HSM 1.00 User Manual

Mark Drela
MIT Department of Aeronautics & Astronautics 24 Jul 2019

Contents

1 Overview

1.1 Code structure

The HSM software suite is a collection of routines for solution of general shell elasticity problems with arbitrarily large deformations. The routines are intended to be embedded in other programs, and hence a general input-preparation program is not provided. Graphical post-processing programs are also not provided for the same reason. What is provided is a few routines to assist in the more complex parts of problem setup, such as stiffness matrix calculation. Also provided is an example driver program (`HSMRUN`) whose code fragments can serve as templates for implementing the various HSM routines in a user's own application.

1.2 Capabilities

Currently, HSM can perform only a direct analysis of a static structural problem, defined as one whose loading and acceleration (if any) are steady in the geometry's xyz coordinate system.

The routines already contain much of the code, specifically the Jacobian calculation, which is needed for also computing the following problems of interest:

- Unsteady structural response to ...
  - General reference-frame motion (e.g. maneuvering aircraft)
  - Time-domain (e.g. nonlinear time-marching)
  - Frequency domain (e.g. linearized Bode response)
  - Eigenmode analysis (e.g. structural instability, flutter)
- Parametric sensitivities to ...
  - Undeformed geometry
  - Mass distribution
  - Stiffness property distribution
  - Loading distribution

These capabilities will be enabled in future HSM versions.

1.3 Solution method

HSM uses the Newton method with an exact Jacobian matrix, and solves the Newton system with a direct sparse block matrix solver to drive the discrete equation residuals to zero. The main solver routine `HSMSOL` can accept a current solution as the initial guess, so that in a gradual parametric sweep the runtime per solution will be considerably smaller than for a single isolated solution.

Currently, the matrix ordering is the same as the unstructured mesh ordering, so that the matrix bandwidth and the runtime will be sensitive to the node order and connectivity. A matrix-reordering algorithm will reduce the sensitivity to ordering, is planned to be implemented in the next HSM version.

Even with optimal ordering, the runtime will still scale roughly as the cube of the number of nodes, which can make large problems costly. Because the HSM formulation has only elliptic Poisson-type equations, its Jacobian matrix is well-conditioned, and hence it is very well suited for an iterative solution method which will reduce the runtime scaling with problem size. Iterative solvers will be considered for future HSM versions.

2 Quick Start

2.1 Program HSMRUN

Program HSMRUN is an ad-hoc example driver program which:

- sets up the inputs for a variety of typical shell elasticity test cases
- calls the main solution routine HSMSOL to compute the primary unknowns
- calls the post-processing routine HSMDEP to compute the dependent unknowns
- calls the output routine HSMOUT which writes the primary and dependent unknowns to formatted files which can be plotted manually.

Therefore, it should serve as a useful template for incorporation of HSM into another application. The input setup for each individual test case is hard-wired in its own commented section, which can be modified as desired.

The program is executed with the following command.

```
% hsmrun [ icase imf ni nj itmax fload ]
```

The arguments are optional, and can be used to modify the case to be run, as described below. Omitting or specifying "-" for an argument will result in its hard-wired default value to be used.

`icase`	case index (see comments in `hsmrun.f` for list of cases implemented) (if `icase < 0`, then use triangular mesh, otherwise use quad mesh)
`imf`	case variant index, in some cases also specifies that a manufactured solution is to be used as the initial guess
`ni,nj`	number of cells in i,j directions
`itmax`	max number of Newton iterations, or ... = `0` perform one system setup and solve, but skip Newton update = `-1` perform one system setup, but skip the solve and Newton update = `-2` only calculate the required size of the work arrays `amat,ipp`
`fload`	scale factor for all imposed loads

All HSM routines assume a general unstructured mesh, consisting of any mix of quad or triangle elements. However, for simplicity HSMRUN initially defines the geometry using one or more logically-rectangular ij surfaces of size specified by `ni, nj`, and then converts these to the unstructured data arrays.

For `itmax = 0` or `-1`, the initial guess will be output as the "solution". This allows an exact manufactured solution to be output, for use as a comparison reference in plotting. It also allows the input geometry and parameters to be examined, which should be useful for debugging of the case setup code.

The hard-wired cases can be set up with either a quad or a triangle mesh, which is selected by the sign of `icase` as indicated above. This argument sign is merely used set logical flag `lquad`, which is what actually controls the mesh type in HSMSOL.

The program prints out the following convergence information for each Newton iteration:

```
iter     dr         dd         dp         rlx   max
 1    0.566E+00  0.157E+00  0.452E-01    0.353  drz @ (   0.80   5.00   0.00)
 1*   0.468E-02  0.000E+00  0.000E+00    1.000  dry @ (   0.80   5.00   0.00)
 1*   0.102E-04  0.000E+00  0.000E+00    1.000  dry @ (   0.80   5.00  -0.00)
 2    0.364E+00  0.101E+00  0.295E-01    0.549  drz @ (   0.80   5.00  -0.00)
 2*   0.460E-02  0.000E+00  0.000E+00    1.000  dry @ (   0.69   5.00  -0.03)
 2*   0.948E-05  0.000E+00  0.000E+00    1.000  dry @ (   0.21   5.00  -0.04)
 3    0.181E+00  0.500E-01  0.143E-01    1.000  drz @ (   0.80   5.00   0.00)
 .
 .
```

The `iter` value is the iteration number, with the attached `*` denoting that it's a membrane-only sub-iteration in which all the $\hat{\mathbf{d}}$ and ψ variables are held fixed. These sub-iterations are relatively quick, since they have only 3 rather than 6 variables per node, and greatly improve convergence robustness for large-deformation cases.

Each `dr`,`dd`,`dp` value is the maximum absolute value of all the Newton changes of that variable at all the nodes, with the vector xyz components conflated. Specifically,

$$\texttt{dr} \;=\; \max_k \Big(\left|\delta r_{x_k}\right|, \left|\delta r_{y_k}\right|, \left|\delta r_{z_k}\right| \Big)$$

$$\texttt{dd} \;=\; \max_k \Big(\left|\delta d_{x_k}\right|, \left|\delta d_{y_k}\right|, \left|\delta d_{z_k}\right| \Big)$$

$$\texttt{dp} \;=\; \max_k \Big(\left|\delta \psi_k\right| \Big)$$

The `rlx` value is the under-relaxation factor used for that iteration to keep the largest Newton change within the specified limits. The printout also indicates which variable triggered the under-relaxation (if any), and where it is located on the undeformed geometry. For example, in the first iteration above, the under-relaxation factor value `rlx = 0.353` was set to limit δr_z at location $\mathbf{r}_0 = (0.8, 5.0, 0.0)$.

