarty and A. Craig Hansen. Aerodyn theory manual. Technical Report, National Renewable Energy Laboratory, December 2005. NREL/EL-500-36881.
- [ad-MB11] J. Murray and M. Barone. The development of cactus : a wind and marine turbine performance simulation code. Technical Report, 49th AIAA Aerospace Sciences Meeting, Orlando, Florida, 2011.
- [ad-Nin14] S. Andrew Ning. A simple solution method for the blade element momentum equations with guaranteed convergence. *Wind Energy*, 17(9):1327–1345, 2014. doi:https://doi.org/10.1002/we.1636.
- [ad-Oye91] S. Øye. Dynamic stall, simulated as a time lag of separation. *Proceedings of the 4th IEA Symposium on the Aerodynamics of Wind Turbines*, 1991.
- [olaf-Abe16] H. Abedi. *Development of Vortex Filament Method for Wind Power Aerodynamics*. PhD thesis, Chalmers University of Technology, Gothenburg, Sweden, 2016.
- [olaf-ALR02] S. Ananthan, J. G. Leishman, and M. Ramasamy. The role of filament stretching in the free-vortex modeling of rotor wakes. In *58th Annual Forum and Technology Display of the American Helicopter Society International*. Montreal, Canada, 2002.
- [olaf-BL93] A. Bagai and J. G. Leishman. Flow visualization of compressible vortex structures using density gradient techniques. *Experiments in Fluids*, 15(6):431–442, 1993.
- [olaf-BL94] A. Bagai and J. G. Leishman. Rotor free-wake modeling using a pseudo-implicit technique including comparisons with experimental data. In *50th Annual Forum of the American Helicopter Society*. Washington, D.C., 1994.
- [olaf-Bra17] E. Branlard. *Wind Turbine Aerodynamics and Vorticity-Based Methods: Fundamentals and Recent Applications*. Springer International Publishing, 2017. ISBN 978-3-319-55163-0. doi:10.1007/978-3-319-55164-7.

