

# Solve Algorithms in OpenFAST

B Jonkman, A Platt

December 16, 2021

## 1 Definitions and Nomenclature

<b>Module Name</b>	<b>Abbreviation in Module</b>	<b>Abbreviation in this Document</b>
ElastoDyn	ED	ED
BeamDyn	BD	BD
AeroDyn14	AD14	AD14
AeroDyn	AD	AD
ServoDyn	SrvD	SrvD
SubDyn	SD	SD
ExtPtfm	ExtPtfm	ExtPtfm
HydroDyn	HydroDyn	HD
MAP++	MAPp	MAP
FEAMooring	FEAM	FEAM
MoorDyn	MD	MD
OrcaFlexInterface	Orca	Orca
InflowWind	IfW	IfW
IceFloe	IceFloe	IceF
IceDyn	IceD	IceD
AeroDisk	ADsk	ADsk
Simplified ElastoDyn	SED	SED

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.

```
1: procedure CALCOUTPUTS_AND_SOLVFORINPUTS()
2:
3:    $y\_ED \leftarrow ED\_CALCOUTPUT(p\_ED, u\_ED, x\_ED, xd\_ED, z\_ED)$ 
4:    $y\_SED \leftarrow SED\_CALCOUTPUT(p\_SED, u\_SED, x\_SED, xd\_SED, z\_SED)$ 
5:    $u\_BD \leftarrow TRANSFEROUTPUTSTOINPUTS(y\_ED, y\_SrvD)$ 
6:
7:    $y\_BD \leftarrow BD\_CALCOUTPUT(p\_BD, u\_BD, x\_BD, xd\_BD, z\_BD)$ 
8:    $u\_AD(\text{no IfW}) \leftarrow TRANSFEROUTPUTSTOINPUTS(y\_ED, y\_SED, y\_BD)$ 
9:    $u\_ADsk(\text{no IfW}) \leftarrow TRANSFEROUTPUTSTOINPUTS(y\_ED, y\_SED, y\_BD)$ 
10:   $u\_IfW \leftarrow TRANSFEROUTPUTSTOINPUTS(y\_ED, y\_SED, u\_ADnodes)$ 
11:
12:   $y\_IfW \leftarrow IfW\_CALCOUTPUT(u\_IfW \text{ and other IfW data structures})$ 
13:   $u\_AD(\text{InflowWind only}) \leftarrow TRANSFEROUTPUTSTOINPUTS(y\_IfW)$ 
14:   $u\_ADsk(\text{InflowWind only}) \leftarrow TRANSFEROUTPUTSTOINPUTS(y\_IfW)$ 
15:   $u\_SrvD \leftarrow TRANSFEROUTPUTSTOINPUTS(y\_ED, y\_SED, y\_IfW, y\_BD)$ 
16:
17:   $y\_AD \leftarrow AD\_CALCOUTPUT(p\_AD, u\_AD, x\_AD, xd\_AD, z\_AD)$ 
18:   $y\_ADsk \leftarrow ADSK\_CALCOUTPUT(p\_ADsk, u\_ADsk, x\_ADsk, xd\_ADsk, z\_ADsk)$ 
19:   $y\_SrvD \leftarrow SRVD\_CALCOUTPUT(p\_SrvD, u\_SrvD,$ 
                                 $x\_SrvD, xd\_SrvD, z\_SrvD)$ 
20:   $u\_ED \leftarrow TRANSFEROUTPUTSTOINPUTS(y\_AD, y\_ADsk, y\_SrvD)$ 
21:   $u\_BD \leftarrow TRANSFEROUTPUTSTOINPUTS(y\_AD, y\_ADsk, y\_SrvD)$ 
22:
23:   $u\_HD \leftarrow TRANSFERMESHMOTIONS(y\_ED)$ 
24:   $u\_SD \leftarrow TRANSFERMESHMOTIONS(y\_ED)$ 
25:   $u\_ExtPtfm \leftarrow TRANSFERMESHMOTIONS(y\_ED)$ 
26:   $u\_MAP \leftarrow TRANSFERMESHMOTIONS(y\_ED)$ 
27:   $u\_FEAM \leftarrow TRANSFERMESHMOTIONS(y\_ED)$ 
28:   $u\_MD \leftarrow TRANSFERMESHMOTIONS(y\_ED)$ 
29:   $u\_Orca \leftarrow TRANSFERMESHMOTIONS(y\_ED)$ 
30:   $u\_SrvD\%PtfmStC \leftarrow TRANSFERMESHMOTIONS(y\_ED)^*$ 
31:
32:  SOLVEOPTION1()
33:
34:   $u\_IfW \leftarrow TRANSFEROUTPUTSTOINPUTS(u\_AD, u\_ADsk, y\_ED, y\_SED)$ 
35:   $u\_AD \leftarrow TRANSFEROUTPUTSTOINPUTS(y\_ED, y\_SED)$ 
```