2.2 Example cases

2.2.1 Pinned square plate (case 4)

Generate manufactured solution 1 for case 4 on fine mesh, and write to output text files:

```
% hsmrun 4 1 40 40 -1
```

Move the output files to a new directory for use as a comparison reference:

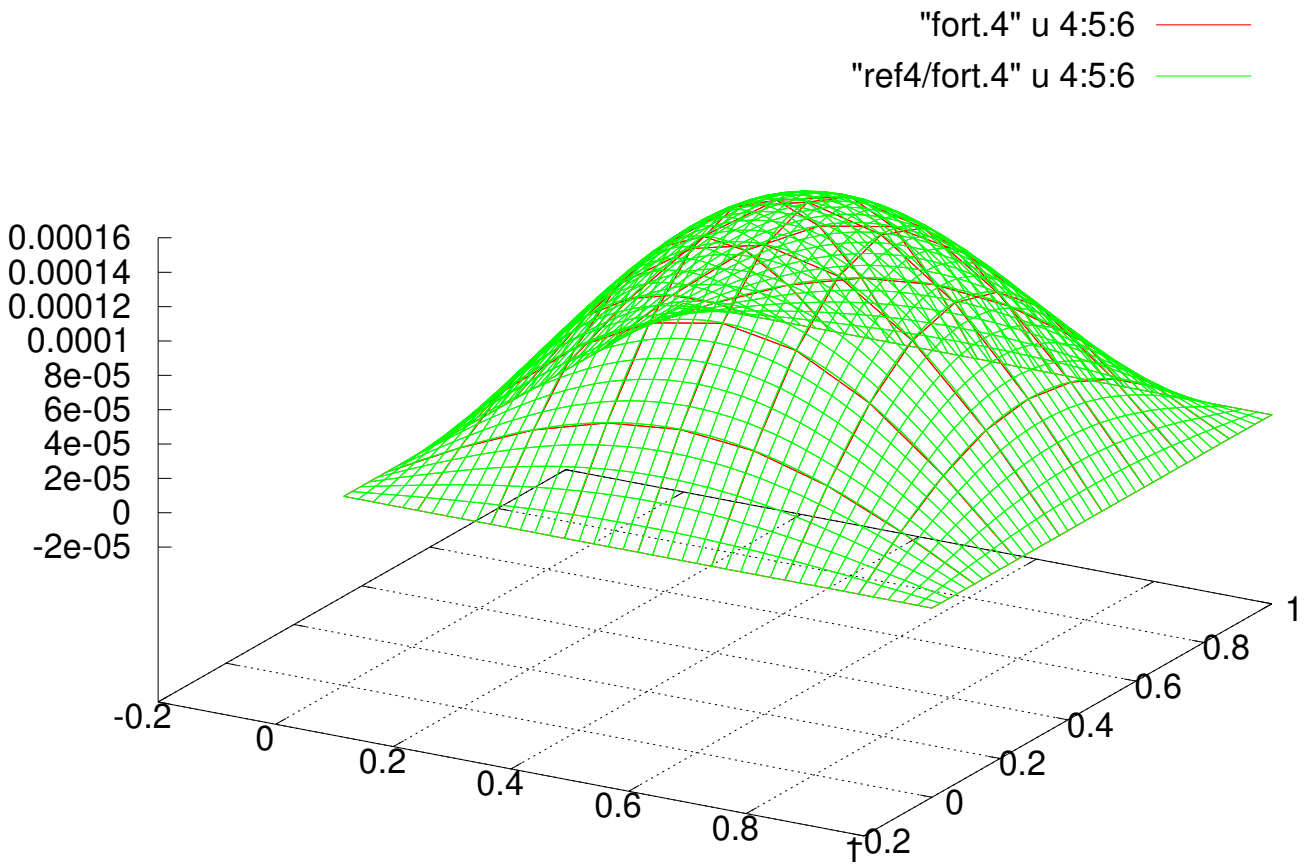
```
% mkdir ref4
% mv fort.* ref4/
```

Compute FE solution on coarse mesh, and write to output text files:

```
% hsmrun 4 1 8 8 20
```

Use `gnuplot` to compare numerical and exact deformed geometry, preferably in another window to allow easy re-plotting:

```
% gnuplot
 > splot "fort.4" u 4:5:6 w l, "ref4/fort.4" u 4:5:6 w l
```



Compare FE and exact values of normal bending moment m_{11} (along $\hat{\mathbf{e}}_1$ direction):

```
 > splot "fort.4" u 4:5:12 w l, "ref4/fort.4" u 4:5:12 w l
```

2.2.2 Quarter of clamped square plate, with two symmetry planes (case 6)

Generate manufactured solution 2 for case 6 on fine mesh, and write to output text files:

```
% hsmrun 6 2 40 40 -1
```

Move the output files to a new directory for use as the reference in comparison plots:

```
% mkdir ref6
% mv fort.* ref6/
```

Compute FE solution on coarse mesh, and write to output text files:

```
% hsmrun 6 2 5 5 20
```

Save one of the coarse solution files for comparisons:

```
% mv fort.4 fort.4c
```

Compute FE solution on medium mesh, and write to output text files:

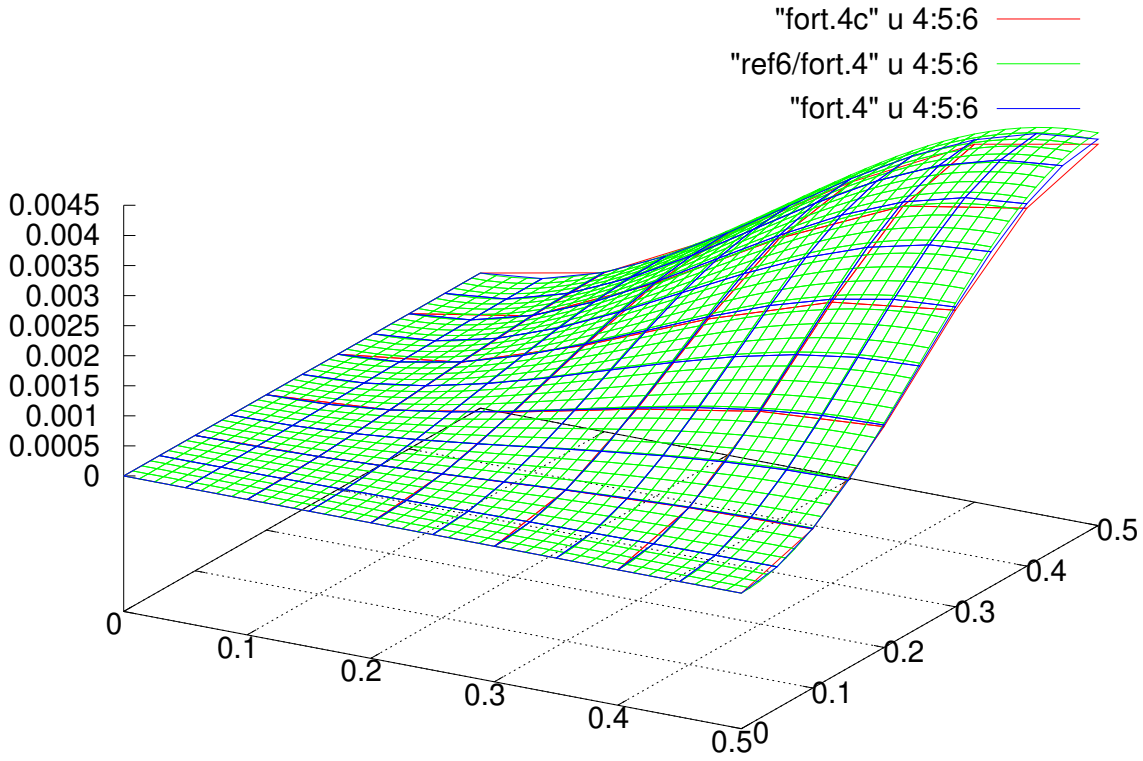
```
% hsmrun 6 2 10 10 20
```

Use `gnuplot` to compare coarse, medium, and exact deformed geometry:

```
% gnuplot
 > splot "fort.4c" u 4:5:6 w l,"ref6/fort.4" u 4:5:6 w l, "fort.4" u 4:5:6 w l
```

Compare coarse, medium, and exact twisting moment m_{12}:

```
 > splot "fort.4c" u 4:5:14 w l,"ref6/fort.4" u 4:5:14 w l, "fort.4" u 4:5:14 w l
```



2.2.3 Wing-like shell beam (case 8)

This case has joints specified between all the top and bottom nodes at the trailing edge.