- [olaf-BPG+15] E. Branlard, G. Papadakis, M. Gaunaa, G. Winckelmans, and T. J. Larsen. Aeroelastic large eddy simulations using vortex methods: unfrozen turbulent and sheared inflow. *Journal of Physics: Conference Series (Online)*, 2015. doi:10.1088/1742-6596/625/1/012019.
- [olaf-Gup06] S. Gupta. *Development of a Time-Accurate Viscous Lagrangian Vortex Wake Model for Wind Turbine Applications*. PhD thesis, Univeristy of Maryland, College Park, MD, 2006.
- [olaf-GL02] S. Gupta and J. G. Leishman. Free-vortex filament methods for the analysis of helicopter rotor wakes. *Journal of Aircraft*, 39(5):759–775, 2002.
- [olaf-Han08] M. O. L. Hansen. *Aerodynamics of Wind Turbines*. Earthscan, London; Sterling, VA, 2008.
- [olaf-Jon13] J. Jonkman. The new modularization framework for the fast wind turbine cae tool. Technical report NREL/CP-5000-57228, National Renewable Energy Laboratory, 2013.
- [olaf-Ker00] J. Kerwin. Lecture notes hydrofoil and propellers. Technical Report, M.I.T., 2000.
- [olaf-Lei06] J. Leishman. *Principles of Helicopter Aerodynamics*. Cambridge Univ. Press, Cambridge, MA, 2006.
- [olaf-LBB02] J. G. Leishman, M. J. Bhagwat, and A. Bagai. Free-vortex filament methods for the analysis of helicopter rotor wakes. *Journal of Aircraft*, 39(5):759–775, 2002.
- [olaf-Pap14] G. Papadakis. *Development of a hybrid compressible vortex particle method and application to external problems including helicopter flows*. PhD thesis, National Technical University of Athens, 2014.
- [olaf-Ran58] W. J. M. Rankine. *Manual of Applied Mechanics*. Griffen Co., London, 1858.
- [olaf-Rib07] M. Ribera. *Helicopter Flight Dynamics Simulation with a Time-Accurate Free-Vortex Wake Model*. PhD thesis, University of Maryland, College Park, MD, 2007.
- [olaf-Ros31] L. Rosenhead. The formation of vortices from a surface of discontinuity. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 134(823):170–192, 1931. URL: <http://www.jstor.org/stable/95835>.
- [olaf-Scu75] M. P. Scully. *Computation of Helicopter Rotor Wake Geometry and Its Influence on Rotor Harmonic Airloads*. PhD thesis, Massachusetts Institute of Technology, Cambridga, MA, 1975.
- [olaf-SGarciaSorensenS17] M. Sessarego, N. Ramos García, J. N. Sørensen, and W. Z. Shen. Development of an aeroelastic code based on three-dimensional viscous-inviscid method for wind turbine computations. *Wind Energy*, 20(7):1145–1170, 2017. doi:10.1002/we.2085.
- [olaf-SJJ15] Michael A. Sprague, Jason M. Jonkman, and Bonnie J. Jonkman. Fast modular framework for wind turbine simulation: new algorithms and numerical examples. Technical Report NREL/CP-2C00-63203, National Renewable Energy Laboratory, 2015.
- [olaf-vG03] A. van Garrel. Development of a wind turbine aerodynamics simulation module. Technical Report ECN-C-03-079, ECN, 2003.
- [olaf-VKM91] G. H. Vatistas, V. Koezel, and W. C. Mih. A simpler model for concentrated vortices. *Experiments in Fluids*, 11(1):73–76, 1991.
- [olaf-Vou06] S. G. Voutsinas. Vortex methods in aeronautics: how to make things work. *International Journal of Computational Fluid Dynamics*, 2006.
- [olaf-Wei47] J. Weissinger. The lift distribution of swept-back wings. Technical report TM 1120, NACA, 1947.
- [olaf-WL93] G. S. Winckelmans and A. Leonard. Contributions to vortex particle methods for the computation of 3-dimensional incompressible unsteady flows. *Journal Of Computational Physics*, 109(2):247–273, 1993.
- [aa-Ami75] Roy K. Amiet. Acoustic radiation from an airfoil in a turbulent stream. *Journal of Sound and Vibration*, 41(4):407–420, 1975. doi:10.1016/S0022-460X(75)80105-2.

