Algorithms in OpenFAST v2

Bonnie Jonkman

April 17, 2020

1 Definitions and Nomenclature

Module Name	**Abbreviation in Module**	**Abbreviation in this Document**
ElastoDyn	ED	ED
BeamDyn	BD	BD
AeroDyn14	AD14	AD14
AeroDyn	AD	AD
ServoDyn	SrvD	SrvD
SubDyn	SD	SD
HydroDyn	HydroDyn	HD
MAP++	MAPp	MAP
FEAMooring	FEAM	FEAM
MoorDyn	MD	MD
OrcaFlexInterface	Orca	Orca
InflowWind	IfW	IfW
IceFloe	IceFloe	IceF
IceDyn	IceD	IceD

Table 1: Abbreviations for modules in FAST v8

2 Initializations

3 Input-Output Relationships

3.1 Input-Output Solves (Option 2 Before 1)

This algorithm documents the procedure for the Input-Output solves in FAST, assuming all modules are in use. If an individual module is not in use during a particular simulation, the calls to that module's subroutines are omitted and the module's inputs and outputs are neither set nor used.

```
procedure CalcOutputs_And_SolveForInputs()
    y_ED ← ED_CalcOutput(p_ED, u_ED, x_ED, xd_ED, z_ED)

    u_AD(not InflowWind) ← TransferOutputsToInputs(y_ED)
    u_IfW ← TransferOutputsToInputs(y_EDatu_ADnodes)
    y_IfW ← IfW_CalcOutput(u_IfWandotherIfWdatastructures)
    u_AD(InflowWind only) ← TransferOutputsToInputs(y_IfW)
    y_AD ← AD_CalcOutput(p_AD, u_AD, x_AD, xd_AD, z_AD)

    u_SrvD ← TransferOutputsToInputs(y_ED, y_IfW)
    y_SrvD ← SrvD_CalcOutput(p_SrvD, u_SrvD,
                                  x_SrvD, xd_SrvD, z_SrvD)

    u_ED(not platform reference point) ← TransferOutputsToInputs(y_SrvD, y_AD)
    u_HD ← TransferMeshMotions(y_ED)
    u_SD ← TransferMeshMotions(y_ED)
    u_MAP ← TransferMeshMotions(y_ED)
    u_FEAM ← TransferMeshMotions(y_ED)

    ED_HD_SD_Mooring_Ice_InputOutputSolve()

    u_AD ← TransferOutputsToInputs(y_ED)
    u_SrvD ← TransferOutputsToInputs(y_ED, y_AD)
end procedure
```

Note that inputs to *ElastoDyn* before calling CalcOutput() in the first step are not set in CalcOutputs_And_SolveForInputs(). Instead, the *ElastoDyn* inputs are set depending on where CalcOutputs_And_SolveForInputs() is called:

- At time 0, the inputs are the initial guess from *ElastoDyn*;
- On the prediction step, the inputs are extrapolated values from the time history of ElastoDyn inputs;
- On the first correction step, the inputs are the values calculated in the prediction step;
- On subsequent correction steps, the inputs are the values calculated in the previous correction step.

3.2 Input-Output Solve for *HydroDyn*, *SubDyn*, *MAP*, *FEAMooring*, *IceFloe*, and the Platform Reference Point Mesh in *ElastoDyn*

