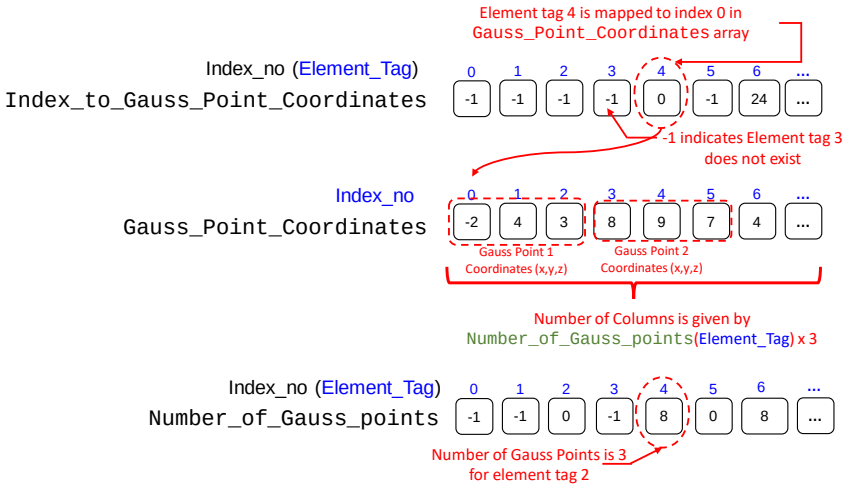


Figure 206.15: Arrays describing gauss coordinates information in `Elements` group directory of HDF5 file.

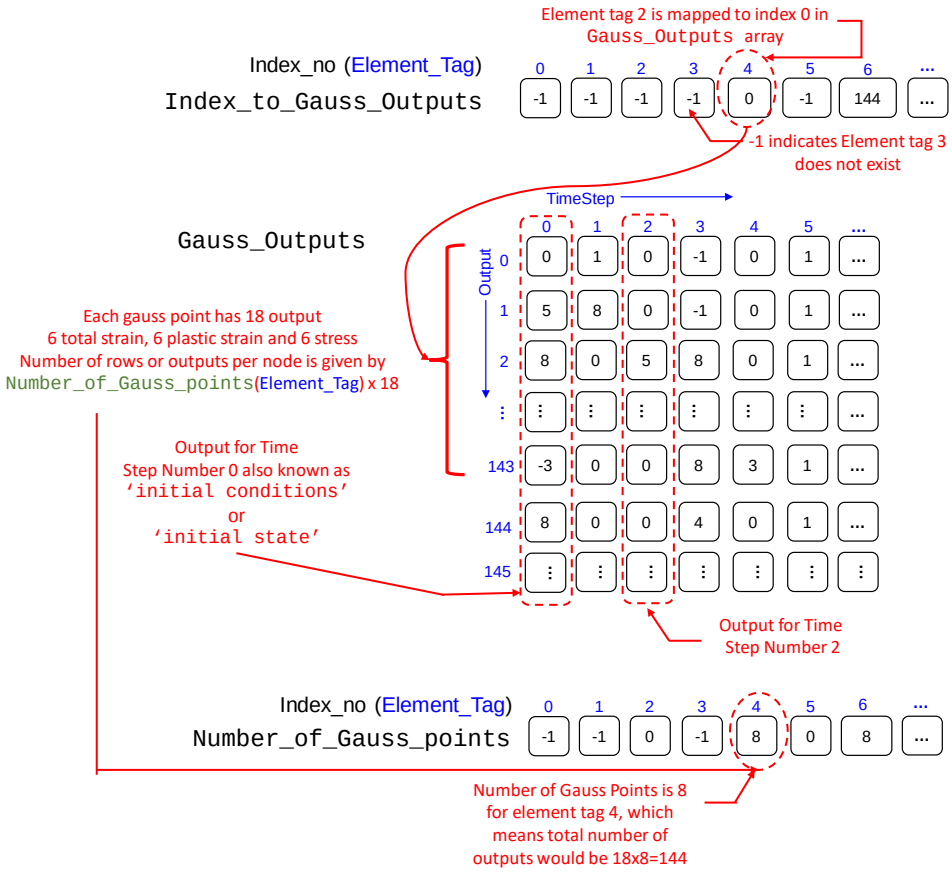


Figure 206.16: Arrays describing gauss output information in `Elements` group directory of HDF5 file.