- [aa-BTD+19] Pietro Bortolotti, Helena Canet Tarres, Katherine Dykes, Karl Merz, Latha Sethuraman, David Verelst, and Frederik Zahle. Systems engineering in wind energy - wp2.1 reference wind turbines. Technical Report, IEA Technical Report, 2019. URL: <https://www.nrel.gov/docs/fy19osti/73492.pdf>.
- [aa-BPM89] Thomas F. Brooks, D. Stuart Pope, and Michael A. Marcolini. Airfoil self-noise and prediction. Reference Publication 1218, NASA, 1989.
- [aa-DG87] Mark Drela and Michael B. Giles. Viscous-inviscid analysis of transonic and low reynolds number airfoils. *AIAA Journal*, 25(10):1347–1355, 1987. doi:10.2514/3.9789.
- [aa-GBW+97] Gianfranco Guidati, Rainer Bareiss, Siegfried Wagner, Rene Parchen, Gianfranco Guidati, Rainer Bareiss, Siegfried Wagner, and Rene Parchen. Simulation and measurement of inflow-turbulence noise on airfoils. In *3rd AIAA/CEAS Aeroacoustics Conference*. 1997. doi:10.2514/6.1997-1698.
- [aa-KGW+18] Levin Klein, Jonas Gude, Florian Wenz, Thorsten Lutz, and Ewald Krämer. Advanced computational fluid dynamics (cfd)–multi-body simulation (mbs) coupling to assess low-frequency emissions from wind turbines. *Wind Energy Science Journal*, 3:713–728, 2018. doi:10.5194/wes-3-713-2018.
- [aa-Low70] Martin V. Lowson. Theoretical analysis of compressor noise evaluation. *The Journal of the Acoustical Society of America*, 47:371–385, 1970. doi:10.1121/1.1911508.
- [aa-MGM04] Patrick Moriarty, Gianfranco Guidati, and Paul Migliore. Recent improvement of a semi-empirical aeroacoustic prediction code for wind turbines. In *10th AIAA/CEAS Aeroacoustics Conference*. 2004. doi:10.2514/6.2004-3041.
- [aa-MGM05] Patrick Moriarty, Gianfranco Guidati, and Paul Migliore. Prediction of turbulent inflow and trailing-edge noise for wind turbines. In *11th AIAA/CEAS Aeroacoustics Conference*. 2005. doi:10.2514/6.2005-2881.
- [aa-Mor05] Patrick J. Moriarty. Nafnoise user's guide. Technical Report, National Renewable Energy Laboratory, Golden, CO, 2005. URL: <https://github.com/NREL/NAFNoise/blob/master/NAFNoise.pdf>.
- [aa-MH05] Patrick J. Moriarty and A. C. Hansen. Aerodyn theory manual. Technical Report NREL/TP-500-36881, National Renewable Energy Laboratory, Golden, CO, 2005. URL: <https://www.nrel.gov/docs/fy05osti/36881.pdf>.
- [aa-MM03] Patrick J. Moriarty and Paul G. Migliore. Semi-empirical aeroacoustic noise prediction code for wind turbines. Technical Report NREL/TP-500-34478, National Renewable Energy Laboratory, Golden, CO, 2003. URL: <https://www.nrel.gov/docs/fy04osti/34478.pdf>.
- [aa-Par98] René R. Parchen. Progress report DRAW: a prediction scheme for trailing edge noise based on detailed boundary layer characteristics. Technical Report, TNO Institute of Applied Physics, 1998.
- [aa-PA76] R. Paterson and R. Amiet. Acoustic radiation and surface pressure characteristics of an airfoil due to incident turbulence. In *3rd Aeroacoustics Conference*. AIAA, 1976. doi:10.2514/6.1976-571.
- [aa-SBCB18] CR Sucameli, P Bortolotti, A Croce, and CL Bottasso. Comparison of some wind turbine noise emission models coupled to BEM aerodynamics. *Journal of Physics: Conference Series*, 1037:022038, jun 2018. doi:10.1088/1742-6596/1037/2/022038.
- [aa-Vit81] Larry A. Viterna. Method for predicting impulsive noise generated by wind turbine rotors. Technical Report DOE/NASA/20320-36, 1981. URL: <https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19820013840.pdf>.
- [aa-ZHS05] Wei J. Zhu, Nicolai Heilskov, and Wen Zhong Shen. Modeling of aerodynamically generated noise from wind turbines. *Journal of Solar Energy Engineering*, 127(4):517–528, 2005. doi:10.1115/1.2035700.
- [BC80] K. J. Bathe and A. P. Cimento. Some practical procedures for the solution of nonlinear finite element equations. *Computer Methods in Applied Mechanics and Engineering*, 22:59–85, 1980. http://web.mit.edu/kjb/www/Publications_Prior_to_1998/Some_Practical_Procedures_for_the_Solution_of_Nonlinear_Finite_Element_Equations.pdf. doi:10.1016/0045-7825(80)90051-1.
- [Bau10] O. A. Bauchau. *Flexible Multibody Dynamics*. Springer, 2010. doi:10.1007/978-94-007-0335-3.