Generate solution for case 8 on medium mesh, and write to output text files:

```
% hsmrun 8 0 40 30 50
```

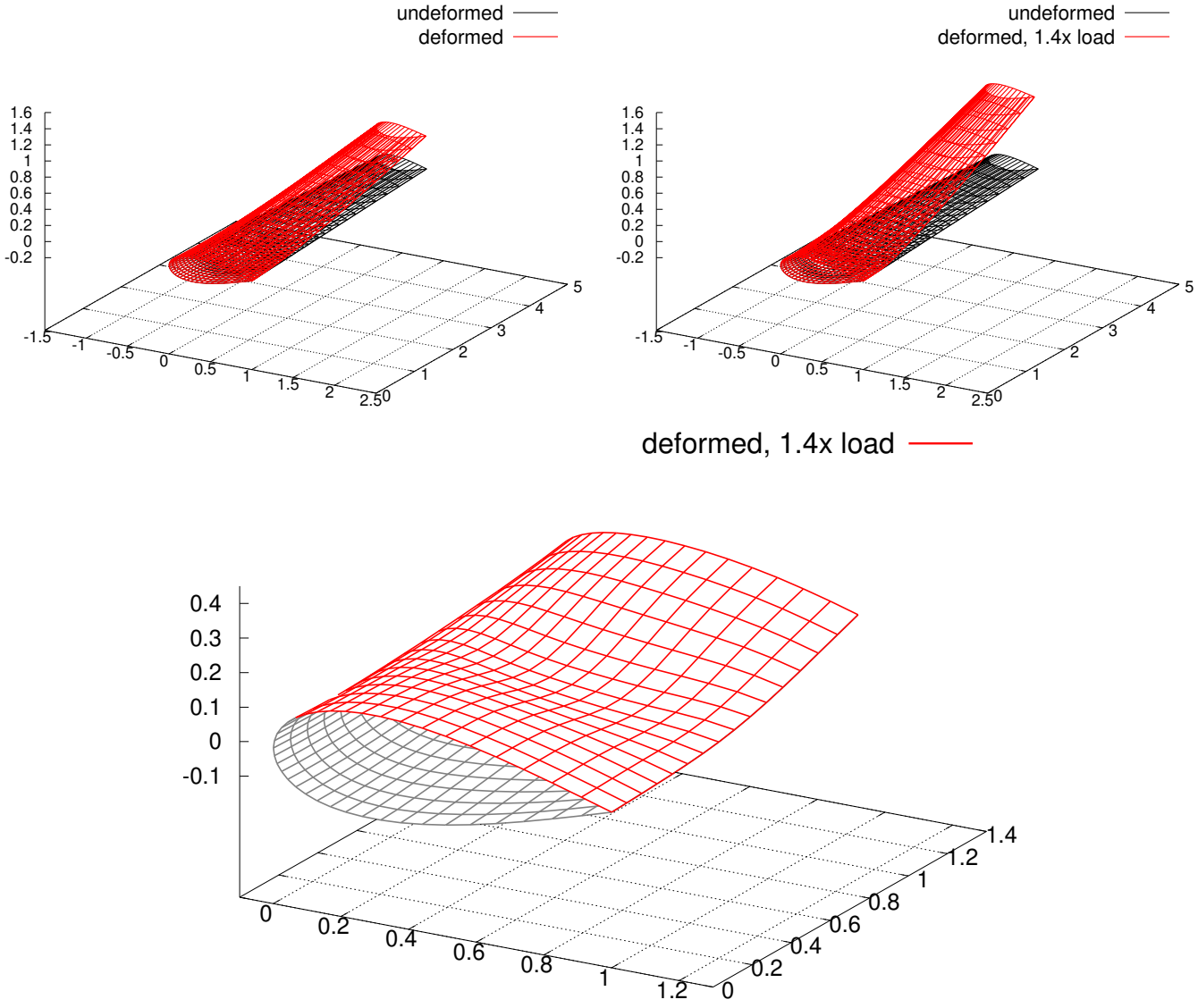
Plot deformed geometry:

```
% gnuplot
 > splot [-1.5:2.5][0:5][-0.2:1.6] "fort.4" u 4:5:6 w l
```

Recompute solution with loading increased 1.4×:

```
% hsmrun 8 0 40 30 50 1.4
```

This will exhibit buckling of the top surface near the wing root, as shown below.



3 Geometry Topology

The components of all boldface vectors and tensors are defined in the global xyz cartesian basis, sketched in Figure 1. One or more points of the structure must be fixed in these axes, which are specified by suitable boundary conditions as described later.

The undeformed shell geometry for HSM is defined by triangular or quadrilateral *elements*, pointing to *nodes* each with position $\mathbf{r}_0$, a surface-normal vector $\hat{\mathbf{n}}_0$, and orthogonal surface-tangent vectors $\hat{\mathbf{e}}_{0_1}, \hat{\mathbf{e}}_{0_2}$. The latter two form the local basis for shell stiffness tensor definition. Typically the elements will be a discretization of some number of C^1 continuous (e.g. CAD) *surfaces*, although HSM itself deals only with the elements and nodes in a fully unstructured manner, and has no concept of a surface.

A surface intersection point with a slope discontinuity must be represented in HSM by a *joint pair* of two nodes, typically with the same $\mathbf{r}_0$ but different $\hat{\mathbf{n}}_0, \hat{\mathbf{e}}_{0_1}, \hat{\mathbf{e}}_{0_2}$, as indicated in Figure 1. More than two surfaces can meet at a corner or along an intersection line, in which case multiple joint pairs must be defined at each such point. Joints will be later described in more detail.

