**Flagshyp Functions Edits**

Input\_data\_and\_initialisation.m (input\_reading)

Added a call to nodes\_in\_host.m to create GEOM.Embedd and GEOM.Zeta

% Identify host element/embedded node pairs and find natural

% coordinates of embedded nodes, if there are embedded elements

if ~isempty(FEM.mesh.embedded)

GEOM = nodes\_in\_host(GEOM,FEM,BC.tienodes);

end

%|-/

Explicit\_Dynamics\_algorithm.m (solution equations)

Added an update of the deformed nodal coordinates so the deformation would actually be reflected in the force calculations

% Update coodinates.

displ = disp\_n-disp\_prev;

GEOM.x = update\_geometry(GEOM.x, 1,displ(BC.freedof),BC.freedof);

Added a call to update\_embedded\_displacments

Force\_vectors\_assembly.m (global\_assembly)

In the implicit method, the element internal forces are calculated based on the incremental displacement. Then to assemble to global force vector

Force(global\_dofs,1) = Force(global\_dofs,1) + elt\_force;

In explicit, the element internal force is calculated based on the Total displacement. So in the assembly of the global force vector, the newly calculated element internal forces need to replace the previously calculated values, since they already contain that information.

function Force = force\_vectors\_assembly(elt\_force,global\_nodes,Force,dofs)

global explicit

global\_dofs = dofs(:,global\_nodes);

if explicit == 0

Force(global\_dofs,1) = Force(global\_dofs,1) + elt\_force;

else

Force(global\_dofs,1) = elt\_force;

end

end

CalculateTimeStep.m (solution equations)

The wave speed used to be calculated as sqrt (E/rho). Longitudinal bulk modulus is more accurately calculated as

ce = sqrt((lambda + 2\*mu)/rho);

**Adding Embedded Element Constraints**

1. Tie Embedded Node translations to host elements
   1. Identify embedded node constraint as boundary condition #8
      1. Boundarycodes.m (input reading)

Added case 8 to switch statement with all dof specified

case 8

dof = [1;1;1];

* + 1. Find\_fixed\_free\_dof.m (initialization)

Store global dof with boundary code 8 in vector BC.tiedof (similar to BC.fixedof and BC.freedof)

Store node numbers with boundary code 8 in BC.tienodes

% |-/

BC.tienodes = zeros(GEOM.npoin,1);

BC.tiedof = zeros(FEM.mesh.n\_dofs,1);

%

% |-/

for inode=1:GEOM.npoin

if BC.icode(inode)==8

BC.tiedof((inode-1)\*dim+(1:dim)) = [1;1;1];

BC.tienodes(inode)=inode;

end

end

BC.tiedof = BC.tiedof.\*(1:FEM.mesh.n\_dofs)';

BC.tiedof = BC.tiedof(BC.tiedof>0);

BC.tienodes = BC.tienodes(BC.tienodes>0);

%

* 1. Identify host and embedded elements
     1. ***\*\*\*Update to input file***

In element connectivity definition: row definitions

Element number, Material number, *Specify host or embedded*, Connectivity

host==0, embedded==1

* + 1. inelems.m (input\_reading)

Edit line reading in element information

Create new vectors in FEM.mesh:

embedcode – holds embedded element code (0 or 1) for each element number)

host – vector of node numbers that were specified as 0

embedded – vector of node numbers that were specified as 1

%|-/

FEM.mesh.embedcode = info(elements,3);

