**Input File Format**

**Example 1**

One Hex Title

Embedded Elements 0 (yes or no)

VolumeCorrection 0 (yes or no)

1 Number of element types

hexa8 First element type

8 Number of nodes

1 2 1 1 1 Node def:

2 2 1 0 1 Node# | Boundary Code | x | y | z

3 6 1 1 0

4 6 1 0 0

5 3 0 1 1

6 3 0 0 1

7 7 0 1 0

8 7 0 0 0

1 Number of elements of the first type (ie 1 hexa8 element)

1 1 0 4 3 7 8 2 1 5 6 Element # | Material # | Host(0)/Embedded(1) | Connectivity

1 Number of materials assigned to the first type elements

1 1 7800.0 76.92e9 115.4e9 Material definition

0.01 Simulation time

2 Number of rows in amplitude ramp

0 0 Amp row 1 (time, displacement/force amplitude)

0.01 1 Amp row 2

0 4 0 0.0 0.0 0.0 Load and boundary condition information

1 2 0.04 Node number | Spatial direction | Prescribed displacement

3 2 0.04

5 2 0.04

7 2 0.04

**Example 2**

One Hex with Embedded Truss Title

Embedded Elements 1 (yes or no)

VolumeCorrection 1 (yes or no)

2 Number of element types

hexa8 First element type

truss2 Second element type

10 Number of nodes

1 2 1 1 1 Node def:

2 2 1 0 1 Node# | Boundary Code | x | y | z

3 6 1 1 0 BC 8 is for embedded nodes

4 6 1 0 0

5 3 0 1 1

6 3 0 0 1

7 7 0 1 0

8 7 0 0 0

9 8 1 1 0.6

10 8 0 1 0.6

1 Number of elements of the first type (ie 1 hexa8 element)

1 1 0 4 3 7 8 2 1 5 6 Element # | Material # | Host(0)/Embedded(1) | Connectivity

1 Number of elements of the second type

1 1 1 9 10 Element # | Material # | Host(0)/Embedded(1) | Connectivity

1 Number of materials assigned to the first type elements

1 1 7800.0 76.92e9 115.4e9 Material definition

1 Number of materials assigned to the second type elements

1 2 7800.0 2e11 0.3 0.5 2E6 1 Material definition

0.01 Simulation time

3 Number of rows in amplitude ramp

0 0 Amp row 1 (time, displacement/force amplitude)

0.005 0.9 Amp row 2

0.01 1 Amp row 3

4 0 0 0.0 0.0 0.0 Load and boundary condition information

1 0 5E9 0 Node number | Force vector

3 0 5E9 0

5 0 5E9 0

7 0 5E9 0

Element Types: The following element types (FEM.mesh.element\_type) are recognized:

truss2: 2-noded truss

hexa8: 8-noded trilinear hexahedron.

Boundary Codes:

0: free

1: x prescribed

2: y prescribed

3: x, y prescribed

4: z prescribed

5: x, z prescribed

6: y, z prescribed

7: x, y, z prescribed

8: embedded node

Prescribed degrees of freedom are assumed to be fixed (no displacement) unless otherwise prescribed to be different from zero in the load and boundary condition information.

Material Definitions:

1: plane strain or three-dimensional compressible neo-Hookean

2: one-dimensional stretch-based hyperelastic plastic (truss2 only)

3: plane strain or three-dimensional hyperelastic in principal directions

4: plane stress hyperelastic in principal directions

5: plane strain or three-dimensional nearly incompressible neo-

Hookean

6: plane stress incompressible neo-Hookean

7: plane strain or three-dimensional nearly incompressible hyperelas

ticity in principal directions

8: plane stress incompressible hyperelasticity in principal directions

17: plane strain or three-dimensional nearly incompressible hyperelastic

plastic in principal directions.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Type** | props(1) | props(2) | props(3) | props(4) | props(5) | props(6) |
| 1 | *ρ* | *μ* | *λ* | – | – | – |
| 2 | *ρ* | *E* | *ν* | *area* | *τy* | *H* |
| 3 | *ρ* | *μ* | *λ* | – | – | – |
| 4 | *ρ* | *μ* | *λ* | *h* | – | – |
| 5 | *ρ* | *μ* | *κ* | – | – | – |
| 6 | *ρ* | *μ* | *h* | – | – | – |
| 7 | *ρ* | *μ* | *κ* | – | – | – |
| 8 | *ρ* | *μ* | *h* | – | – | – |
| 17 | *ρ* | *μ* | *λ* | – | *τy* | *H* |

In this table, *ρ* represents the density in the reference configuration, *λ* and *μ* are the Lamé coefficients, *κ* = *λ* + 2*μ/*3 is the bulk modulus,*∗ h* is the thickness for plane stress cases, *E* is the Young’s modulus, *ν* is Poisson’s ratio, and *area* is the initial cross-sectional area. Finally, *τy* and *H* are the yield stress and hardening parameter, respectively.