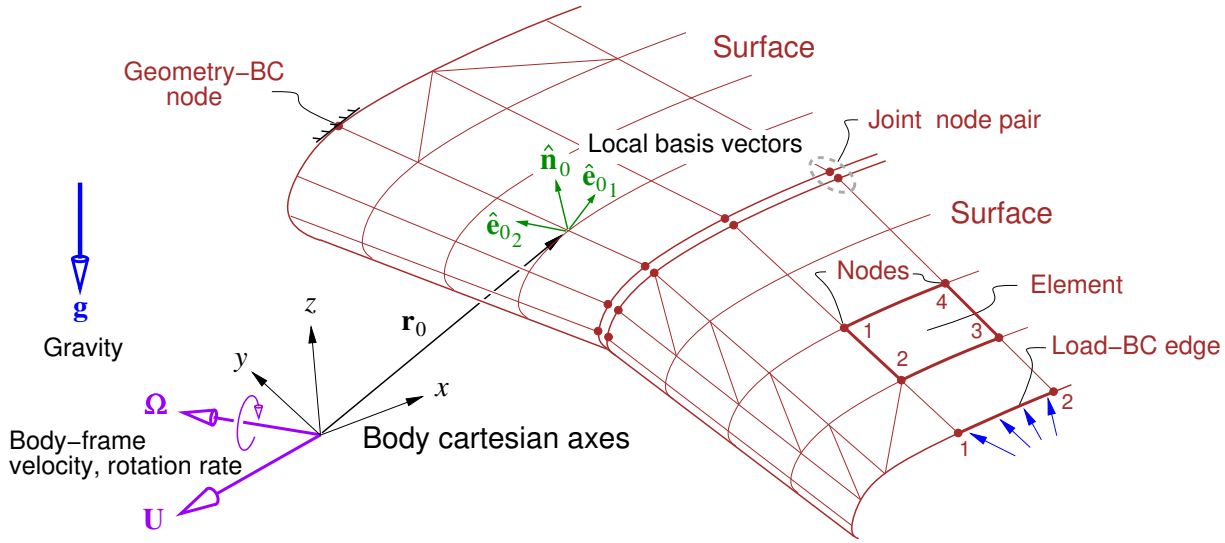


Figure 1: Geometry entities and global parameters.

The discretized geometry and its topology are defined by the data listed in the following table.

Entity	index	pointers to nodes	associated data arrays
Node	`k = 1:nnode`	—	`vars(.k)`, `pars(.k)`, `deps(.k)`
Element	`n = 1:nelem`	`kelem(1:4,n)`	
Load-BC edge	`l = 1:nbcedge`	`kbcedge(1:2,l)`	`pare(.l)`
Geometry-BC node	`l = 1:nbcnode`	`kbcnode(l)`	`lbcnode(l)`, `parp(.l)`
node joint	`j = 1:njoint`	`kjoint(1:2,j)`	

Concentrated force and moment loads can also be applied to any single node, which avoids the need to resolve highly concentrated loading distributions. Since HSM strongly conserves force and moment, the global effect of such a load will be accurately represented. However, the strains and stresses at a point load are theoretically singular, so the solution quality in the point load's immediate vicinity will likely be poor.

4 Data Array Indexing

HSM stores most of its data in the floating-point arrays described below. Example declarations are in the driving program `HSMRUN`.

Convenient mnemonic parameters for the first index of each real array above are defined in the `index.inc` file, with in-line comments describing what each index parameter is. Some examples:

`vars(ivry,k)`	y component of $\mathbf{r}$ (deformed position vector) of node `k`
`pars(lvr0y,k)`	y component of $\mathbf{r}_0$ (undeformed position vector) of node `k`
`deps(jvf11,k)`	f_{11} component of $\bar{\bar{\mathbf{f}}}$ (stress tensor) at node `k`
`pare(lef2z,l)`	z component of $\mathbf{f}_{xyz}$ (specified load/length) at node 2 of edge `l`
`parp(lpry,l)`	y component of $\mathbf{r}_{\mathrm{BC}}$ (specified position) of node `l`
`parg(lgvelz)`	z component of $\mathbf{U}$ (velocity of the xyz origin relative to an inertial frame)

The `index.inc` file is included in all routines, and the mnemonic parameters are used everywhere to make the code more readable.

5 Input Data

The variables and arrays listed in this section must be set to define an HSM case.

5.1 Geometry sizes and pointers

`nnode`	number of nodes
`nelem`	number of elements
`kelem(1:4,n)`	`k` indices of the three or four nodes of each element `n`

When an element is observed with its normal vector $\hat{\mathbf{n}}_0$ pointing towards the observer, its nodes run counterclockwise around the element, as indicated in Figure 1. A triangle element is indicated by its 4th node index being zero, i.e. `kelem(4,n) = 0`.

5.2 Parameters at each node

array elements	num.	sym.	description
`pars(lvr0x:lvr0z,k)`	3	$\mathbf{r}_0$	xyz coordinates of undeformed geometry
`pars(lve01x:lve01z,k)`	3	$\hat{\mathbf{e}}_{0_1}$	in-surface basis unit vector 1 of undeformed geometry
`pars(lve02x:lve02z,k)`	3	$\hat{\mathbf{e}}_{0_2}$	in-surface basis unit vector 2 of undeformed geometry
`pars(lvn0x:lvn0z,k)`	3	$\hat{\mathbf{n}}_0$	normal basis unit vector of undeformed geometry
`pars(lva11:lva66,k)`	6	$\bar{\bar{A}}$	extension/shear stiffness matrix, in $\bar{\bar{\mathbf{e}}}_0$ basis
`pars(lvb11:lvb66,k)`	6	$\bar{\bar{B}}$	extension/bending stiffness matrix, in $\bar{\bar{\mathbf{e}}}_0$ basis
`pars(lvd11:lvd66,k)`	6	$\bar{\bar{D}}$	bending stiffness matrix, in $\bar{\bar{\mathbf{e}}}_0$ basis
`pars(lvs55:lvs44,k)`	2	$\bar{\bar{S}}$	transverse-shear compliance matrix, in $\bar{\bar{\mathbf{e}}}_0$ basis
`pars(lvqn,k)`	1	q_n	shell-following normal load/area
`pars(lvqx:lvqz,k)`	3	$\mathbf{q}_{xyz}$	fixed-direction load/area
`pars(lvmu,k)`	1	μ	mass/area, can be zero if $\mathbf{g}=\mathbf{0}$ and $\boldsymbol{a}=\mathbf{0}$
`pars(lvtsh,k)`	1	h	shell thickness (for post-processing only)
`pars(lvzrf,k)`	1	ζ_{ref}	ref.-surface location within shell (for post-processing only)

If the surface geometry is defined analytically, e.g. as a B-spline or NURB surface $\mathbf{r}_{0(u,v)}$, the unit basis vectors are best computed analytically also. Specifically, we have

$$\hat{\mathbf{n}}_0 \;=\; \pm\frac{\partial_u\mathbf{r}_0 \times \partial_v\mathbf{r}_0}{|\partial_u\mathbf{r}_0 \times \partial_v\mathbf{r}_0|}$$

and $\hat{\mathbf{e}}_{0_1}$ would be computed as some linear combination of $\partial_u\mathbf{r}_0$ and $\partial_v\mathbf{r}_0$, normalized to unit length. Finally, we compute $\hat{\mathbf{e}}_{0_2} = \hat{\mathbf{n}}_0 \times \hat{\mathbf{e}}_{0_1}$. Subroutine `CROSS` is provided to evaluate the cross products.

