

Algorithms in FAST v8

Bonnie Jonkman

February 11, 2015

1 Definitions and Nomenclature

Module Name	Abbreviation in Module	Abbreviation in this Document
ElastoDyn	ED	ED
AeroDyn	AD	AD
ServoDyn	SrvD	SrvD
SubDyn	SD	SD
HydroDyn	HydroDyn	HD
MAP	MAP	MAP
FEAMooring	FEAM	FEAM
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.

```

1: procedure CALCOUTPUTS_AND_SOLVEFORINPUTS()
2:    $y\_ED \leftarrow \text{ED\_CALCOUTPUT}(p\_ED, u\_ED, x\_ED, xd\_ED, z\_ED)$ 
3:
4:    $u\_AD \leftarrow \text{TRANSFEROUTPUTSTOINPUTS}(y\_ED)$ 
5:    $y\_AD \leftarrow \text{AD\_CALCOUTPUT}(p\_AD, u\_AD, x\_AD, xd\_AD, z\_AD)$ 
6:
7:    $u\_SrvD \leftarrow \text{TRANSFEROUTPUTSTOINPUTS}(y\_ED, y\_AD)$ 
8:    $y\_SrvD \leftarrow \text{SRVD\_CALCOUTPUT}(p\_SrvD, u\_SrvD,$ 
                                    $x\_SrvD, xd\_SrvD, z\_SrvD)$ 
9:
10:   $u\_ED(\text{not platform reference point}) \leftarrow \text{TRANSFEROUTPUTSTOINPUTS}(y\_SrvD, y\_AD)$ 
11:   $u\_HD \leftarrow \text{TRANSFERMESHMOTIONS}(y\_ED)$ 
12:   $u\_SD \leftarrow \text{TRANSFERMESHMOTIONS}(y\_ED)$ 
13:   $u\_MAP \leftarrow \text{TRANSFERMESHMOTIONS}(y\_ED)$ 
14:   $u\_FEAM \leftarrow \text{TRANSFERMESHMOTIONS}(y\_ED)$ 
15:
16:   $\text{ED\_HD\_SD\_MOORING\_ICE\_INPUTOUTPUTSOLVE}()$ 
17:
18:   $u\_AD \leftarrow \text{TRANSFEROUTPUTSTOINPUTS}(y\_ED)$ 
19:   $u\_SrvD \leftarrow \text{TRANSFEROUTPUTSTOINPUTS}(y\_ED, y\_AD)$ 
20: 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.