---

\*Only if using ServoDyn Structural control with platform TMD.

```

36:    $u\_ADsk \leftarrow \text{TRANSFEROUTPUTSTOINPUTS}(y\_ED, y\_SED)$ 
37:    $u\_SrvD \leftarrow \text{TRANSFEROUTPUTSTOINPUTS}(y\_ED, y\_SED, y\_AD, u\_ADsk, y\_BD, y\_SD)$ 
38: 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*, *OrcaFlex-Interface*, *BeamDyn*, *ExtPtfm*, *MAP*, *FEAMooring*, *MoorDyn*, *FEAMooring*, *IceFloe*, *IceDyn*, and the Platform Reference Point Mesh in *ElastoDyn*

This procedure implements Solve Option 1 for the accelerations and loads in *HydroDyn*, *SubDyn*, *MAP*, *FEAMooring*, *OrcaFlexInterface*, *MoorDyn*, *BeamDyn*, *ExtPtfm*, *IceFloe*, *IceDyn*, and *ElastoDyn* (at its platform reference point mesh). The other input-output relationships for these modules are solved using Solve Option 2. This solve option is not used when the *Simplified ElastoDyn* module is in use.

```

1: procedure SOLVEOPTION1()
2:
3:    $y\_MAP \leftarrow \text{CALCOUTPUT}(p\_MAP, u\_MAP, x\_MAP, xd\_MAP, z\_MAP)$ 
4:    $y\_MD \leftarrow \text{CALCOUTPUT}(p\_MD, u\_MD, x\_MD, xd\_MD, z\_MD)$ 
5:    $y\_FEAM \leftarrow \text{CALCOUTPUT}(p\_FEAM, u\_FEAM, x\_FEAM, xd\_FEAM, z\_FEAM)$ 
6:    $y\_IceF \leftarrow \text{CALCOUTPUT}(p\_IceF, u\_IceF, x\_IceF, xd\_IceF, z\_IceF)$ 
7:    $y\_IceD(:) \leftarrow \text{CALCOUTPUT}(p\_IceD(:), u\_IceD(:), x\_IceD(:), xd\_IceD(:), z\_IceD(:))$ 
8:    $y\_SrvD \leftarrow \text{CALCOUTPUT}(p\_SrvD, u\_SrvD, x\_SrvD, xd\_SrvD, z\_SrvD)^\dagger$ 
9:
10:  ▷ Form  $u$  vector using loads and accelerations from  $u\_HD$ ,  $u\_BD$ ,  $u\_SD$ ,
     $u\_Orca$ ,  $u\_ExtPtfm$ ,  $u\_SrvD^\ddagger$  and platform reference input from  $u\_ED$ 
11:
12:   $u \leftarrow \text{U\_VEC}(u\_HD, u\_SD, u\_ED, u\_BD, u\_Orca, u\_ExtPtfm)$ 
13:   $k \leftarrow 0$ 
14:  loop    ▷ Solve for loads and accelerations (direct feed-through terms)

```

<sup>†</sup>Only if using ServoDyn Structural control with platform or tower TMD and full *ElastoDyn*.

<sup>‡</sup>Only if using ServoDyn Structural control with platform or tower TMD and full *ElastoDyn* and *SubDyn*.