total length of gauss output for that element is $8 \times 18 = 144$. The index from which the coordinates information start is 0. gauss values for first 18 index $(0, 1..17)$ corresponds to gauss point 1 and next 18 index $(18..35)$ corresponds to gauss point 2 and so on. For gauss point 1, first 6 index $(0, 1..5)$ corresponds to total strain, next 6 index $(6..11)$ corresponds to plastic strain and next 6 index $(12..17)$ corresponds to stress. Section 206.7 describes how gauss output is stored and what are the units and meaning of those outputs.

206.5.4.10 `Element_Outputs`

`Element_Outputs` array stores output for the elements except those stored at gauss points (i.e. stress, total strain and plastic strain). It is a 2D `float` array indexed by the `Index_to_Element_Outputs` array and the number of rows is defined `Number_of_Element_Outputs`. he column index is represented by the time step of the simulation. Time index 0 represents initial state conditions, i.e. the state before the start of new stage and end of previous stage.

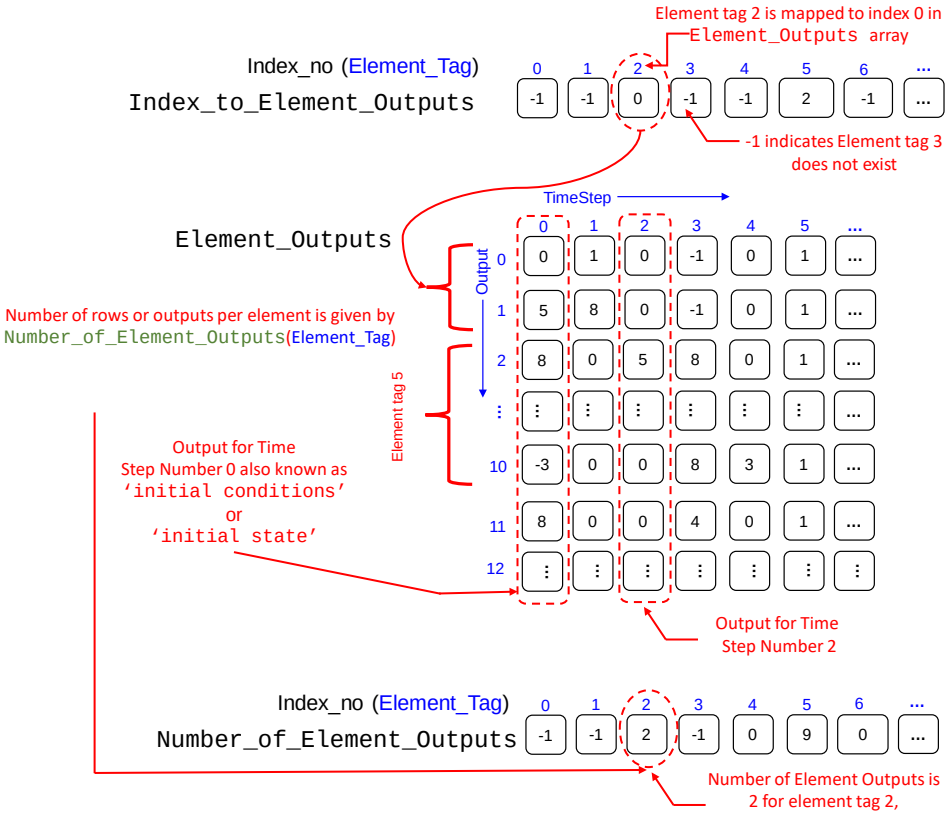


Figure 206.17: Arrays describing output information in `Elements` group directory of HDF5 file.