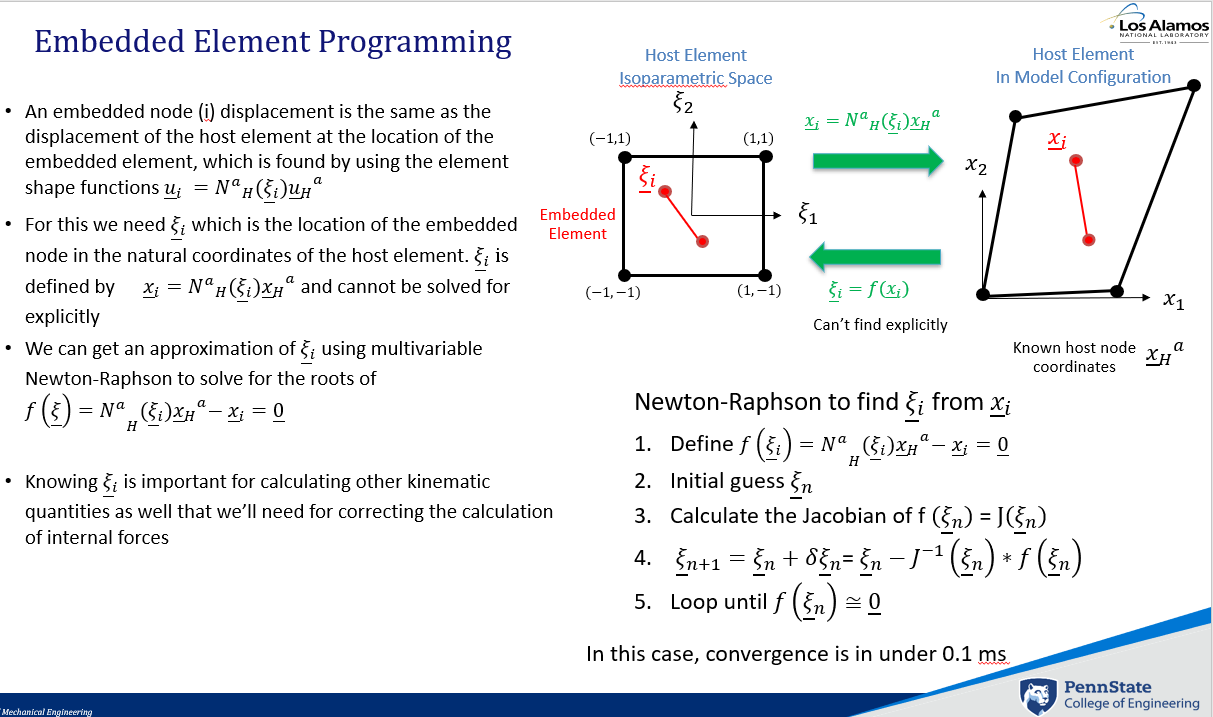
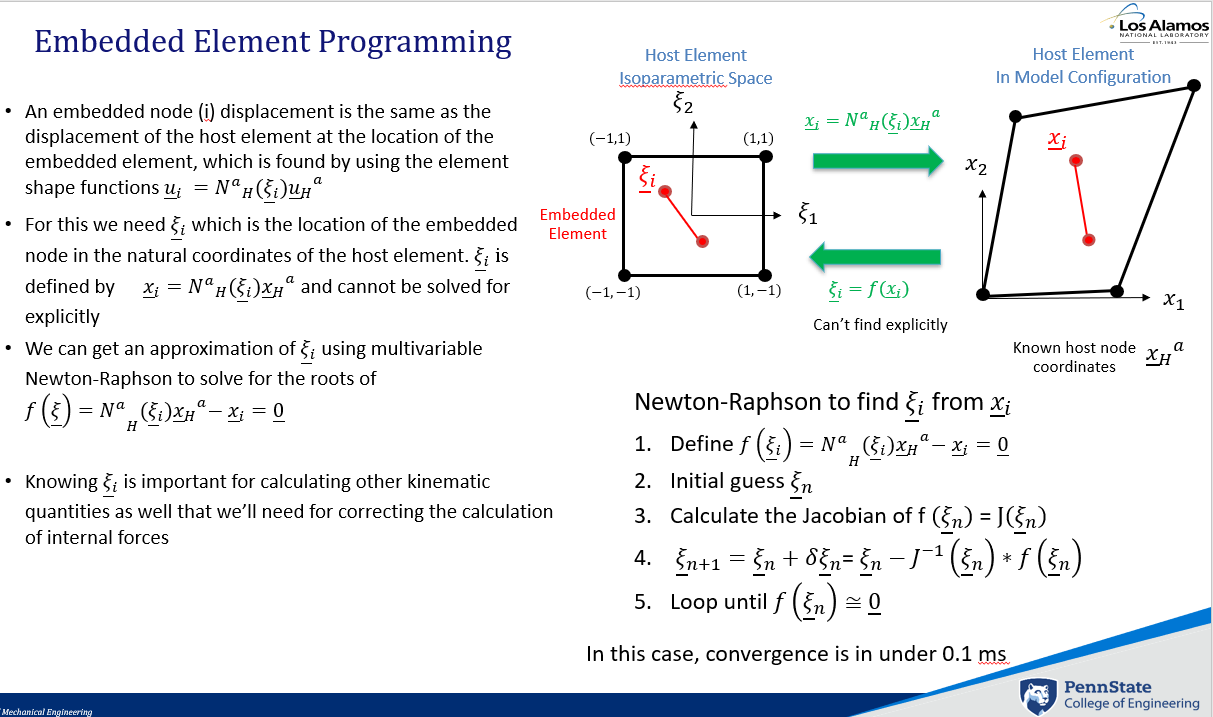
FEM.mesh.host = elements(FEM.mesh.embedcode==0);

FEM.mesh.embedded = elements(FEM.mesh.embedcode==1);

%

1. Find locations of embedded nodes wrt host elements
   1. nodes\_in\_host.m (new function)
      1. Find host element of each embedded node AND embedded element quadrature point
      2. Find coordinates of embedded nodes wrt host natural coordinates based on initial configuration x0
         1. Find\_natural\_coords.m (new function). Implimented in nodes\_in\_host.m

Newton Raphson Method to solve



1. Apply boundary condition: displacement of embedded node is equal to the displacement of the host element displacement (do this right after host mesh displacements are updated)
   1. Update\_embedded\_displacments\_explict.m (new function)