Applied Load and Displacements:

Load and displacements on the nodes are applied in a ramped fashion, the ramp is defined by a table below the total simulation time. The number of rows in the table must be specified before the table.

For the load and boundary conditions information line enter:

Number of loaded nodes | Number of nonzero prescribed displacements | Number of surface elements with pressure | Gravity Vector

If both loads and boundary conditions are specified, loaded nodes are listed first. All applied displacements must be specified here as well as indicated in the nodal information as a prescribed DOF.

**ExplicitVolumeCorrectionEmbededFiniteElement.m**

Most of the important problem information is stored in a set of Structures:

BC – Boundary conditions (degrees of freedom, prescribed displacements)

CON – Some numerical stuff and output stuff

CONSTANT- Convenient identity tensors

DAMPING – The two bulk viscosity damping coefficients

FEM\* - Mesh information (number of element types) Two substructures

Mesh – (element type, # nodes of the element, element connectivity)

Interpolation – Shape functions and derivatives of the shape functions evaluated at the element’s Gauss Quadrature points. This has two of its own substructures: element and boundary. Boundary is only used when calculating forces over an area (pressure forces)

GEOM – Problem geometry (number of nodes, number of dimensions, initial nodal coordinates, current nodal coordinates, element volume, etc)

GLOBAL – Nodal forces, accelerations, and velocities expressed in vectors in the problem global coordinate system

KINEMATICS\* - This on is odd in that it’s always used as temporary variables. Each time the KINEMATICS structure is used, gradients.m or some other function is called to recalculated the values of KINEMATICS based on the current deformation

LOAD – Applied external loads (gravity, pressure)

MAT\* - Material properties. Each MAT structure represents all of the material types that are used for one element type (ie if there are hexa8 and truss2 elements, even if they use the same material, it is defined in each structure)

PLAST\*- Plasticity. I haven’t done much with this so your guess is as good as mine

PRO – Input/Output/Restart file names, folder names etc.

QUADRATURE\* - Definitions for Gauss Quadrature. (3 Gauss points, Gauss point weights, local Gauss point coordinates). Two substructures: element and boundary. Similar to FEM.interpolation, boundary is only used in surface calculations.

\* denotes this is an array of structures when multiple element types are used. Each element in the array is a single structure for that element type

1. **Input\_data\_and\_initilaization.m** (input\_reading)

Welcome just fills out names of things for output, not that interesting

1. **Reading\_input\_file.m** (input\_reading)

Reads input. It runs the following functions.

* 1. **Elinfo.m** (input\_reading)

Reads in info about the element type and creates structures FEM\* and QUADRATURE based on that information. If there are multiple element types in an input file, FEM and QUADRATURE will be arrays of structures in the main program

* 1. **Element\_quadrature\_rules.m** (numerical\_integration)

Gets values for QUADRATURE\* based on element type

* 1. **Shape\_functions\_iso\_derivs.m** (FEM\_shape\_functions)

Fills in FEM.interpolation using **shape\_functions\_library.m** (FEM\_shape\_functions)

* 1. **Innodes.m** (input\_reading)

Reads node info (# of nodes, boundary codes, node coordinates) and assigns the data to the appropriate struct

* 1. **Inelems.m** (input\_reading)

Reads element info for each element type separately and assigns it to FEM\*

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

Based on the boundary codes, determines the global degrees of freedom that are prescribed (applied displacement or load), free (to be calculated), or tied (embedded element constraint)

* 1. **Matprop.m** (input\_reading)

Reads in material info for each element type to be stored in MAT\*

* 1. **Inloads**.**m** (input\_reading)

Reads in external applied loads/pressures.

1. **Constant\_entities.m** (support)

Literally just makes any identity matrices you need

1. **Initialisation.m** (initialization)

Initializes kinematic variables, equivalent force vector, and computes the initial tangent matrix (not needed for explicit, so I may get rid of it)

* 1. **Kinematics\_initialisation.m** (initialization)

Creates KINEMATICS struct for each element type

* 1. **Initial\_volume.m** (initialization)

Calculates the volume of each element. GEOM.Ve is a matrix, row is element number, column is element type.

* 1. **Gravity\_vector\_assembly.m** (global\_assembly)

I don’t anticipate using gravity soon so I’ll just comment this out.

* 1. **Residual\_and\_stiffness\_assembly** (global\_assembly)

This function is for implicit methods. I’m just replacing it by setting initial internal force and residual to zero.