```

15:  $y_{ED} \leftarrow \text{ED\_CALCOUTPUT}(p_{ED}, u_{ED}, x_{ED}, xd_{ED}, z_{ED})$ 
16:  $y_{SD} \leftarrow \text{SD\_CALCOUTPUT}(p_{SD}, u_{SD}, x_{SD}, xd_{SD}, z_{SD})$ 
17:  $y_{HD} \leftarrow \text{HD\_CALCOUTPUT}(p_{HD}, u_{HD}, x_{HD}, xd_{HD}, z_{HD})$ 
18:  $y_{BD} \leftarrow \text{BD\_CALCOUTPUT}(p_{BD}, u_{BD}, x_{BD}, xd_{BD}, z_{BD})$ 
19:  $y_{Orca} \leftarrow \text{ORCA\_CALCOUTPUT}(p_{Orca}, u_{Orca}, x_{Orca}, xd_{Orca}, z_{Orca})$ 
20:  $y_{ExtPtfm} \leftarrow \text{CALCOUTPUT}(p_{ExtPtfm}, u_{ExtPtfm}, x_{ExtPtfm}, xd_{ExtPtfm}, z_{ExtPtfm})$ 
21: if  $k \geq k_{max}$  then
22:     exit loop
23: end if
24:  $u_{BD\_tmp} \leftarrow \text{TRANSFERMESHMOTIONS}(y_{ED})$ 
25:  $u_{MAP\_tmp} \leftarrow \text{TRANSFERMESHMOTIONS}(y_{ED})$ 
26:  $u_{FEAM\_tmp} \leftarrow \text{TRANSFERMESHMOTIONS}(y_{ED})$ 
27:  $u_{Orca\_tmp} \leftarrow \text{TRANSFERMESHMOTIONS}(y_{ED})$ 
28:  $u_{MD\_tmp} \leftarrow \text{TRANSFERMESHMOTIONS}(y_{ED})$ 
29:  $u_{IceF\_tmp} \leftarrow \text{TRANSFERMESHMOTIONS}(y_{SD})$ 
30:  $u_{IceD\_tmp}(\cdot) \leftarrow \text{TRANSFERMESHMOTIONS}(y_{SD})$ 
31:  $u_{HD\_tmp} \leftarrow \text{TRANSFERMESHMOTIONS}(y_{ED}, y_{SD})$ 
32:  $u_{SrvD\_tmp} \leftarrow \text{TRANSFERMESHMOTIONS}(y_{ED}, y_{SD})^{\S}$ 
33:  $u_{SD\_tmp} \leftarrow \text{TRANSFERMESHMOTIONS}(y_{ED})$ 
     $\cup \text{TRANSFERMESHLOADS}(y_{SD},$ 
     $y_{HD}, u_{HD\_tmp},$ 
     $y_{IceF}, u_{IceF\_tmp},$ 
     $y_{IceD}(\cdot), u_{IceD\_tmp}(\cdot),$ 
34:  $u_{ED\_tmp} \leftarrow \text{TRANSFERMESHLOADS}(y_{ED},$ 
     $y_{HD}, u_{HD\_tmp},$ 
     $y_{SD}, u_{SD\_tmp},$ 
     $y_{MAP}, u_{MAP\_tmp},$ 
     $y_{FEAM}, u_{FEAM\_tmp},$ 
     $y_{SrvD}, u_{SrvD\_tmp}^{\S})$ 
35:
36:  $U\_Residual \leftarrow u - \text{U\_VEC}(u_{HD\_tmp},$ 
     $u_{SD\_tmp},$ 
     $u_{ED\_tmp},$ 
     $u_{BD\_tmp},$ 
     $u_{Orca\_tmp},$ 
     $u_{ExtPtfm\_tmp})$ 
37:
38: if last Jacobian was calculated at least  $DT\_UJac$  seconds ago then
39:     Calculate  $\frac{\partial U}{\partial u}$ 
40: end if

```

---

<sup>\S</sup>Only if using ServoDyn Structural control with platform or tower TMD and full Elasto-Dyn.

```

41:      Solve  $\frac{\partial U}{\partial u} \Delta u = -U\_Residual$  for  $\Delta u$ 
42:
43:      if  $\|\Delta u\|_2 < \text{tolerance}$  then                                 $\triangleright$  To be implemented later
44:          exit loop
45:      end if
46:
47:       $u \leftarrow u + \Delta u$ 
48:      Transfer  $u$  to  $u\_HD$ ,  $u\_SD$ ,  $u\_BD$ ,  $u\_Orca$ ,  $u\_ExtPtfm$ , and  $u\_ED$   $\triangleright$ 
        loads and accelerations only
49:       $k = k + 1$ 
50:  end loop
51:       $\triangleright$  Transfer non-acceleration fields to motion input meshes
52:
53:   $u\_BD(\text{not accelerations}) \leftarrow \text{TRANSFERMESHMOTIONS}(y\_ED)$ 
54:   $u\_HD(\text{not accelerations}) \leftarrow \text{TRANSFERMESHMOTIONS}(y\_ED, y\_SD)$ 
55:   $u\_SD(\text{not accelerations}) \leftarrow \text{TRANSFERMESHMOTIONS}(y\_ED)$ 
56:   $u\_Orca(\text{not accelerations}) \leftarrow \text{TRANSFERMESHMOTIONS}(y\_ED)$ 
57:   $u\_ExtPtfm(\text{not accelerations}) \leftarrow \text{TRANSFERMESHMOTIONS}(y\_ED)$ 
58:
59:   $u\_MAP \leftarrow \text{TRANSFERMESHMOTIONS}(y\_ED)$ 
60:   $u\_MD \leftarrow \text{TRANSFERMESHMOTIONS}(y\_ED)$ 
61:   $u\_FEAM \leftarrow \text{TRANSFERMESHMOTIONS}(y\_ED)$ 
62:   $u\_IceF \leftarrow \text{TRANSFERMESHMOTIONS}(y\_SD)$ 
63:   $u\_IceD(\cdot) \leftarrow \text{TRANSFERMESHMOTIONS}(y\_SD)$ 
64:   $u\_SrvD \leftarrow \text{TRANSFERMESHMOTIONS}(y\_ED, y\_SD)$ ¶
65: 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:

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

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

$$[F^{DL}] = [A] [F^D] \quad (2)$$

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

$$[M^{DL}] = [B \ A] \begin{bmatrix} F^D \\ M^D \end{bmatrix} = [B] [F^D] + [A] [M^D] \quad (3)$$

or

$$[M^{DL}] - [B] [F^D] = [A] [M^D] \quad (4)$$

---

<sup>¶</sup>Only if using ServoDyn Structural control with platform or tower TMD and full Elasto-Dyn.

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.

```
1: procedure FAST_ADVANCESTATES()
2:   ED_UPDATESTATES(p_ED, u_ED, x_ED, xd_ED, z_ED)
3:   y_ED ← ED_CALCOUTPUT(p_ED, u_ED, x_ED, xd_ED, z_ED)
4:
5:   SED_UPDATESTATES(p_SED, u_SED, x_SED, xd_SED, z_SED)
6:   y_SED ← SED_CALCOUTPUT(p_SED, u_SED, x_SED, xd_SED, z_SED)
7:
8:   u_BD(hub and root motions) ← TRANSFEROUTPUTS_TO_INPUTS(y_ED)
9:   BD_UPDATESTATES(p_BD, u_BD, x_BD, xd_BD, z_BD)
10:  y_BD ← BD_CALCOUTPUT(p_BD, u_BD, x_BD, xd_BD, z_BD)
11:
12:  u_AD(not InflowWind) ← TRANSFEROUTPUTS_TO_INPUTS(y_ED, y_SED, y_BD)
13:  u_ADSk(not InflowWind) ← TRANSFEROUTPUTS_TO_INPUTS(y_ED, y_SED)
14:  u_IfW ← TRANSFEROUTPUTS_TO_INPUTS(y_ED, y_SED, y_BD at u_AD
    nodes)
15:  IFW_UPDATESTATES(p_IfW, u_IfW, x_IfW, xd_IfW, z_IfW)
16:  y_IfW ← IFW_CALCOUTPUT(u_IfW and other IfW data structures)
17:
18:  u_AD(InflowWind only) ← TRANSFEROUTPUTS_TO_INPUTS(y_IfW)
19:  u_ADSk(InflowWind only) ← TRANSFEROUTPUTS_TO_INPUTS(y_IfW)
20:  u_SrvD ← TRANSFEROUTPUTS_TO_INPUTS(y_ED, y_SED, y_BD, y_IfW)
21:  AD_UPDATESTATES(p_AD, u_AD, x_AD, xd_AD, z_AD)
22:  ADSK_UPDATESTATES(p_ADSk, u_ADSk, x_ADSk, xd_ADSk, z_ADSk)
23:  SRVD_UPDATESTATES(p_SrvD, u_SrvD, x_SrvD, xd_SrvD, z_SrvD)
24:
25:  All other modules (used in Solve Option 1) advance their states
26: end procedure
```

Note that AeroDyn and ServoDyn outputs get calculated inside the *CalcOutputsAndSolveForInputs* routine. ElastoDyn, BeamDyn, and InflowWind outputs do not get recalculated in *CalcOutputsAndSolveForInputs* 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

$$\begin{Bmatrix} \vec{u}^D \\ \vec{u}^{SA} \\ \vec{f}^{SA} \\ \vec{m}^{SA} \end{Bmatrix} = \begin{bmatrix} I_{N_D} & 0 & 0 & 0 \\ 0 & M^A & 0 & 0 \\ 0 & 0 & M^A & 0 \\ 0 & 0 & 0 & M^A \end{bmatrix} \begin{Bmatrix} \vec{u}^D \\ \vec{u}^S \\ \vec{f}^S \\ \vec{m}^S \end{Bmatrix} \quad (5)$$