Figure 206.17 shows how to read `Element_Outputs` array for a particular element and particular

time step. In the given example Listing 206.2, element tag 2 has two outputs. Similarly, element tag 5 has 9 outputs. On the other hand element tag 4 has no output. For element 2, the output starts at row index 0 and ends at 1. Similarly, for element tag 4, the output is stored in row index 2..10.

Since the `Element_Outputs` array format depends on the elements present in the model, one must refer to each element specifically (Section 206.8) to identify what each output component means.

206.5.5 The `Physical_Groups` group

`Physical_Groups` group contains the physical groups of nodes or elements defined in the analysis. It contains two subgroups: `Physical_Node_Groups` and `Physical_Element_Groups` as shown in Figure 206.18 and is described in the following sections.

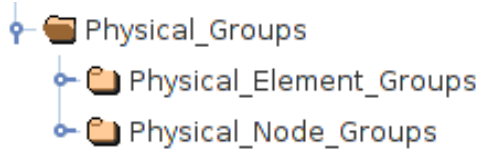


Figure 206.18: `Physical_Groups` group directory of HDF5 file.

For example, lets imagine that the user has defined one phyical group of nodes and elements respectively as shown below :

Listing 206.3: PhyGrp_Example

```
1 // defining physical groups
2 define physical_node_group ``Physical_Node_Group#1'';
3 define physical_element_group ``Physical_Element_Group#1'';
4
5 // adding items to allready defined physical groups
6 add nodes (1,4,5,7,2,30,42) to physical_node_group ``Physical_Node_Group#1''
7 add elements (1,4,5,7,2,30,42) to physical_node_group ``Physical_Element_Group#1''
```

The data arrays for the given example would look like the following as shown in the subsections ahead.

206.5.5.1 The `Physical_Element_Groups`

This group contains information about physical groups of elements defined in the analysis. For each of the physical group defined, a new `integer` data array is created inside `Physical_Element_Groups` which stored the element tags belonging to that group. Figure 206.19 shows how to read `Physical_Element_Groups` dataset array for a particular defined physical group.

In the given example Listing 206.3, a physical group array with name ‘‘Physical_Element_Group#1’’ is created which contains the list of element tags that were part of that physical group. Figure 206.19 shows that ‘‘Physical_Element_Group#1’’ contains element tags 1,4,5,7, and so on;

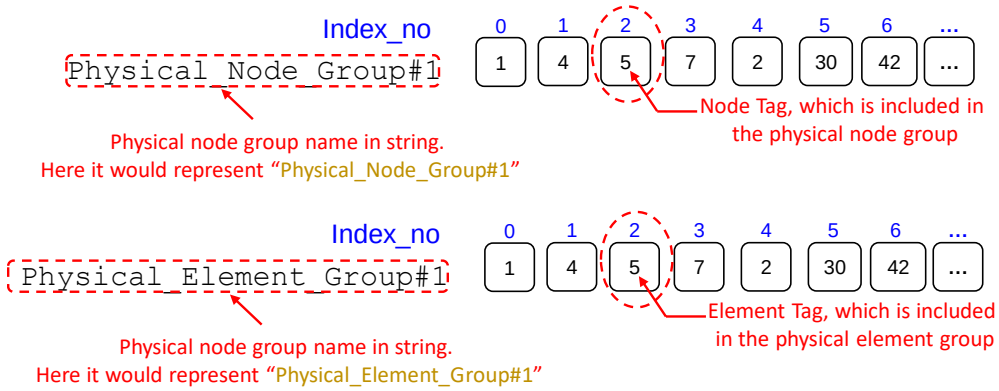


Figure 206.19: Arrays describing physical groups information in Physical_Groups group directory of HDF5 file.