Subroutine `ORTMAT` is provided to conveniently set the stiffness matrices $\bar{\bar{A}}, \bar{\bar{B}}, \bar{\bar{D}}, \bar{\bar{S}}$ for an orthotropic material with the minimum number of material properties and shell geometry parameters:

E_1	modulus along $\hat{\mathbf{e}}_{0_1}$ direction
E_2	modulus along $\hat{\mathbf{e}}_{0_2}$ direction
ν_{12}	Poisson's ratio, $\nu_{12} = -\varepsilon_{22}/\varepsilon_{11}$, where ε_{22} and ε_{11} are the result of applied stress σ_{11}
G_{12}	shear modulus in 12 plane
G_{13}	transverse shear modulus in 1n plane
G_{23}	transverse shear modulus in 2n plane
h	shell thickness
ζ_{ref}	reference-surface location within shell ($-1 \le \zeta_{\text{ref}} \le +1$)

An isotropic material has only the two stiffness properties E, ν. We then have $E_1 = E_2 = E$, $\nu_{12} = \nu$, and $G_{12} = G_{13} = G_{23} = G = \frac{1}{2}E/(1+\nu)$. In this case the orientation of $\hat{\mathbf{e}}_{0_1}, \hat{\mathbf{e}}_{0_2}$ within the surface tangent plane is arbitrary.

5.3 Global parameters

Global parameters describe gravity (if any), the movement of the xyz origin relative to an inertial (e.g. earth) frame, and also the orientation of the xyz frame relative to the earth XYZ frame (not explicitly shown in Figure 1).

array elements	num.	sym.	description
`parg(lggeex:lggeez)`	3	$\mathbf{g}$	gravity acceleration
`parg(lgvelx:lgvelz)`	3	$\mathbf{U}$	velocity of xyz origin
`parg(lgrotx:lgrotz)`	3	$\mathbf{\Omega}$	rotation rate of frame
`parg(lgvacx:lgvacz)`	3	$\dot{\mathbf{U}}$	velocity rate of xyz origin
`parg(lgracx:lgracz)`	3	$\dot{\mathbf{\Omega}}$	angular acceleration of frame
`parg(lgposx:lgposz)`	3	$\vec{R}^E$	position of xyz origin in XYZ (earth) axes
`parg(lgephi)`	1	Φ	roll angle of frame
`parg(lgethe)`	1	Θ	elevation angle of frame
`parg(lgepsi)`	1	Ψ	heading angle of frame
`parg(lggabx:lggabz)`	3	$\vec{g}^E$	gravity in XYZ (earth) axes, typically $(0, 0, -g)$

For the simplest static case without gravity, all the global parameters above can be set to zero.

5.4 Boundary conditions

For a well-posed problem, all exposed edges of the shell require boundary conditions. These can be either

- loading (Neumann) BCs, applied via weighted residual terms in FE weak formulation, or,
- geometric (Dirichlet) BCs which are applied in strong form, by replacement of the appropriate equation residuals of the boundary nodes.

The two types of BCs are defined by separate arrays, described in the following two sections. HSM also allows point loads, which are implemented together with the geometric BCs since their data requirements are the same.

5.4.1 Edge Loading BCs

Boundary edges with no specified loads are by default free edges with zero loading. Boundary edges which have a nonzero loading are identified with the following pointers.

`nbcedge`	number of boundary element edges where loads are imposed
`kbcedge(1:2,l)`	`k` indices of the two nodes of loaded boundary element edge `l`

The load BC data for each edge is specified by setting the `pare` array elements for the two nodes of the edge. The node values are linearly interpolated along the edge, as indicated in Figure 2.

array elements	num.	sym.	description
`pare(lef1x:lef1z,l)`	3	$(\mathbf{f}_{xyz_{\rm BC}})_1$	fixed-direction force/length at node 1
`pare(lef2x:lef2z,l)`	3	$(\mathbf{f}_{xyz_{\rm BC}})_2$	fixed-direction force/length at node 2
`pare(lef1t:lef1n,l)`	3	$(f_{t_{\rm BC}}, f_{\ell_{\rm BC}}, f_{d_{\rm BC}})_1$	shell-following force/length at node 1
`pare(lef2t:lef2n,l)`	3	$(f_{t_{\rm BC}}, f_{\ell_{\rm BC}}, f_{d_{\rm BC}})_2$	shell-following force/length at node 2
`pare(lem1x:lem1z,l)`	3	$(\mathbf{m}_{xyz_{\rm BC}})_1$	fixed-direction moment/length at node 1
`pare(lem2x:lem2z,l)`	3	$(\mathbf{m}_{xyz_{\rm BC}})_2$	fixed-direction moment/length at node 2
`pare(lem1t:lem1n,l)`	3	$(m_{t_{\rm BC}}, m_{\ell_{\rm BC}}, m_{d_{\rm BC}})_1$	shell-following moment/length at node 1
`pare(lem2t:lem2n,l)`	3	$(m_{t_{\rm BC}}, m_{\ell_{\rm BC}}, m_{d_{\rm BC}})_2$	shell-following moment/length at node 2

The 1,2 nodes of each edge must run in the same direction as the counterclockwise ordering of the

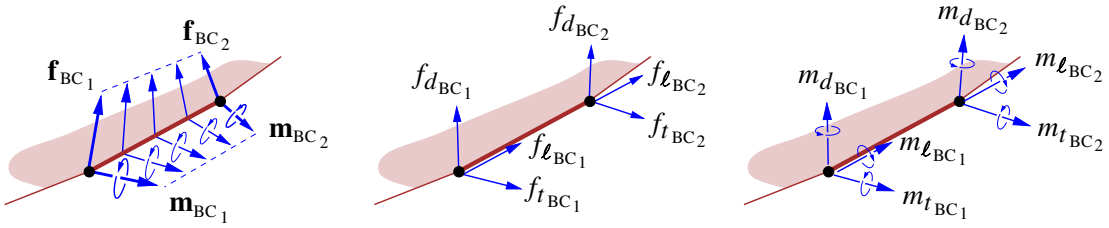


Figure 2: Total and component force and moment loads applied to an edge.

element which contains that edge. The wrong opposite ordering will in effect negate the t, ℓ loads. The overall edge loads are the sums of all the specified components,

$$\mathbf{f}_{\mathrm{BC}}(\xi) = \mathbf{f}_{xyz_{\mathrm{BC}}} + f_{t_{\mathrm{BC}}}\hat{\mathbf{t}} + f_{\ell_{\mathrm{BC}}}\hat{\mathbf{l}} + f_{d_{\mathrm{BC}}}\hat{\mathbf{d}} \quad (1)$$

$$\mathbf{m}_{\mathrm{BC}}(\xi) = \mathbf{m}_{xyz_{\mathrm{BC}}} + m_{t_{\mathrm{BC}}}\hat{\mathbf{t}} + m_{\ell_{\mathrm{BC}}}\hat{\mathbf{l}} + m_{d_{\mathrm{BC}}}\hat{\mathbf{d}} \quad (2)$$

An edge can have more than one edge BC array entry assigned to it. In this case the loads for the multiple entries will end up being added together.

5.4.2 Geometric Constraints (and Point Loads)

In typical applications, geometric constraints are imposed on boundary nodes as kinematic boundary conditions. However, HSM allows one or more geometric constraints to be applied to any node, either on a boundary or on the interior. Concentrated point loads can likewise be applied to any node. Regardless of the location or type, any such constraint will here be referred to as a "node BC", and is specified by the following pointers and `parp` data arrays.