* 1. **Inverse\_mapping.m** (embeddd\_element\_functions)

Calculates the host element coordinates of embedded elements and creates NodeHost: list of nodes and their associated host, and ElementHost: list of elements and their associated host. Host totals: list of host nodes and their total number of embedded elements and embedded nodes Embed\_Zeta: list of embedded node coordinates in the natural coordinate system of their respective host

*From this point, assume all elements of the first type are hosts and all of the second type are embedded. This will make programming easier for now.*

* + 1. **Point\_in\_hexahedron.m** (embedded\_element\_functions)

Checks if a given point is inside of a hexahedron defined by 8 node points

* + 1. **Find\_natural\_coords.m** (embedded\_element\_functions)

Newton Rapson iteration to return the solution to [N][x]-[a]=[0], where [a] is the global coordinates of a node and [x] is the isoperimetric coordinates of that node in a host hexahedral element defined by shape functions [N].

**4.5.3. find\_intersection** (embedded\_element\_functions)

Finds the point between two host elements where the shared embedded element crosses

* 1. **Mass\_assembly.m** (global\_assembly)

Computes and assembles the lumped mass matrix

* 1. **Effective\_mass\_assembly.m** (global\_assembly)

Computes and assembles the lumped mass matrix, with the option to remove redundant mass when embedded elements are used. GLOBAL.M is used for calculating accelerations on the host nodes (the masses here include the distributed masses of the embedded elements) while GLOBAL.M\_KE is used to calculate kinetic energy where the host nodes only have the mass of the host elements and the embedded nodes carry their own mass (because these nodes will have different velocities)

1. **Save\_restart\_file.m** (solution\_write)

Initialize restart file

1. **Output\_vtu.m** (solution\_write)

Write first output files (OUTPUT.txt and .vtu files).

1. **ExplicitDynamics\_algorithm.m** (solution\_equations)

This is the main part of the explicit loop. Currently the entire function is part of Frankenstein for easy access. The explicit loop follows the steps in Beltyshenkos or how’s it spelled

* 1. **Get\_Force\_explicit.m** (global\_assembly)

Calculates forces (internal and reaction) on nodes.

* + 1. **Element\_force\_truss.m** (element\_calculations)

Calculates nonlinear stress and strain for a 1D truss. Updated version of element\_force\_and\_stiffness\_truss.m that does not calculate/return truss stiffness

* + 1. **InternalForce\_explicit.m** (element\_calculations)

Gauss quadrature to find the stress in a continuum element. Modified to included artificial viscosity damping based on strain rate.

* + - 1. **TrussCorrectedInternalForce\_explicit.m** (embedded\_element\_functions)

This gets the internal force of the embedded element that is applied to the host element and, if volume correction is turned on, will calculate the amount of internal force needed to correct for the extra strain energy in the host element. The counterpart to this,TrussCorrectedInternalForce\_wShear\_explicit.m attempts to return a 3D stress for the truss element to correct for the loss of shear strength in the host, but this doesn’t actually work

* 1. **CalculateTimeStep.m** (solution\_equations)

Calculates the maximum stable time increment based on element size, sound speed, and damping. Only checks the host element because the embedded element really aren’t degrees of freedom that come into the dynamic problem.

* + 1. **Calc\_element\_size.m** (solution\_equations)

Finds the longest side length of an element to determine the element characteristic length

* 1. **Update\_geometry\_explicit.m** (solution\_update)

Takes the calculated displacement and updates the current nodal coordinates

* 1. **External\_force\_update\_explicit.m** (global assembly)

Increments the applied external force using a smooth ramp over the total time

* 1. **Update\_prescibed\_displacements\_explicit.m** (solution\_update)

Applies boundary condition prescribed displacement. Displacement is prescribed in linear increments

* 1. **Update\_embedded\_displacements\_explicit.m** (embedded\_element\_functions)

Uses host element current coordinates to interpolate (via host element shape functions) the new positions of the embedded elements. Also interpolates host node velocities and accelerations to get embedded node velocities and accelerations

* + 1. **Shape\_function\_values\_at** (embedded\_embedded\_displacement)

Returns the shape function values at a particular isoparametric coordinate for a truss or hex element

* 1. **Get\_Force\_explicit.m** (global assembly)
     1. **InternalForce\_explicit.m** (element\_calculations)
        1. **TrussCorrectedInternalForce\_explicit.m** (embedded\_element\_functions)
  2. **CalculateTimeStep.m** (solution\_equations)

See 7.2

* 1. **Update\_embedded\_displacements\_explicit.m** (embedded\_element\_functions)
  2. **Check\_energy\_explicit.m** (convergence\_check)

Calculates total internal work (disp\*force), external work (applied disp\*force), kinetic energy (1/2 mv^2), and the viscous dissipation energy. For embedded elements, displacement energy is based on only the host nodes with the embedded element mass included. Kinetic energy is calculated for each individual node with only its own mass

* 1. **Output.m** (solution\_write)

Changed a lot of the output function. The output file is different than original Flagshyp. It has more information but is also easier to read without referencing the documentation all of the time

* + 1. **Stress\_output.m** (solution\_write)

Calculates stress in each element. Modified to also output log strain.