- [BEH08] O.A. Bauchau, A. Epple, and S.D. Heo. Interpolation of finite rotations in flexible multibody dynamics simulations. *Proceedings of the Institution of Mechanical Engineers, Part K: Journal of Multi-body Dynamics*, 222:353–366, 2008.
- [CH93] J. Chung and G. M. Hulbert. A time integration algorithm for structural dynamics with improved numerical dissipation: the generalized- method. *Journal of Applied Mechanics*, 60:371–375, 1993. doi:10.1115/1.2900803.
- [GSJ13] A. Gasmi, M.A. Sprague, and J.M. Jonkman. Numerical stability and accuracy of temporally coupled multi physics modules in wind-turbine cae tools. In *Proceedings of the 51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*. Grapevine, Texas, January 2013.
- [Hod06] Dewey H. Hodges. *Nonlinear Composite Beam Theory*. AIAA, 2006.
- [JelenicC99] G. Jelenić and M. A. Crisfield. Geometrically exact 3d beam theory: implementation of a strain-invariant finite element for statics and dynamics. *Computer Methods in Applied Mechanics and Engineering*, 171:141–171, 1999.
- [Jon13] J.M. Jonkman. The new modularization framework for the fast wind turbine cae tool. In *Proceedings of the 51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*. Grapevine, Texas, January 2013.
- [JJ13] Jason Jonkman and Bonnie Jonkman. Fast v8. <https://nwtc.nrel.gov/FAST8>, October 2013. [Online; accessed 29-OCTOBER-2014].
- [Pat84] A. T. Patera. A spectral element method for fluid dynamics: laminar flow in a channel expansion. *Journal of Computational Physics*, 54:468–488, 1984.
- [RP87] E. M. Ronquist and A. T. Patera. A legendre spectral element method for the stefan problem. *International Journal for Numerical Methods in Engineering*, 24:2273–2299, 1987.
- [SG03] M. A. Sprague and T. L. Geers. Spectral elements and field separation for an acoustic fluid subject to cavitation. *Journal of Computational Physics*, 184:149–162, 2003.
- [SG04] M. A. Sprague and T. L. Geers. A spectral-element method for modeling cavitation in transient fluid-structure interaction. *International Journal for Numerical Methods in Engineering*, 60:2467–2499, 2004.
- [SJJ14] M.A. Sprague, J.M. Jonkman, and B.J. Jonkman. Fast modular wind turbine cae tool: non matching spatial and temporal meshes. In *Proceedings of the 32nd ASME Wind Energy Symposium*. National Harbor, Maryland, January 2014.
- [WJSJ15] Q. Wang, N. Johnson, M.A. Sprague, and J. Jonkman. Beamdyn: a high-fidelity wind turbine blade solver in the fast modular framework. In *Proceedings of the 33rd ASME Wind Energy Symposium*. Kissimmee, Florida, January 2015. <https://www.nrel.gov/docs/fy15osti/63165.pdf>.
- [WS13] Q. Wang and M.A. Sprague. A legendre spectral finite element implementation of geometrically exact beam theory. In *Proceedings of the 54th Structures, Structural Dynamics, and Materials Conference*. Boston, Massachusetts, April 2013.
- [WSJJ14] Q. Wang, M.A. Sprague, J. Jonkman, and N. Johnson. Nonlinear legendre spectral finite elements for wind turbine blade dynamics. In *Proceedings of the 32nd ASME Wind Energy Symposium*. National Harbor, Maryland, January 2014.
- [WSJJ16] Q. Wang, M.A. Sprague, J. Jonkman, and B. Jonkman. Partitioned nonlinear structural analysis of wind turbines using beamdyn. In *Proceedings of the 34th ASME Wind Energy Symposium*. San Diego, California, January 2016.
- [WYS13] Q. Wang, W. Yu, and M.A. Sprague. Gemoetrically nonlinear analysis of composite beams using wiener-milenković parameters. In *Proceedings of the 54th Structures, Structural Dynamics, and Materials Conference*. Boston, Massachusetts, April 2013.