`nbcnode`	number of nodes where node BCs are imposed
`kbcnode(l)`	`k` index of node `l` where node BCs are imposed
`lbcnode(l)`	integer which specifies which types of BCs are imposed at node `l`

array elements	num.	sym.	description
`parp(lprx:lprz,l)`	3	$\mathbf{r}_{\mathrm{BC}}$	node position
`parp(lpn1x:lpn1z,l)`	3	$\mathbf{n}^{(1)}_{\mathrm{BC}}$	1st node position restraint direction
`parp(lpn2x:lpn2z,l)`	3	$\mathbf{n}^{(2)}_{\mathrm{BC}}$	2nd node position restraint direction
`parp(lpt1x:lpt1z,l)`	3	$\mathbf{t}^{(1)}_{\mathrm{BC}}$	1st node angle restraint direction
`parp(lpt2x:lpt2z,l)`	3	$\mathbf{t}^{(2)}_{\mathrm{BC}}$	2nd node angle restraint direction
`parp(lpfx:lpfz,l)`	3	$\mathbf{F}_{\mathrm{BC}}$	node force load
`parp(lpmx:lpmz,l)`	3	$\mathbf{M}_{\mathrm{BC}}$	node moment load

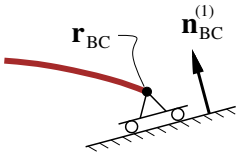
Only some of this array data is used, as indicated by the `lbcnode` compound specifier. This is computed by

$$\texttt{lbcnode} = \texttt{lbcr} + \texttt{lbcd} + \texttt{lbcf} + \texttt{lbcm}$$

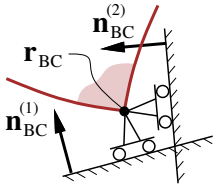
where the individual `lbcr` and `lbcd` specifiers indicate the type of position BC and/or angle BC to be imposed, respectively, and `lbcf` and `lbcm` indicate whether a point force and/or a point moment is applied. A zero specifier value indicates its particular BC or load is absent and hence its array data is not used. The nonzero choices are detailed next.

Position and angle constraints cannot be imposed on a node more than once. Specifically, any given k with nonzero `lbcr` or `lbcd` cannot appear multiple times in the `kbcnode(l)` array, otherwise the effect on the solution is unpredictable. However, force and moment loads on a node can be imposed via multiple array entries, and these entries will be summed for a net resulting load.

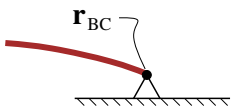
Position BCs. The following figures show the three possible position-BC types and the required BC data. The BC type is chosen by setting its `lbcr` specifier value 1, 2, or 3.



$$\mathbf{n}^{(1)}_{\mathrm{BC}} \cdot (\mathbf{r} - \mathbf{r}_{\mathrm{BC}}) \;=\; 0 \qquad \texttt{(lbcr = 1)} \tag{3}$$



$$\begin{Bmatrix} \mathbf{n}^{(1)}_{\mathrm{BC}} \cdot (\mathbf{r} - \mathbf{r}_{\mathrm{BC}}) \\ \mathbf{n}^{(2)}_{\mathrm{BC}} \cdot (\mathbf{r} - \mathbf{r}_{\mathrm{BC}}) \end{Bmatrix} \;=\; \begin{Bmatrix} 0 \\ 0 \end{Bmatrix} \qquad \texttt{(lbcr = 2)} \tag{4}$$

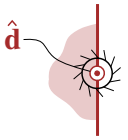


$$\begin{Bmatrix} | \\ \mathbf{r} - \mathbf{r}_{\mathrm{BC}} \\ | \end{Bmatrix} \;=\; \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix} \qquad \texttt{(lbcr = 3)} \tag{5}$$

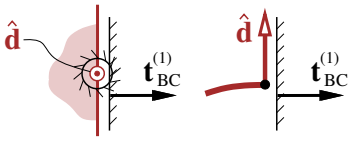
The necessary combination of $\mathbf{r}_{\mathrm{BC}}, \mathbf{n}^{(1)}_{\mathrm{BC}}, \mathbf{n}^{(2)}_{\mathrm{BC}}$ vectors must be set for each BC type. In most situations these will be the same as $\mathbf{r}_0$, $\hat{\mathbf{n}}_0$, and $\hat{\mathbf{e}}_{0_1}$ or $\hat{\mathbf{e}}_{0_2}$, although this is not required. If $\mathbf{r}_{\mathrm{BC}}$ differs from $\mathbf{r}_0$, then the structure will be forced with prescribed displacements, which is an uncommon situation.

Boundary conditions (3)–(5) restrain 1, 2, or 3 displacement degrees of freedom, respectively. For a static structural problem to be well posed, at least three non-redundant displacement restraints must be specified for the overall structure, either at one node or distributed over multiple nodes.

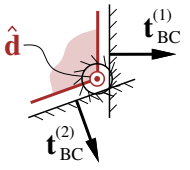
Angle BCs. The following figure shows the three possible angle-BC types and the required BC data. The BC type is chosen by setting its `lbcd` specifier value 10, 20, or 30.



$$\psi \;=\; 0 \qquad \texttt{(lbcd = 10)} \tag{6}$$



$$\begin{Bmatrix} \mathbf{t}^{(1)}_{\mathrm{BC}} \cdot \hat{\mathbf{d}} \\ \psi \end{Bmatrix} \;=\; \begin{Bmatrix} 0 \\ 0 \end{Bmatrix} \qquad \texttt{(lbcd = 20)} \tag{7}$$

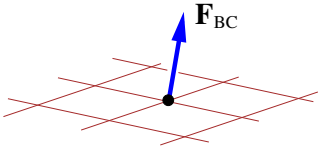


$$\begin{Bmatrix} \mathbf{t}^{(1)}_{\mathrm{BC}} \cdot \hat{\mathbf{d}} \\ \mathbf{t}^{(2)}_{\mathrm{BC}} \cdot \hat{\mathbf{d}} \\ \psi \end{Bmatrix} \;=\; \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix} \qquad \texttt{(lbcd = 30)} \tag{8}$$

Either $\mathbf{t}^{(1)}_{\mathrm{BC}}$ alone or $\mathbf{t}^{(2)}_{\mathrm{BC}}$ additionally must be set as needed. In most applications these will be the same as $\hat{\mathbf{e}}_{0_1}$ or $\hat{\mathbf{e}}_{0_2}$, although this is not required. If they are different, then the structure will be forced with prescribed angle displacements, which is an uncommon situation.

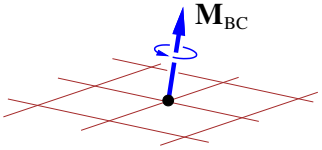
Boundary conditions (6)–(8) restrain 1, 2, or 3 rotational degrees of freedom, respectively. For a static structural problem to be well posed, at least three non-redundant rotation restraints must be specified for the overall structure. These can be imposed via some appropriate combination of position BCs at distributed nodes, or angle BCs at one or more nodes.