```

1: procedure ED_HD_SD_MOORING_ICE_INPUTOUTPUTSOLVE()
2:
3:    $y\_MAP \leftarrow \text{CALCOUTPUT}(p\_MAP, u\_MAP, x\_MAP, xd\_MAP, z\_MAP)$ 
4:    $y\_FEAM \leftarrow \text{CALCOUTPUT}(p\_FEAM, u\_FEAM, x\_FEAM, xd\_FEAM, z\_FEAM)$ 
5:    $y\_IceF \leftarrow \text{CALCOUTPUT}(p\_IceF, u\_IceF, x\_IceF, xd\_IceF, z\_IceF)$ 
6:    $y\_IceD(:) \leftarrow \text{CALCOUTPUT}(p\_IceD(:), u\_IceD(:), x\_IceD(:), xd\_IceD(:), z\_IceD(:))$ 
7:
8:    $\triangleright$  Form  $u$  vector using loads and accelerations from  $u\_HD$ ,  $u\_SD$ , and
platform reference input from  $u\_ED$ 
9:
10:   $u \leftarrow \text{U\_VEC}(u\_HD, u\_SD, u\_ED)$ 
11:   $k \leftarrow 0$ 
12:  loop  $\triangleright$  Solve for loads and accelerations (direct feed-through terms)
13:     $y\_ED \leftarrow \text{ED\_CALCOUTPUT}(p\_ED, u\_ED, x\_ED, xd\_ED, z\_ED)$ 
14:     $y\_SD \leftarrow \text{SD\_CALCOUTPUT}(p\_SD, u\_SD, x\_SD, xd\_SD, z\_SD)$ 
15:     $y\_HD \leftarrow \text{HD\_CALCOUTPUT}(p\_HD, u\_HD, x\_HD, xd\_HD, z\_HD)$ 
16:    if  $k \geq k\_max$  then
17:      exit loop
18:    end if
19:     $u\_MAP\_tmp \leftarrow \text{TRANSFERMESHMOTIONS}(y\_ED)$ 
20:     $u\_FEAM\_tmp \leftarrow \text{TRANSFERMESHMOTIONS}(y\_ED)$ 
21:     $u\_IceF\_tmp \leftarrow \text{TRANSFERMESHMOTIONS}(y\_SD)$ 
22:     $u\_IceD\_tmp(:) \leftarrow \text{TRANSFERMESHMOTIONS}(y\_SD)$ 
23:     $u\_HD\_tmp \leftarrow \text{TRANSFERMESHMOTIONS}(y\_ED, y\_SD)$ 
24:     $u\_SD\_tmp \leftarrow \text{TRANSFERMESHMOTIONS}(y\_ED)$ 
            $\cup \text{TRANSFERMESHLOADS}(y\_SD,$ 
                                    $y\_HD, u\_HD\_tmp,$ 
                                    $y\_IceF, u\_IceF\_tmp)$ 
                                    $y\_IceD(:), u\_IceD\_tmp(:))$ 
25:     $u\_ED\_tmp \leftarrow \text{TRANSFERMESHLOADS}(y\_ED,$ 
                                    $y\_HD, u\_HD\_tmp,$ 
                                    $y\_SD, u\_SD\_tmp,$ 
                                    $y\_MAP, u\_MAP\_tmp,$ 
                                    $y\_FEAM, u\_FEAM\_tmp)$ 
26:
27:   $U\_Residual \leftarrow u - \text{U\_VEC}(u\_HD\_tmp, u\_SD\_tmp, u\_ED\_tmp)$ 

```

```

28:
29:   if last Jacobian was calculated at least  $DT\_UJac$  seconds ago then
30:     Calculate  $\frac{\partial U}{\partial u}$ 
31:   end if
32:   Solve  $\frac{\partial U}{\partial u} \Delta u = -U\_Residual$  for  $\Delta u$ 
33:
34:   if  $\|\Delta u\|_2 < \text{tolerance}$  then ▷ To be implemented later
35:     exit loop
36:   end if
37:
38:    $u \leftarrow u + \Delta u$ 
39:   Transfer  $u$  to  $u\_HD$ ,  $u\_SD$ , and  $u\_ED$  ▷ loads and accelerations only
40:    $k = k + 1$ 
41: end loop ▷ Transfer non-acceleration fields to motion input meshes
42:
43:
44:    $u\_HD(\text{not accelerations}) \leftarrow \text{TRANSFERMESHMOTIONS}(y\_ED, y\_SD)$ 
45:    $u\_SD(\text{not accelerations}) \leftarrow \text{TRANSFERMESHMOTIONS}(y\_ED)$ 
46:
47:    $u\_MAP \leftarrow \text{TRANSFERMESHMOTIONS}(y\_ED)$ 
48:    $u\_FEAM \leftarrow \text{TRANSFERMESHMOTIONS}(y\_ED)$ 
49:    $u\_IceF \leftarrow \text{TRANSFERMESHMOTIONS}(y\_SD)$ 
50:    $u\_IceD(\cdot) \leftarrow \text{TRANSFERMESHMOTIONS}(y\_SD)$ 
51: 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 :

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

or

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

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 connectivity (***bjj: check that this is true***). We use the LU factorization of matrix A so that the second solve does not introduce much additional overhead.