- [CC10] S.C. Chapra and R.P. Canale. *Numerical Methods for Engineers*. McGraw-Hill Science/Engineering/Mat, 2010.
- [Coo01] Robert D. Cook. *Concepts and Applications of Finite Element Analysis*. John Wiley & Sons, 2001.
- [CB68] R. Craig and M. Bampton. Coupling of substructures for dynamic analyses. *AIAA Journal*, 6:1313 – 1319, 1968.
- [DJH15] R. Damiani, J. Jonkman, and G. Hayman. Subdyn user's guide and theory manual. Technical Report NREL/TP-5000-63062, National Renewable Energy Laboratory, 2015.
- [DS13] R. Damiani and H. Song. A jacket sizing tool for offshore wind turbines within the systems engineering initiative. *Offshore Technology Conference*, 2013.
- [DSRJ13] R. Damiani, H. Song, A. Robertson, and J. Jonkman. Assessing the importance of nonlinear structural characteristics in the development of a jacket model for the wind turbine cae tool fast. *32nd International Conference on Ocean, Offshore and Arctic Engineering*, 2013.
- [Fel04] Carlos A. Felippa. Introduction to finite element methods - lecture notes (asen 5007). Technical Report, Department of Aerospace Engineering Sciences and Center for Aerospace Structures, University of Colorado, Boulder, CO, USA, 2004.
- [Hur64] W.C. Hurty. On the dynamic analysis of structural systems using component modes. In *Proceedings of the 51st AIAA Annual Meeting*. Washington, DC, June 29–July 2 1964. AIAA Paper No. 64-487.
- [JJo13] JJonkman. The new modularization framework for the FAST wind turbine CAE tool. In *Proceedings of the 51st AIAA Aerospace Sciences Meeting and 31st ASME Wind Energy Symposium*. Grapevine, Texas, January 7 – 10 2013.
- [JBH+20] Jason Jonkman, Emmanuel Branlard, Matthew Hall, Greg Hayman, Andy Platt, and Amy Robertson. Implementation of substructure flexibility and member-level load capabilities for floating offshore wind turbines in openfast. Technical Report NREL/TP-5000-76822, National Renewable Energy Laboratory, 2020.
- [MJJ14] MSprague, J.M. Jonkman, and B.J. Jonkman. FAST modular wind turbine CAE tool: nonmatching spatial and temporal meshes. In *Proceedings of the 32nd ASME Wind Energy Symposium*. National Harbor, Maryland, January 2014.
- [PHEL09] H. Panzer, J. Hubele, R. Eid, and B. Lohmann. Generating a parametric finite element model of a 3d cantilever timoshenko beam using matlab. Technical Report, Technische Universitat Munchen, 2009. Technical Reports on Automatic Control.
- [SDRJ13] H. Song, R. Damiani, A. Robertson, and J. Jonkman. (2013). new structural-dynamics module for offshore multimember substructures within the wind turbine computer-aided engineering tool FAST. In *Proceedings of the 23rd International Ocean, Offshore and Polar Engineering Conference - ISOPE 2013*. Anchorage, Alaska, June 30 – July 5 2013. <https://www.nrel.gov/docs/fy15osti/63165.pdf>.
- [SKM13] A. Steinboeck, A. Kugi, and H Mang. Energy-consistent shear coefficients for beams with circular cross sections and radially in homogeneous materials. *International Journal of Solids and Structures*, 50(11–12):1859–1868, 2013.
- [API14] API. Planning, designing and constructing fixed offshore platforms - working stress design. Technical Report API Recommended Practice 2A-WSD, 22nd Edition, American Petroleum Institute, 2014.
- [ISO07] ISO. 19902:2007 - petroleum and natural gas industries – fixed steel offshore structures. Technical Report, ISO, Geneva, Switzerland, 2007.
- [ep-BSA+20] E. Branlard, M. Shields, B. Anderson, R. Damiani, F. Wendt, J. Jonkman, W. Musial, and B. Foley. Superelement reduction of substructures for sequential load calculations in OpenFAST. *Journal of Physics: Conference Series*, 1452:012033, jan 2020. URL: <https://doi.org/10.1088/1742-6596/1452/1/012033>, doi:10.1088/1742-6596/1452/1/012033.