Point Force. If `lbcf = 100` is set, the specified $\mathbf{F}_{\text{BC}}$ force is applied to the node by adding it to the force residual for the node.



$$\mathcal{R}^f \;\leftarrow\; \mathcal{R}^f + \mathbf{F}_{\text{BC}} \qquad \texttt{(lbcf = 100)} \tag{9}$$

Point Moment. If `lbcm = 1000` is set, the specified $\mathbf{M}_{\text{BC}}$ moment is applied to the node by adding it to the moment residual for the node.



$$\mathcal{R}^m \;\leftarrow\; \mathcal{R}^m + \mathbf{M}_{\text{BC}} \qquad \texttt{(lbcm = 1000)} \tag{10}$$

5.5 Boundary condition precedence

In general, the geometric BCs "override" the loading BCs and also any joints. For example, consider a boundary node which has some specified applied force load $\mathbf{F}_{\text{BC}}$, either a point load or from distributed loads $\mathbf{f}_{\text{BC}}$ on its adjacent edges, and which is also fixed at location $\mathbf{r}_{\text{BC}}$ by the constraint $\mathbf{r}-\mathbf{r}_{\text{BC}} = \mathbf{0}$, as shown in Figure 3 on the left. In this case the load $\mathbf{F}_{\text{BC}}$ will have no effect on the structural solution, since it's effectively pushing on the "immovable" point $\mathbf{r}_{\text{BC}}$.

If the loaded node is restrained in only one direction $\mathbf{n}_{\text{BC}}$ as given by (3), then the load component in that direction $\mathbf{F}_{\text{BC}} \cdot \hat{\mathbf{n}}_{\text{BC}}$ will have no effect. Only the remaining load

$$\mathbf{F}_{\text{imparted}} \;=\; \mathbf{F}_{\text{BC}} \;-\; (\mathbf{F}_{\text{BC}} \cdot \hat{\mathbf{n}}_{\text{BC}})\; \hat{\mathbf{n}}_{\text{BC}}$$

will be imparted to the shell, as shown in Figure 3 on the right.

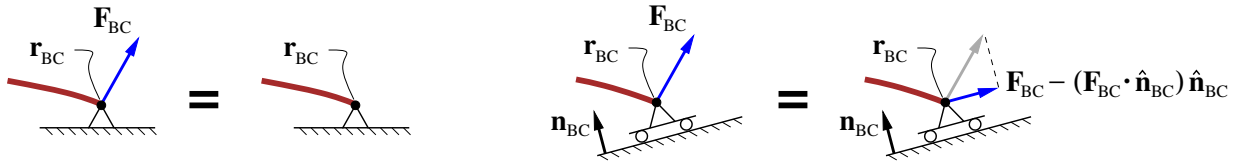


Figure 3: Complete or partial elimination of applied load by position BC.

Angle BCs cancel one or more components of an applied moment load in a similar manner. Specifically, if a moment $\mathbf{M}_{\text{BC}}$ is imposed on the node with the one-angle BC (6), then this is equivalent to an applied remaining moment

$$\mathbf{M}_{\text{imparted}} \;=\; \mathbf{M}_{\text{BC}} \;-\; \left(\mathbf{M}_{\text{BC}} \cdot \hat{\mathbf{d}}\right)\, \hat{\mathbf{d}}$$

which excludes the component along $\hat{\mathbf{d}}$. If $\mathbf{M}_{\text{BC}}$ is applied together with the two-angle BC (7), then this is equivalent to applying

$$\mathbf{M}_{\text{imparted}} \;=\; \mathbf{M}_{\text{BC}} \;-\; \left(\mathbf{M}_{\text{BC}} \cdot \hat{\mathbf{d}}\right)\, \hat{\mathbf{d}} \;-\; \left(\mathbf{M}_{\text{BC}} \cdot (\hat{\mathbf{t}}_{\text{BC}} \times \hat{\mathbf{d}})\right)\, (\hat{\mathbf{t}}_{\text{BC}} \times \hat{\mathbf{d}})$$

which excludes the components along $\hat{\mathbf{d}}$, and also the component orthogonal to both $\hat{\mathbf{t}}_{\text{BC}}$ and $\hat{\mathbf{d}}$.

5.6 Joints

Joints are imposed by setting the following variables and pointers.

`njoint`	number of joint pairs
`kjoint(1:2,j)`	`k` indices of the two nodes of joint pair `j`

In general, a physical joint between N co-located nodes requires the declaration of $N-1$ joint pairs, each of which conceptually has a *joint direction* from `kjoint(1,j)` to `kjoint(2,j)`, as indicated by the joint arrows in Figure 4.

For a physical joint with $N=2$ the single joint pair's direction is immaterial. For a physical joint with $N \geq 3$, all the joint pairs must point at the same *joint common node*, which is node k_C in the $N = 4$ example in Figure 4. The choice of the joint common node is arbitrary.

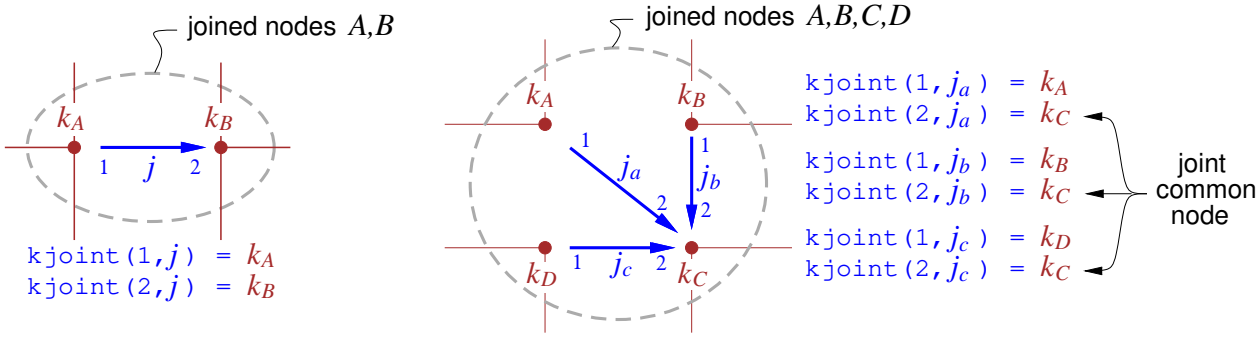


Figure 4: For physical joints with more than two nodes, as on the right, the same joint common node's index (k_C in this example) must be assigned to `kjoint(2,j)` for all the joint pairs.

Joints are not restricted to surface edge nodes. For example, a "T joint" can be made between the edge nodes of one surface and the interior nodes of another surface. A "lap joint" can be made between interior nodes of one surface and the interior nodes of another face-to-face adjoining surface, thus mimicing a surface bond. Such an isolated joint pair roughly mimics a spot weld or rivet between the two surfaces.

6 Main HSM Interface Subroutines

This section gives summary descriptions of the main HSM subroutines which will interface with the user application. The inputs and outputs of each subroutine's call list are fully described in the comment header of each subroutine, and most were already described above. Hence, the call lists will not be described here to avoid duplication. Instead, only the routine operations will be summarized.

