Algorithms in FAST v8

Bonnie Jonkman

June 19, 2014

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.

procedure CalcOutputs_And_SolveForInputs()
  $y\_ED \leftarrow$ ED_CalcOutput($p\_ED, u\_ED, x\_ED, xd\_ED, z\_ED$)

  $u\_AD \leftarrow$ TransferOutputsToInputs(y_ED)
  $y\_AD \leftarrow$ AD_CalcOutput($p\_AD, u\_AD, x\_AD, xd\_AD, z\_AD$)

  $u\_SrvD \leftarrow$ TransferOutputsToInputs(y_ED, y_AD)
  $y\_SrvD \leftarrow$ SrvD_CalcOutput($p\_SrvD, u\_SrvD,$
    x_SrvD, xd_SrvD, z_SrvD)

  u_ED(not platform reference point) $\leftarrow$ TransferOutputsToInputs(y_SrvD, y_AD)
  $u\_HD \leftarrow$ TransferMeshMotions(y_ED)
  $u\_SD \leftarrow$ TransferMeshMotions(y_ED)
  $u\_MAP \leftarrow$ TransferMeshMotions(y_ED)
  $u\_FEAM \leftarrow$ TransferMeshMotions(y_ED)

  ED_HD_SD_Mooring_Ice_InputOutputSolve()

  If AeroDyn or ServoDyn had states to update, we should do this:
    $u\_AD \leftarrow$ TransferOutputsToInputs(y_ED)
    $u\_SrvD \leftarrow$ TransferOutputsToInputs(y_ED, y_AD)
  However, they don't so we'll omit these steps for efficiency.
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: $y\_MAP \leftarrow$ CalcOutput($p\_MAP, u\_MAP, x\_MAP, xd\_MAP, z\_MAP$)
4: $y\_FEAM \leftarrow$ CalcOutput($p\_FEAM, u\_FEAM, x\_FEAM, xd\_FEAM, z\_FEAM$)
5: $y\_IceF \leftarrow$ CalcOutput($p\_IceF, u\_IceF, x\_IceF, xd\_IceF, z\_IceF$)
6: $y\_IceD(:) \leftarrow$ CalcOutput($p\_IceD, u\_IceD(:), x\_IceD(:), xd\_IceD(:), z\_IceD(:)$)
7:
8: ▷ Form u vector using loads and accelerations from u_HD, u_SD, and platform reference input from u_ED
9:
10: $u \leftarrow$ u_vec(u_HD, u_SD, u_ED)
11: $k \leftarrow 0$
12: **loop** ▷ Solve for loads and accelerations (direct feed-through terms)
13: $y\_ED \leftarrow$ ED_CalcOutput($p\_ED, u\_ED, x\_ED, xd\_ED, z\_ED$)
14: $y\_SD \leftarrow$ SD_CalcOutput($p\_SD, u\_SD, x\_SD, xd\_SD, z\_SD$)
15: $y\_HD \leftarrow$ 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: $u\_MAP\_tmp \leftarrow$ TransferMeshMotions(y_ED)
20: $u\_FEAM\_tmp \leftarrow$ TransferMeshMotions(y_ED)
21: $u\_IceF\_tmp \leftarrow$ TransferMeshMotions(y_SD)
22: $u\_IceD\_tmp(:) \leftarrow$ TransferMeshMotions(y_SD)
23: $u\_HD\_tmp \leftarrow$ TransferMeshMotions(y_ED, y_SD)
24: $u\_SD\_tmp \leftarrow$ TransferMeshMotions(y_ED) $\cup$ TransferMeshLoads(y_SD, y_HD, u_HD_tmp, y_IceF, u_IceF_tmp) $y\_IceD(:), u\_IceD\_tmp(:)$)
25: $u\_ED\_tmp \leftarrow$ 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: $U\_Residual \leftarrow u -$ 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 Δu
33:
34: **if** $\|\Delta u\|_2 <$ 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**
42: ▷ Transfer non-acceleration fields to motion input meshes
43:
44: u_HD(not accelerations) $\leftarrow$ TransferMeshMotions(y_ED, y_SD)
45: u_SD(not accelerations) $\leftarrow$ TransferMeshMotions(y_ED)
46:
47: $u\_MAP \leftarrow$ TransferMeshMotions(y_ED)
48: $u\_FEAM \leftarrow$ TransferMeshMotions(y_ED)
49: $u\_IceF \leftarrow$ TransferMeshMotions(y_SD)
50: $u\_IceD(:) \leftarrow$ 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:

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

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 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.