Similar to update\_perscirbed\_displacements\_explicit.m (solution update), except it updates the locations of the embedded nodes based on the displacement of their hosts

Loop through embedded nodes

1. Create vector of host element current nodal locations
2. Get of embedded element (from GEOM.Embedded\_Zeta)
3. Interpolate location of embedded element in the current host domain

Get host element from GEOM.EmbeddHost. Find xe using find\_xyz\_in\_host.m

1. Save these new embedded element locations in vector TieXUpdate (similar to Dirichlet\_dof vector)
2. Update GEOM.x (current coordinates of global nodes) with TieXUpdate

nodes\_in\_host.m

Run during initialization to add data to GEOM. Finds the host element of each embedded node using point\_in\_hexahedron.m. Embedded and Host pairs are saved in GEOM.EmbedHost (2xn\_enodes). It also uses the host/embedded pairs to find the natural coordinates of the embedded element with respect to the host element’s natural domain (find\_natural\_coords.m) and saves these coordinates in matrix GEOM.Embed\_Zeta (4xn\_enodes)

Calc\_element\_size.m (solution\_equations)

Added a max element length return for use in nodes\_in\_host.m

**Linear Bulk Viscosity Damping**

<https://abaqus-docs.mit.edu/2017/English/SIMACAEGSARefMap/simagsa-c-ovwbulkvisc.htm>

Where and are damping coefficients (decided by trial and error), is the current material density, , is the current dilatational wave speed, is the element characteristic length (largest dimension of the element), and is the volumetric strain rate.

These are bulk viscosity pressures that are added to the Cauchy stress when calculating the internal force. Linear bulk viscosity is used to damp out high frequency ringing, while quadratic smears out any shock fronts. Quadratic bulk viscosity is only applied when the element is in compression and I haven’t tested that yet.

Volumetric strain is equal to

Since I need volumetric strain rate, I approximated it by

where n is the step number.

Initialization.m (initialization)

Added vector GEOM.Jn\_1 to store the Jacobian of each element from the last step. This can’t be done in the KINEMATICS structure because KINEMATICS is overwritten multiple times each step to be used for each element. Putting Jn\_1 inside of GEOM allows it to be passed to InternalForce\_explicit.m without modifying any function arguments.

if (explicit == 1)

% GLOBAL.velocities = zeros(GEOM.npoin,GEOM.ndime);

% GLOBAL.accelerations = zeros(GEOM.npoin,GEOM.ndime);

GLOBAL.velocities = zeros(FEM.mesh.n\_dofs,1);

GLOBAL.accelerations = zeros(FEM.mesh.n\_dofs,1);

GEOM.Jn\_1 = ones(FEM.mesh.nelem,1);

end

Kinematics\_initilazation.m (initialization)

Changed the initialization of KINEMATICS.J to be all ones instead of all zeros. This way the first calculation of volumetric strain rate starts as zero.

KINEMATICS.J = ones(QUADRATURE.ngauss,1);

InternalForce\_explicit.m (global\_assembly)

Changed the function input to include dt

Changed output to return the vector GEOM.Jn\_1 (ie geomJn\_1) (this needs to be returned to the main function in order to save the previous element Jacobians)

Added calculation of the bulk viscosity terms and changed the calculation of internal force to include these pressures

Volumetric strain rate is calculated as left difference of Jacobian. Gets previous Jacobian value from GEOM.Jn\_1(ielement) (the Jacobian should be constant for the entire element, not different at different Gauss points). GEOM.Jn\_1 is also updated with the current value of the Jacobian after the internal force calculation.

Before Loop through Gauss points

KINEMATICS = gradients(xlocal,x0local,FEM.interpolation.element.DN\_chi,...

QUADRATURE,KINEMATICS);

%|-/

geomJn\_1=GEOM.Jn\_1;

Jn\_1=GEOM.Jn\_1(ielement);

Inside Gauss points loop

% Calculate bulk viscosity damping

b1=0.043; b2=1.2;

le=calc\_element\_size(FEM,GEOM,ielement);

rho=properties(1); mu=properties(2); lambda=properties(3);

J=KINEMATICS.J(igauss);

eps\_dot = (J-Jn\_1)/dt;

Cd=sqrt((lambda + 2\*mu)/rho);

p1 = rho\*b1\*le\*Cd\*eps\_dot\*CONSTANT.I;

p2 = rho\*(b2\*le)^2\*abs(eps\_dot)\*min(0,eps\_dot)\*CONSTANT.I;

Outside Gauss points loop

%Update previous Jacobian

geomJn\_1(ielement)=J;

Calc\_element\_size.m (solution\_equations)

Minor update, I just deleted the input variables that the function doesn’t actually use

getForce\_explicit.m (global\_assembly)

Changed the function input to include dt

Changed output to return the vector GEOM.Jn\_1