This procedure implements Solve Option 1 for the accelerations and loads in *HydroDyn*, *SubDyn*, *MAP*, *FEAMooring*, and *ElastoDyn* (at its platform reference point mesh). The other input-output relationships for these modules are solved using Solve Option 2.

```
procedure ED_HD_SD_Mooring_Ice_InputOutputSolve()
    y_MAP ← CalcOutput(p_MAP, u_MAP, x_MAP, xd_MAP, z_MAP)
    y_FEAM ← CalcOutput(p_FEAM, u_FEAM, x_FEAM, xd_FEAM, z_FEAM)
    y_IceF ← CalcOutput(p_IceF, u_IceF, x_IceF, xd_IceF, z_IceF)
    y_IceD(:) ← CalcOutput(p_IceD(:), u_IceD(:), x_IceD(:), xd_IceD(:), z_IceD(:))
      ▷ Form u vector using loads and accelerations from u_HD, u_SD, and
    platform reference input from u_ED
    u ← u_vec(u_HD, u_SD, u_ED)
    k ← 0
    loop      ▷ Solve for loads and accelerations (direct feed-through terms)
        y_ED ← ED_CalcOutput(p_ED, u_ED, x_ED, xd_ED, z_ED)
        y_SD ← SD_CalcOutput(p_SD, u_SD, x_SD, xd_SD, z_SD)
        y_HD ← HD_CalcOutput(p_HD, u_HD, x_HD, xd_HD, z_HD)
        if k ≥ k_max then
            exit loop
        end if
        u_MAP_tmp ← TransferMeshMotions(y_ED)
        u_FEAM_tmp ← TransferMeshMotions(y_ED)
        u_IceF_tmp ← TransferMeshMotions(y_SD)
        u_IceD_tmp(:) ← TransferMeshMotions(y_SD)
        u_HD_tmp ← TransferMeshMotions(y_ED, y_SD)
        u_SD_tmp ← TransferMeshMotions(y_ED)
                   ∪ TransferMeshLoads(y_SD,
                                       y_HD, u_HD_tmp,
                                       y_IceF, u_IceF_tmp)
                                       y_IceD(:), u_IceD_tmp(:))
        u_ED_tmp ← TransferMeshLoads(y_ED,
                                     y_HD, u_HD_tmp,
                                     y_SD, u_SD_tmp,
                                     y_MAP, u_MAP_tmp,
                                     y_FEAM, u_FEAM_tmp)
        U_Residual ← u − u_vec(u_HD_tmp, u_SD_tmp, u_ED_tmp)
```

```
        if last Jacobian was calculated at least DT_UJac seconds ago then
            Calculate ∂U/∂u
        end if
        Solve (∂U/∂u) Δu = −U_Residual for Δu

        if ‖Δu‖_2 < tolerance then                        ▷ To be implemented later
            exit loop
        end if

        u ← u + Δu
        Transfer u to u_HD, u_SD, and u_ED▷ loads and accelerations only
        k = k + 1
    end loop
                          ▷ Transfer non-acceleration fields to motion input meshes

    u_HD(not accelerations) ← TransferMeshMotions(y_ED, y_SD)
    u_SD(not accelerations) ← TransferMeshMotions(y_ED)

    u_MAP ← TransferMeshMotions(y_ED)
    u_FEAM ← TransferMeshMotions(y_ED)
    u_IceF ← TransferMeshMotions(y_SD)
    u_IceD(:) ← TransferMeshMotions(y_SD)
end procedure
```

3.3 Implementation of line2-to-line2 loads mapping

The inverse-lumping of loads is computed by a block matrix solve for the distributed forces and moments, using the following equation:

$$\begin{bmatrix} F^{DL} \\ M^{DL} \end{bmatrix} = \begin{bmatrix} A & 0 \\ B & A \end{bmatrix} \begin{bmatrix} F^{D} \\ M^{D} \end{bmatrix} \tag{1}$$

Because the forces do not depend on the moments, we first solve for the distributed forces, F^D:

$$\left[F^{DL}\right] = [A] \left[F^{D}\right] \tag{2}$$

We then use the known values to solve for the distributed moments, M^D:

$$\left[M^{DL}\right] = \begin{bmatrix} B & A \end{bmatrix} \begin{bmatrix} F^{D} \\ M^{D} \end{bmatrix} = [B] \left[F^{D}\right] + [A] \left[M^{D}\right] \tag{3}$$

or

$$\left[M^{DL}\right] - [B] \left[F^{D}\right] = [A] \left[M^{D}\right] \tag{4}$$

Rather than store the matrix B, we directly perform the cross products that the matrix B represents. This makes the left-hand side of Equation 4 known, leaving us with one matrix solve. This solve uses the same matrix A used to obtain the distributed forces in Equation 2; A depends only on element reference positions and connectivity. We use the LU factorization of matrix A so that the second solve does not introduce much additional overhead.

4 Solve Option 2 Improvements

4.1 Input-Output Solves inside AdvanceStates

This algorithm documents the procedure for advancing states with option 2 Input-Output solves in FAST, assuming all modules are in use. If an individual module is not in use during a particular simulation, the calls to that module's subroutines are omitted and the module's inputs and outputs are neither set nor used.

```
procedure FAST_AdvanceStates()
    ED_UpdateStates(p_ED, u_ED, x_ED, xd_ED, z_ED)
    y_ED ← ED_CalcOutput(p_ED, u_ED, x_ED, xd_ED, z_ED)

    u_BD(hub and root motions) ← TransferOutputsToInputs(y_ED)
    BD_UpdateStates(p_BD, u_BD, x_BD, xd_BD, z_BD)
    y_BD ← BD_CalcOutput(p_BD, u_BD, x_BD, xd_BD, z_BD)

    u_AD(not InflowWind) ← TransferOutputsToInputs(y_ED, y_BD)
    u_IfW ← TransferOutputsToInputs(y_ED, y_BD at u_AD nodes)
    IfW_UpdateStates(p_IfW, u_IfW, x_IfW, xd_IfW, z_IfW)
    y_IfW ← IfW_CalcOutput(u_IfW and other IfW data structures)

    u_AD(InflowWind only) ← TransferOutputsToInputs(y_IfW)
    u_SrvD ← TransferOutputsToInputs(y_ED, y_BD, y_IfW)
    AD_UpdateStates(p_AD, u_AD, x_AD, xd_AD, z_AD)
    SrvD_UpdateStates(p_SrvD, u_SrvD, x_SrvD, xd_SrvD, z_SrvD)

    All other modules (used in Solve Option 1) advance their states
end procedure
```