- [ep-CB68] R. Craig and M. Bampton. Coupling of Substructures for Dynamic Analysis. *AIAA Journal*, 6(7):1313–1319, 1968.
- [ep-Guy65] R. Guyan. Reduction of stiffness and mass matrices. *AIAA Journal*, 3:380, 1965.
- [LN06] C. H. Lee and J. N. Newman. *WAMIT User Manual*. WAMIT, Inc, Chestnut Hill, MA, 6.3, 6.3pc, 6.3s, 6.3s-pc edition, 2006.
- [RGSW87] W. J. Breeding R. G. Standing and D. Wilson. *Recent Developments in the Analysis of Wave Drift Forces, Low-Frequency Damping and Response*. Proceedings of the 79th Annual OTC, 1987.
- [SD81] N. Sharma and R. Dean. Second-order directional seas and associated wave forces. *Society of Petroleum Engineers Journal*, 4:129–140, 1981.
- [stc-LR11a] Matthew A. Lackner and Mario A. Rotea. Passive structural control of offshore wind turbines. *Wind energy*, 14(3):373–388, 2011. URL: <http://onlinelibrary.wiley.com/doi/10.1002/we.426/full>.
- [stc-LR11b] Matthew A. Lackner and Mario A. Rotea. Structural control of floating wind turbines. *Mechatronics*, 21(4):704–719, 2011. URL: <http://www.sciencedirect.com/science/article/pii/S0957415810002072>.
- [stc-NRL13] Hazim Namik, M. A. Rotea, and Matthew Lackner. Active structural control with actuator dynamics on a floating wind turbine. In *Proceedings of the 51st AIAA Aerospace Sciences Meeting*, 7–10. 2013. URL: <http://arc.aiaa.org/doi/pdf/10.2514/6.2013-455>.
- [stc-SL13] G. Stewart and M. A. Lackner. Optimization of a passive tuned mass damper for reducing loads in offshore wind turbines. *IEEE Transactions on Control Systems Technology*, 21(4):1090–1104, 2013.
- [stc-SL11] Gordon M. Stewart and Matthew A. Lackner. The effect of actuator dynamics on active structural control of offshore wind turbines. *Engineering Structures*, 33(5):1807–1816, 2011. URL: <http://www.sciencedirect.com/science/article/pii/S0141029611000915>.
- [stc-SL14] Gordon M. Stewart and Matthew A. Lackner. The impact of passive tuned mass dampers and wind-wave misalignment on offshore wind turbine loads. *Engineering Structures*, 73:54–61, 2014. URL: <http://www.sciencedirect.com/science/article/pii/S0141029614002673>.
- [ff-Ain88] J. F. Ainslie. Calculating the flowfield in the wake of wind turbines. *Journal of Wind Engineering and Industrial Aerodynamics*, 27:213–224, 1988. doi:[https://doi.org/10.1016/0167-6105\(88\)90037-2](https://doi.org/10.1016/0167-6105(88)90037-2).
- [ff-Ceal15] M. J. Churchfield and et al. A comparison of the dynamic wake meandering model, large-eddy simulations, and field data at the egmond aan zee offshore wind plant. In 33rd Wind Energy Symposium. Kissimmee, FL, 2015. AIAA. doi:<http://dx.doi.org/10.2514/6.2015-0724>.
- [ff-CN96] J. Crank and P. Nicolson. A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. *Advances in Computational Mathematics*, 6:207–226, 1996.
- [ff-Deal18] P. Doubrawa and et al. Optimization-based calibration of fast.farm parameters against sowfa. In 36th Wind Energy Symposium. Kissimmee, FL, January 2018. AIAA. doi:<https://arc.aiaa.org/doi/pdf/10.2514/6.2018-0512>.
- [ff-Geal16] P. M. O. Gebraad and et al. Wind plant power optimization through yaw control using a parametric model for wake effects – a cfd simulation study. *Wind Energy*, 19(1):95–114, 2016. doi:<http://onlinelibrary.wiley.com/doi/10.1002/we.1822/epdf>.
- [ff-Hao16] Y. Hao. *Wind Farm Wake Modeling and Analysis of Wake Impacts in a Wind Farm*. Phd thesis, University of Massachusetts, Amherst, Massachusetts, 2016.
- [ff-Heal14] Y. Hao and et al. Implementing the dynamic wake meandering model in the nwtc design codes. In 32nd Wind Energy Symposium. National Harbor, MD, January 2014. AIAA. doi:<http://dx.doi.org/10.2514/6.2014-1089>.
- [ff-Jon14] B. Jonkman. Turbsim user's guide v2.00.00. Technical Report NREL/TP-xxxx-xxxxx, National Renewable Energy Laboratory, Golden, CO, October 2014.