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

$$\begin{Bmatrix} \vec{u}^{SA} \\ \vec{F}^{SAL} \\ \vec{M}^{SAL} \end{Bmatrix} = \begin{bmatrix} I_{N_{SA}} & 0 & 0 \\ 0 & M_{li}^{SL} & 0 \\ M_{uS}^{SL} & M_f^{SL} & M_{li}^{SL} \end{bmatrix} \begin{Bmatrix} \vec{u}^{SA} \\ \vec{f}^{SA} \\ \vec{m}^{SA} \end{Bmatrix} \quad (6)$$



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

$$\begin{Bmatrix} \vec{u}^D \\ \vec{u}^S \\ \vec{F}^D \\ \vec{M}^D \end{Bmatrix} = \begin{bmatrix} I_{N_D} & 0 & 0 & 0 \\ 0 & I_{N_S} & 0 & 0 \\ 0 & 0 & M_{li}^D & 0 \\ M_{uD}^D & M_{uS}^D & M_f^D & M_{li}^D \end{bmatrix} \begin{Bmatrix} \vec{u}^D \\ \vec{u}^S \\ \vec{F}^S \\ \vec{D}^S \end{Bmatrix} \quad (7)$$

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

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

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

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

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:

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

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

$$\begin{aligned} \begin{Bmatrix} \vec{f}^D \\ \vec{m}^D \end{Bmatrix} &= \begin{bmatrix} 0 & [M_{li}^{DL}]^{-1} & 0 \\ -[M_{li}^{DL}]^{-1} M_{uD}^{DL} & -[M_{li}^{DL}]^{-1} M_f^{DL} [M_{li}^{DL}]^{-1} & [M_{li}^{DL}]^{-1} \end{bmatrix} \\ &\quad \begin{bmatrix} I_{ND} & 0 & 0 & 0 \\ 0 & 0 & M_{li}^D & 0 \\ M_{uD}^D & M_{uS}^D & M_f^D & M_{li}^D \end{bmatrix} \begin{bmatrix} I_{ND} & 0 & 0 & 0 \\ 0 & I_{NSA} & 0 & 0 \\ 0 & 0 & M_{li}^{SL} & 0 \\ 0 & M_{uS}^{SL} & M_f^{SL} & M_{li}^{SL} \end{bmatrix} \\ &\quad \begin{bmatrix} I_{ND} & 0 & 0 & 0 \\ 0 & M^A & 0 & 0 \\ 0 & 0 & M^A & 0 \\ 0 & 0 & 0 & M^A \end{bmatrix} \begin{Bmatrix} \vec{u}^D \\ \vec{u}^S \\ \vec{f}^S \\ \vec{m}^S \end{Bmatrix} \quad (11) \end{aligned}$$

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

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

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

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

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

$$\begin{aligned} \begin{Bmatrix} \vec{F}^D \\ \vec{M}^D \end{Bmatrix} &= \begin{bmatrix} 0 & 0 & M_{li}^D & 0 \\ M_{uD}^D & M_{uS}^D & M_f^D & M_{li}^D \end{bmatrix} \begin{bmatrix} I_{ND} & 0 & 0 & 0 \\ 0 & I_{NSA} & 0 & 0 \\ 0 & 0 & M_{li}^{SL} & 0 \\ 0 & M_{uS}^{SL} & M_f^{SL} & M_{li}^{SL} \end{bmatrix} \\ &\quad \begin{bmatrix} I_{ND} & 0 & 0 & 0 \\ 0 & M^A & 0 & 0 \\ 0 & 0 & M^A & 0 \\ 0 & 0 & 0 & M^A \end{bmatrix} \begin{Bmatrix} \vec{u}^D \\ \vec{u}^S \\ \vec{f}^S \\ \vec{m}^S \end{Bmatrix} \quad (16) \end{aligned}$$

The linearization routine returns these four matrices:

$$M_{li} = M_{li}^D M_{li}^{SL} M_A \quad (17)$$

$$M_{uD} = M_{uD}^D \quad (18)$$

$$M_{uS} = [M_{uS}^D + M_{li}^D M_{uS}^{SL}] M_A \quad (19)$$

$$M_f = [M_f^D M_{li}^{SL} + M_{li}^D M_f^{SL}] M_A \quad (20)$$

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

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

The linearization routine returns these four matrices:

$$M_{li} = (M_{li}^{DL})^{-1} M_{li}^D \quad (22)$$

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

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

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

- Point-to-Point: Perform step 3.

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

The linearization routine returns these four matrices:

$$M_{li} = M_{li}^D \quad (27)$$

$$M_{uD} = M_{uD}^D \quad (28)$$

$$M_{uS} = M_{uS}^D \quad (29)$$

$$M_f = M_f^D \quad (30)$$