Note that AeroDyn and ServoDyn outputs get calculated inside the *CalcOutputs_And_SolveForInputs* routine. ElastoDyn, BeamDyn, and InflowWind outputs do not get recalculated in *CalcOutputs_And_SolveForInputs* except for the first time the routine is called (because CalcOutput is called before UpdateStates at time 0).

5 Linearization

5.1 Loads Transfer

The loads transfer can be broken down into four components, all of which are used in the Line2-to-Line2 loads transfer:

1. Augment the source mesh with additional nodes.
2. Lump the distributed loads on the augmented Line2 source mesh to a Point mesh.
3. Perform Point-to-Point loads transfer.

4. Distribute (or "unlump") the point loads.

The other loads transfers are just subsets of the Line2-to-Line2 transfer:

- Line2-to-Line2: Perform steps 1, 2, 3, and 4.
- Line2-to-Point: Perform steps 1, 2, and 3.
- Point-to-Line2: Perform steps 3 and 4.
- Point-to-Point: Perform step 3.

Each of the four steps can be represented with a linear equation. The linearization of the loads transfers is just multiplying the appropriate matrices generated in each of the steps.

5.1.1 Step 1: Augment the source mesh

The equation that linearizes mesh augmentation is

$$\left\{\begin{matrix} \vec{u}^D \\ \vec{u}^{SA} \\ \vec{f}^{SA} \\ \vec{m}^{SA} \end{matrix}\right\} = \begin{bmatrix} I_{N_D} & 0 & 0 & 0 \\ 0 & M^A & 0 & 0 \\ 0 & 0 & M^A & 0 \\ 0 & 0 & 0 & M^A \end{bmatrix} \left\{\begin{matrix} \vec{u}^D \\ \vec{u}^S \\ \vec{f}^S \\ \vec{m}^S \end{matrix}\right\} \tag{5}$$

where $M^A \in \mathbb{R}^{N_{SA},N_S}$ indicates the mapping of nodes from the source mesh (with N_S nodes) to the augmented source mesh (with N_{SA} nodes). The destination mesh (with N_D nodes) is unchanged, as is indicated by matrix I_{N_D}.

5.1.2 Step 2: Lump loads on a Line2 mesh to a Point mesh

The equation that linearizes the lumping of loads is

$$\left\{\begin{matrix} \vec{u}^{SA} \\ \vec{F}^{SAL} \\ \vec{M}^{SAL} \end{matrix}\right\} = \begin{bmatrix} I_{N_{SA}} & 0 & 0 \\ 0 & M_{li}^{SL} & 0 \\ M_{uS}^{SL} & M_f^{SL} & M_{li}^{SL} \end{bmatrix} \left\{\begin{matrix} \vec{u}^{SA} \\ \vec{f}^{SA} \\ \vec{m}^{SA} \end{matrix}\right\} \tag{6}$$

where $M_{li}^{SL}, M_{uS}^{SL}, M_f^{SL} \in \mathbb{R}^{N_{SA},N_{SA}}$ are block matrices that indicate the mapping of the lumped values to distributed values. M_{li}^{SL} is matrix A in Equation 2, which depends only on element reference positions and connectivity. Matrices M_{uS}^{SL} and M_f^{SL} also depend on values at their operating point.

5.1.3 Step 3: Perform Point-to-Point loads transfer

The equation that performs Point-to-Point load transfer can be written as

$$\left\{\begin{matrix} \vec{u}^D \\ \vec{u}^S \\ \vec{F}^D \\ \vec{M}^D \end{matrix}\right\} = \begin{bmatrix} I_{N_D} & 0 & 0 & 0 \\ 0 & I_{N_S} & 0 & 0 \\ 0 & 0 & M_{li}^D & 0 \\ M_{uD}^D & M_{uS}^D & M_f^D & M_{li}^D \end{bmatrix} \left\{\begin{matrix} \vec{u}^D \\ \vec{u}^S \\ \vec{F}^S \\ \vec{D}^S \end{matrix}\right\} \tag{7}$$

where $M_{li}^D, M_{uS}^D, M_f^D \in \mathbb{R}^{N_D,N_S}$ are block matrices that indicate the transfer of loads from one source node to a node on the destination mesh. $M_{uD}^D \in \mathbb{R}^{N_D,N_D}$ is a diagonal matrix that indicates how the destination mesh's displaced position effects the transfer.