- [ff-Jon13] J. Jonkman. The new modularization framework for the fast wind turbine cae tool. In 51st AIAA Aerospace Sciences Meeting. Dallas, TX, 2013. AIAA.
- [ff-Jeal09] J. Jonkman and et al. Definition of a 5-mw reference wind turbine for offshore system development. Technical Report NREL/TP-500-38060, National Renewable Energy Laboratory, Golden, CO, February 2009.
- [ff-Katiceal86] I. Kati\`c and et al. A simple model for cluster efficiency. In European Wind Energy Association Conference and Exhibition. Rome, Italy, 1986.
- [ff-Keal13] R.-E. Keck and et al. *A Consistent Turbulence Formulation for the Dynamic Wake Meandering Model in the Atmospheric Boundary Layer*. Phd thesis, DTU, Denmark, 2013.
- [ff-Leal08] G. C. Larsen and et al. Wake meander: a pragmatic approach. *Wind Energy*, 11:337–95, 2008. doi:<http://onlinelibrary.wiley.com/doi/10.1002/we.267/epdf>.
- [ff-Meal10] H. A. Madsen and et al. Calibration and validation of the dynamic wake meandering model for implementation in an aeroelastic code. *Journal of Solar Energy Engineering*, November 2010. doi:<https://doi.org/10.1115/1.4002555>.
- [ff-Meal16] H. A. Madsen and et al. Wake flow characteristics at high wind speed. In 34th Wind Energy Symposium. San Diego, CA, 2016. AIAA. doi:<http://dx.doi.org/10.2514/6.2016-1522>.
- [ff-MT21] L. A. Martinez-Tossas. Wind turbine wakes: high-thrust coefficient. *Wind Energy*, 2021. Publication pending.
- [ff-Seal19a] K. Shaler and et al. Effects of inflow spatiotemporal discretization on wake meandering and turbine structural response using fast.farm. *Journal of Physics: Conference Series*, May 2019. doi:[10.1088/1742-6596/1256/1/012023](https://doi.org/10.1088/1742-6596/1256/1/012023).
- [ff-Seal19b] K. Shaler and et al. Fast.farm response of varying wind inflow techniques. In 37th Wind Energy Symposium. San Diego, CA, 2019. AIAA. doi:<https://arc.aiaa.org/doi/pdf/10.2514/6.2019-2086>.
- [ff-Smi06] S. W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. Californial Technical Publishing, 2006. ISBN 978-0966017632.
- [ff-Seal14] M. A. Sprague and et al. Fast modular wind turbine cae tool: nonmatching spatial and temporal meshes. In 50th AIAA Aerospace Sciences Meeting. National Harbor, MD, January 2014. AIAA. doi:<http://arc.aiaa.org/doi/pdf/10.2514/6.2014-0520>.
- [ff-Seal15] M. A. Sprague and et al. Fast modular framework for wind turbine simulation: new algorithms and numerical examples. In 51th AIAA Aerospace Sciences Meeting. Kissimmee, FL, 2015. AIAA. doi:<http://arc.aiaa.org/doi/pdf/10.2514/6.2014-0520>.
- [ff-Tho49] L. H. Thomas. Elliptic problems in linear difference equations over a network. Technical Report, Watson Science Computer Laboratory, New York, NY, 1949.

INDEX

D

debug
 input file parameter, 334
dryRun
 input file parameter, 334
dtFAST
 input file parameter, 334

F

FAST_input_filename
 input file parameter, 335

I

input file parameter
 debug, 334
 dryRun, 334
 dtFAST, 334
 FAST_input_filename, 335
 nEveryCheckPoint, 334
 nTurbinesGlob, 334
 num_force_pts_blade, 335
 num_force_pts_tower, 335
 restart_filename, 335
 simStart, 334
 tEnd, 334
 tMax, 334
 tStart, 334
 turb_id, 335
 turbine_base_pos, 335

N

nEveryCheckPoint
 input file parameter, 334
nTurbinesGlob
 input file parameter, 334
num_force_pts_blade
 input file parameter, 335
num_force_pts_tower
 input file parameter, 335

R

restart_filename

input file parameter, 335

S

simStart
 input file parameter, 334

T

tEnd
 input file parameter, 334
tMax
 input file parameter, 334
tStart
 input file parameter, 334
turb_id
 input file parameter, 335
turbine_base_pos
 input file parameter, 335