6.1 Subroutine `HSMSOL`

This is the main top-level HSM analysis routine, which performs the following actions:

- initializes the primary variables if `lvinit = F`
- calls `HSMEQN` for each element to compute residuals and Jacobians
- accumulates residuals and Jacobians to element nodes
- imposes loading BCs by adding on boundary integrals for loaded edges `l = 1:nbcedge`
- calls `HSMBB` to compute $\hat{\mathbf{b}}_1, \hat{\mathbf{b}}_2, \hat{\mathbf{b}}_n$ projection vectors for each node
- projects residuals and Jacobians
- imposes geometric BCs by overwriting appropriate equations for BC nodes `l = 1:nbcnode`
- puts Jacobian arrays into sparse block matrix form
- solves Newton system
- performs Newton update of primary variables
- checks whether membrane sub-iterations are needed
- performs sub-iterations to convergence if needed
- checks for final convergence, and exits if below tolerance or iteration limit was reached

6.2 Subroutine `HSMDEP`

This is the main post-processing routine, which performs the following actions:

- computes the $\hat{\mathbf{e}}_1, \hat{\mathbf{e}}_2, \hat{\mathbf{n}}$ vectors at all nodes of the deformed geometry
- computes the tensors $\bar{\bar{\varepsilon}}, \bar{\bar{\kappa}}, \bar{\bar{f}}, \bar{\bar{m}}$ at all nodes in the local $12n$ axes
- computes the vectors $\vec{f}_n, \vec{\gamma}$ at all nodes in the local $12n$ axes

6.3 Subroutine `HSMOUT`

This is the main output routine which writes the solution to text files. Data for each quad element is in 5 lines, for nodes 1,2,3,4,1, thus forming a closed polygon in space. Data for each triangle element is in 4 lines, for nodes 1,2,3,1. Each of the node lines has that node's data in multiple columns, as described below. Bold vectors are in xyz axes. Italic vectors and tensors are in local $12n$ axes.

- `fort.1` : $\mathbf{r}_0 \quad \mathbf{r} \quad \hat{\mathbf{d}} \quad \phi \quad q_n \quad \mathbf{q}$
- `fort.2` : $\mathbf{r}_0 \pm \zeta\hat{\mathbf{d}} \quad \mathbf{r} \pm \zeta\hat{\mathbf{d}}$ (top/bot surfaces)
- `fort.3` : $\mathbf{r}_0 \quad \mathbf{r} \quad \bar{\bar{\varepsilon}} \quad \bar{\bar{\kappa}} \quad \vec{\gamma}$
- `fort.4` : $\mathbf{r}_0 \quad \mathbf{r} \quad \bar{\bar{f}} \quad \bar{\bar{m}} \quad \vec{f}_n$
- `fort.7` : $\mathbf{r}_0 \quad \mathbf{r} \quad \bar{f}_1 \quad \bar{f}_2 \quad \theta_{f_1}$ (principal forces and angle of $\bar{\bar{f}}$)

- `fort.8` : $\mathbf{r}_0 \quad \mathbf{r} \quad \bar{m}_1 \quad \bar{m}_2 \quad \theta_{m_1}$ (principal moments and angle of $\bar{\bar{m}}$)
- `fort.9` : $\mathbf{r}_0 \quad \mathbf{r}-\mathbf{r}_0$
- `fort.11`: $\hat{\mathbf{e}}_1$ vectors
- `fort.12`: $\hat{\mathbf{e}}_2$ vectors
- `fort.13`: $\hat{\mathbf{n}}$ vectors
- `fort.14`: $\hat{\mathbf{d}}$ vectors

7 HSM Utility Subroutines

This section gives summary descriptions of the major HSM utility subroutines which are called by the user-level routines. Normally these do not need to be called by the user's application.

7.1 Subroutine `HSMEQN`

Computes residual and primary-variable Jacobian area-integral contributions of one element to the nodes of that element. Does not include the perimeter-integral contributions $\Delta\mathcal{R}$. Handles both triangle and quad elements, indicated by `neln=3` or `neln=4`, respectively. For the `neln=3` case, the `par4` and `var4` parameters are ignored, and the `res_var4` parameters are returned as zero.

- calls `HSMABD` to rotate the $\bar{\bar{A}}, \bar{\bar{B}}\,\bar{\bar{D}}, \bar{\bar{S}}$ matrices at the nodes into the element coordinates ξ, η
- calls `HSMGEO` to compute basis vectors and other geometric quantities at each Gauss point
- interpolates node quantities to each Gauss point
- computes residual and Jacobian integrands at each Gauss point
- weight-sums the integrands to perform the Gaussian quadrature

7.2 Subroutine `HSMFBC`

Computes perimeter-integral residual contributions $\Delta\mathcal{R}$ for one edge, using boundary condition data for that edge.

- calls `HSMGEO2` to compute basis vectors and other geometric quantities at each Gauss point
- interpolates node quantities to each Gauss point
- computes residual and Jacobian integrands at each Gauss point
- weight-sums the integrands to perform the Gaussian quadrature

7.3 Subroutine `HSMABD`

Rotates the components of the $\bar{\bar{A}}, \bar{\bar{B}}, \bar{\bar{D}}, \bar{\bar{S}}$ tensors from the $12n$ axes at the element nodes, into the common ξ, η axes of the element. The result is the $\tilde{A}, \tilde{B}, \tilde{D}, \tilde{S}$ tensor components at the nodes, which can then be interpolated over the element. Handles both triangle and quad elements, indicated by `neln=3` or `neln=4`, respectively. For the `neln=3` case, the `par4` parameter is ignored.

- calls `HSMGEO` at the ξ, η location of each corner, to give the $\mathbf{a}$ basis vectors there
- applies the direction cosines $\hat{\mathbf{e}} \cdot \mathbf{a}$ at each node to perform the tensor rotations

7.4 Subroutine `HSMGEO`

Computes all the necessary geometric manifold quantities and their Jacobians, listed below. The quantities are for the specified element coordinate location $(\xi, \eta) = (\xi^1, \xi^2)$. Handles both triangle and quad elements, indicated by `neln=3` or `neln=4`, respectively. For the `neln=3` case, the `r4`, `d4`, `p4`, parameters are ignored.

- $\hat{\mathbf{d}}$ material-normal vector
- $\mathbf{a}_1, \mathbf{a}_2$ covariant basis vectors
- $\mathbf{a}^1, \mathbf{a}^2$ contravariant basis vectors
- $\breve{g}_{\alpha\beta}$ metric tensor
- $\breve{h}_{\alpha\beta}$ curvature tensor
- $\breve{g}^{\alpha\beta}$ inverse metric tensor
- $\breve{g}$ metric Jacobian
- $\partial_1\mathbf{r} \times \partial_2\mathbf{r}$ bilinear area vector

7.5 Subroutine `HSMGEO2`

Computes the geometric manifold quantities and their Jacobians for an edge segment, listed below. The quantities are for the specified line coordinate location ξ.

- $\hat{\mathbf{d}}$ material-normal vector
- $\mathbf{a}_1$ covariant basis vector
- $\breve{g}$ metric Jacobian
- $\breve{h}$ curvature along edge
- $\hat{\mathbf{t}}$ edge-normal tangent unit vector
- $\hat{\mathbf{l}}$ edge-parallel tangent unit vector