5.1.4 Step 4: Distribute Point loads to a Line2 mesh

Distributing loads from a Point mesh to a Line2 mesh is the inverse of step 2.

From Equation 6 the equation that linearizes the lumping of loads on a destination mesh is

$$\begin{Bmatrix} \vec{u}^D \\ \vec{F}^D \\ \vec{M}^D \end{Bmatrix} = \begin{bmatrix} I_{N_D} & 0 & 0 \\ 0 & M_{li}^{DL} & 0 \\ M_{uD}^{DL} & M_f^{DL} & M_{li}^{DL} \end{bmatrix} \begin{Bmatrix} \vec{u}^D \\ \vec{f}^D \\ \vec{m}^D \end{Bmatrix} \tag{8}$$

where $M_{li}^{DL}, M_{uD}^{DL}, M_f^{DL} \in \mathbb{R}^{N_D,N_D}$ are block matrices that indicate the mapping of the lumped values to distributed values. It follows that the inverse of this equation is

$$\begin{Bmatrix} \vec{u}^D \\ \vec{f}^D \\ \vec{m}^D \end{Bmatrix} = \begin{bmatrix} I_{N_D} & 0 & 0 \\ 0 & \left[M_{li}^{DL}\right]^{-1} & 0 \\ -\left[M_{li}^{DL}\right]^{-1} M_{uD}^{DL} & -\left[M_{li}^{DL}\right]^{-1} M_f^{DL} \left[M_{li}^{DL}\right]^{-1} & \left[M_{li}^{DL}\right]^{-1} \end{bmatrix} \begin{Bmatrix} \vec{u}^D \\ \vec{F}^D \\ \vec{M}^D \end{Bmatrix} \tag{9}$$

The only inverse we need is already formed (stored as an LU decomposition) from the loads transfer, so we need not form it again.

5.1.5 Putting it together

To form the matrices for loads transfers for the various mappings available, we now need to multiply a few matrices to return the linearization matrix that converts loads from the source mesh to loads on the line mesh:

$$\begin{Bmatrix} \vec{f}^D \\ \vec{m}^D \end{Bmatrix} = \begin{bmatrix} 0 & 0 & M_{li} & 0 \\ M_{uD} & M_{uS} & M_f & M_{li} \end{bmatrix} \begin{Bmatrix} \vec{u}^D \\ \vec{u}^S \\ \vec{f}^D \\ \vec{m}^D \end{Bmatrix} \tag{10}$$

- Line2-to-Line2: Perform steps 1, 2, 3, and 4.

$$\begin{Bmatrix} \vec{f}^D \\ \vec{m}^D \end{Bmatrix} = \begin{bmatrix} 0 & \left[M_{li}^{DL}\right]^{-1} & 0 \\ -\left[M_{li}^{DL}\right]^{-1} M_{uD}^{DL} & -\left[M_{li}^{DL}\right]^{-1} M_f^{DL} \left[M_{li}^{DL}\right]^{-1} & \left[M_{li}^{DL}\right]^{-1} \end{bmatrix} \begin{bmatrix} I_{N_D} & 0 & 0 & 0 \\ 0 & 0 & M_{li}^D & 0 \\ M_{uD}^D & M_{uS}^D & M_f^D & M_{li}^D \end{bmatrix} \begin{bmatrix} I_{N_D} & 0 & 0 & 0 \\ 0 & I_{N_{SA}} & 0 & 0 \\ 0 & 0 & M_{li}^{SL} & 0 \\ 0 & M_{uS}^{SL} & M_f^{SL} & M_{li}^{SL} \end{bmatrix} \begin{bmatrix} I_{N_D} & 0 & 0 & 0 \\ 0 & M^A & 0 & 0 \\ 0 & 0 & M^A & 0 \\ 0 & 0 & 0 & M^A \end{bmatrix} \begin{Bmatrix} \vec{u}^D \\ \vec{u}^S \\ \vec{f}^S \\ \vec{m}^S \end{Bmatrix} \tag{11}$$