* + - 1. **Element\_force\_truss.m** (element\_calculations)

Updated from element\_force\_and\_stiffness.m to include a calculation of log strain and not calculate stiffness

* + - 1. **Essentially just getForce\_explicit.m and InternalForce\_explicit.m**

Updated this to include a calculation of log strain

* 1. **Output\_vtu.m** (solution\_write)

**List of all folders and their functions**

**FEM\_shape\_functions**

shape\_functions\_iso\_derivs.m

shape\_functions\_library.m

shape\_functions\_library\_boundary.m

**constitutive laws**

Cauchy\_type\_selection.m

elasticity\_modulus\_selection.m

muab\_choice.m

elasticity\_tensor

ctens1.m

ctens17.m

ctens3.m

ctens4.m

ctens5.m

ctens6.m

ctens7.m

ctens8.m

stress

stress1.m

stress17.m

stress3.m

stress4.m

stress5.m

stress6.m

stress7.m

stress8.m

**convergence\_check**

check\_energy\_explicit.m

check\_residual\_norm.m

**element\_calculations**

InternalForce\_explicit.m

constitutive\_matrix.m

element\_force\_truss.m

element\_gravity\_vector.m

element\_gravity\_vector\_truss.m

mean\_dilatation\_pressure.m

mean\_dilatation\_pressure\_addition.m

mean\_dilatation\_volumetric\_matrix.m

pressure\_element\_load\_and\_stiffness.m

pressure\_load\_matrix.m

pressure\_load\_stiffness\_vector.m

**embedded\_element\_functions**

find\_natural\_coords.m

find\_intersection.m

inverse\_mapping.m

nodes\_in\_host.m

point\_in\_hexahedron.m

shape\_function\_values\_at.m

TrussCorrectedInternalForce\_explicit.m

TrussCorrectedInternalForce\_explicit\_from\_mem.m

update\_embedded\_displacements\_explicit.m

**global\_assembly**

effective\_mass\_assembly.m

external\_force\_update.m

external\_force\_update\_explicit.m

force\_vectors\_assembly.m

getForce\_explicit.m

getForce\_parallel.m

gravity\_vector\_assembly.m

lumped\_mass\_assembly.m

mass\_assembly.m

pressure\_load\_and\_stiffness\_assembly.m

residual\_and\_stiffness\_assembly.m

residual\_assembly\_explicit.m

**initialisation**

find\_fixed\_free\_dofs.m

initial\_volume.m

initialisation.m

kinematics\_initialisation.m

**input\_reading**

boundary\_codes.m

elinfo.m

incontr.m

inelems.m

inloads.m

innodes.m

input\_data\_and\_initialisation.m

matprop.m

reading\_input\_file.m

welcome.m

**kinematics**

gradients.m

gradientsTruss.m

isoparametric\_gradients.m

kinematics\_gauss\_point.m

normal\_vector\_boundary.m

thickness\_plane\_stress.m

**numerical\_integration**

edge\_quadrature\_rules.m

element\_quadrature\_rules.m

**plasticity**

Von\_Mises\_yield\_function.m

plasticity\_initialisation.m

plasticity\_storage.m

plasticity\_update.m

radial\_return\_algorithm.m

selecting\_internal\_variables\_element.m

**solution\_equations**

Arc\_Length\_Newton\_Raphson\_algorithm.m

CalculateTimeStep.m

ExplicitDynamics\_algorithm.m

Line\_Search\_Newton\_Raphson\_algorithm.m

Newton\_Raphson\_algorithm.m

arclen.m

calc\_element\_size.m

calc\_max\_element\_size.m

calc\_min\_element\_size.m

linear\_solver.m

search.m

**solution\_update**

update\_geometry.m

update\_geometry\_explicit.m

update\_prescribed\_displacements.m

update\_prescribed\_displacements\_explicit.m

**solution\_write**

output.m

output\_textfile.m

output\_vtk.m

output\_vtu.m

plot\_Eulerian\_strain.m

plot\_Lagrangian\_strain.m

plot\_lnV.m

plot\_stresses.m

save\_output.m

save\_restart\_file.m

stress\_output.m

stress\_output\_from\_mem.m

write\_energy\_output.m

**support**

Levi\_civita\_contraction\_vector.m

constant\_entities.m