$$M_{li} = \left(M_{li}^{DL}\right)^{-1} M_{li}^D M_{li}^{SL} M_A \tag{12}$$

$$M_{uD} = \left(M_{li}^{DL}\right)^{-1} \left[M_{uD}^D - M_{uD}^{DL}\right] \tag{13}$$

$$M_{uS} = \left(M_{li}^{DL}\right)^{-1} \left[M_{uS}^D + M_{li}^D M_{uS}^{SL}\right] M_A \tag{14}$$

$$M_f = \left(M_{li}^{DL}\right)^{-1} \left(\left[M_f^D - M_f^{DL} \left(M_{li}^{DL}\right)^{-1} M_{li}^D\right] M_{li}^{SL} + M_{li}^D M_f^{SL} \right) M_A \tag{15}$$

- Line2-to-Point: Perform steps 1, 2, and 3.

$$\begin{Bmatrix} \vec{F}^D \\ \vec{M}^D \end{Bmatrix} = \begin{bmatrix} 0 & 0 & M_{li}^D & 0 \\ M_{uD}^D & M_{uS}^D & M_f^D & M_{li}^D \end{bmatrix} \begin{bmatrix} I_{N_D} & 0 & 0 & 0 \\ 0 & I_{N_{SA}} & 0 & 0 \\ 0 & 0 & M_{li}^{SL} & 0 \\ 0 & M_{uS}^{SL} & M_f^{SL} & M_{li}^{SL} \end{bmatrix} \begin{bmatrix} I_{N_D} & 0 & 0 & 0 \\ 0 & M^A & 0 & 0 \\ 0 & 0 & M^A & 0 \\ 0 & 0 & 0 & M^A \end{bmatrix} \begin{Bmatrix} \vec{u}^D \\ \vec{u}^S \\ \vec{f}^S \\ \vec{m}^S \end{Bmatrix} \tag{16}$$

The linearization routine returns these four matrices:

$$M_{li} = M_{li}^D M_{li}^{SL} M_A \tag{17}$$

$$M_{uD} = M_{uD}^D \tag{18}$$

$$M_{uS} = \left[M_{uS}^D + M_{li}^D M_{uS}^{SL}\right] M_A \tag{19}$$

$$M_f = \left[M_f^D M_{li}^{SL} + M_{li}^D M_f^{SL}\right] M_A \tag{20}$$

- Point-to-Line2: Perform steps 3 and 4.

$$\begin{Bmatrix} \vec{f}^D \\ \vec{m}^D \end{Bmatrix} = \begin{bmatrix} 0 & \left[M_{li}^{DL}\right]^{-1} & 0 \\ -\left[M_{li}^{DL}\right]^{-1} M_{uD}^{DL} & -\left[M_{li}^{DL}\right]^{-1} M_f^{DL} \left[M_{li}^{DL}\right]^{-1} & \left[M_{li}^{DL}\right]^{-1} \end{bmatrix} \begin{bmatrix} I_{N_D} & 0 & 0 & 0 \\ 0 & 0 & M_{li}^D & 0 \\ M_{uD}^D & M_{uS}^D & M_f^D & M_{li}^D \end{bmatrix} \begin{Bmatrix} \vec{u}^D \\ \vec{u}^S \\ \vec{F}^S \\ \vec{M}^S \end{Bmatrix} \tag{21}$$

The linearization routine returns these four matrices:

$$M_{li} = \left(M_{li}^{DL}\right)^{-1} M_{li}^D \tag{22}$$

$$M_{uD} = \left(M_{li}^{DL}\right)^{-1} \left[M_{uD}^D - M_{uD}^{DL}\right] \tag{23}$$

$$M_{uS} = \left(M_{li}^{DL}\right)^{-1} M_{uS}^D \tag{24}$$

$$M_f = \left(M_{li}^{DL}\right)^{-1} \left[M_f^D - M_f^{DL} M_{li}\right] \tag{25}$$

- Point-to-Point: Perform step 3.

$$\begin{Bmatrix} \vec{F}^D \\ \vec{M}^D \end{Bmatrix} = \begin{bmatrix} 0 & 0 & M_{li}^D & 0 \\ M_{uD}^D & M_{uS}^D & M_f^D & M_{li}^D \end{bmatrix} \begin{Bmatrix} \vec{u}^D \\ \vec{u}^S \\ \vec{F}^S \\ \vec{M}^S \end{Bmatrix} \tag{26}$$

The linearization routine returns these four matrices:

$$M_{li} = M_{li}^D \tag{27}$$

$$M_{uD} = M_{uD}^D \tag{28}$$

$$M_{uS} = M_{uS}^D \tag{29}$$

$$M_f = M_f^D \tag{30}$$