206.5.5.2 The Physical_Node_Groups

This group contains information about physical groups of nodes defined in the analysis. For each of the physical group defined, a new integer data array is created inside Physical_Node_Groups which stored the node tags belonging to that group. Figure 206.19 shows how to read Physical_Node_Groups dataset array for a particular defined physical group.

In the given example Listing 206.3, a physical group array with name ‘‘Physical_Node_Group#1’’ is created which contains the list of node tags that were part of that physical group. Figure 206.19 shows that ‘‘Physical_Node_Group#1’’ contains node tags 1,4,5,7, and so on;

206.5.6 The Eigen_Mode_Analysis group

Eigen_Mode_Analysis group gets created in HDF5 .feioutput file after an eigen mode analysis. The data arrays available inside this group are described below and is also shown in Figure 206.20.

206.5.6.1 Number_of_Eigen_Modes

Number_of_Modes is an integer that stores information about the number of modes solved for the eigen problem. Figure 206.21 shows how to read it.

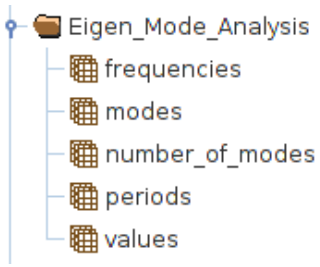


Figure 206.20: `Eigen_Mode_Analysis` group directory of HDF5 file.

206.5.6.2 `Eigen_Frequencies`

`Frequencies` is an `float` array that stores the natural frequencies of the model corresponding to different modes. The length of the array is number of modes + 1. Index '0' does not correspond to any eigen mode and thus stores -1. Figure 206.21 shows how to read it.

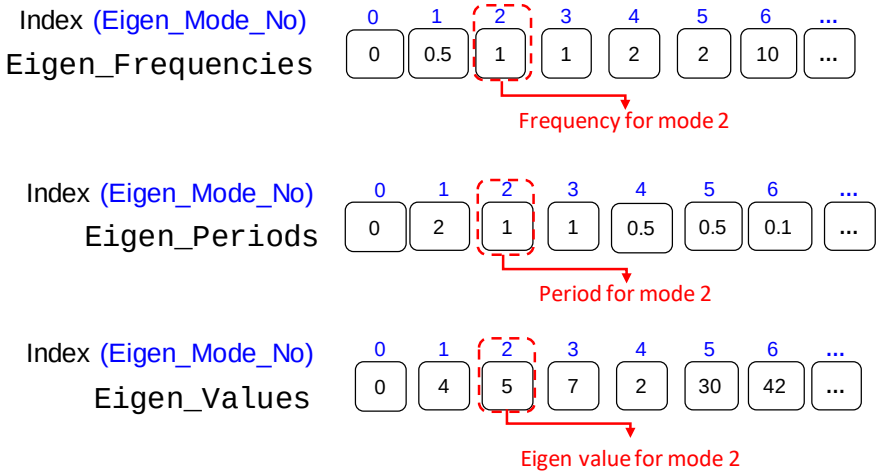


Figure 206.21: `Eigen_Mode_Analysis` group directory of HDF5 file.

206.5.6.3 `Eigen_Periods`

`Frequencies` is an `float` array that stores the natural periods of the model corresponding to different modes. The length of the array is number of modes + 1. Index '0' does not correspond to any eigen mode and thus stores -1. Figure 206.21 shows how to read it.

206.5.6.4 `Eigen_Values`

`Frequencies` is an `float` array that stores the natural eigen values of the model corresponding to different modes. The length of the array is number of modes + 1. Index '0' does not correspond to any

eigen mode and thus stores -1. Figure 206.21 shows how to read it.

206.5.6.5 Modes

Modes is a 2-D float array that stores the generalized displacements of the nodes defined in the model corresponding to different modes. The column index no. n this 2-D array defines the mode no i.e. column index 1 corresponds to mode number 1 and so on. Figure ?? shows how to read it.

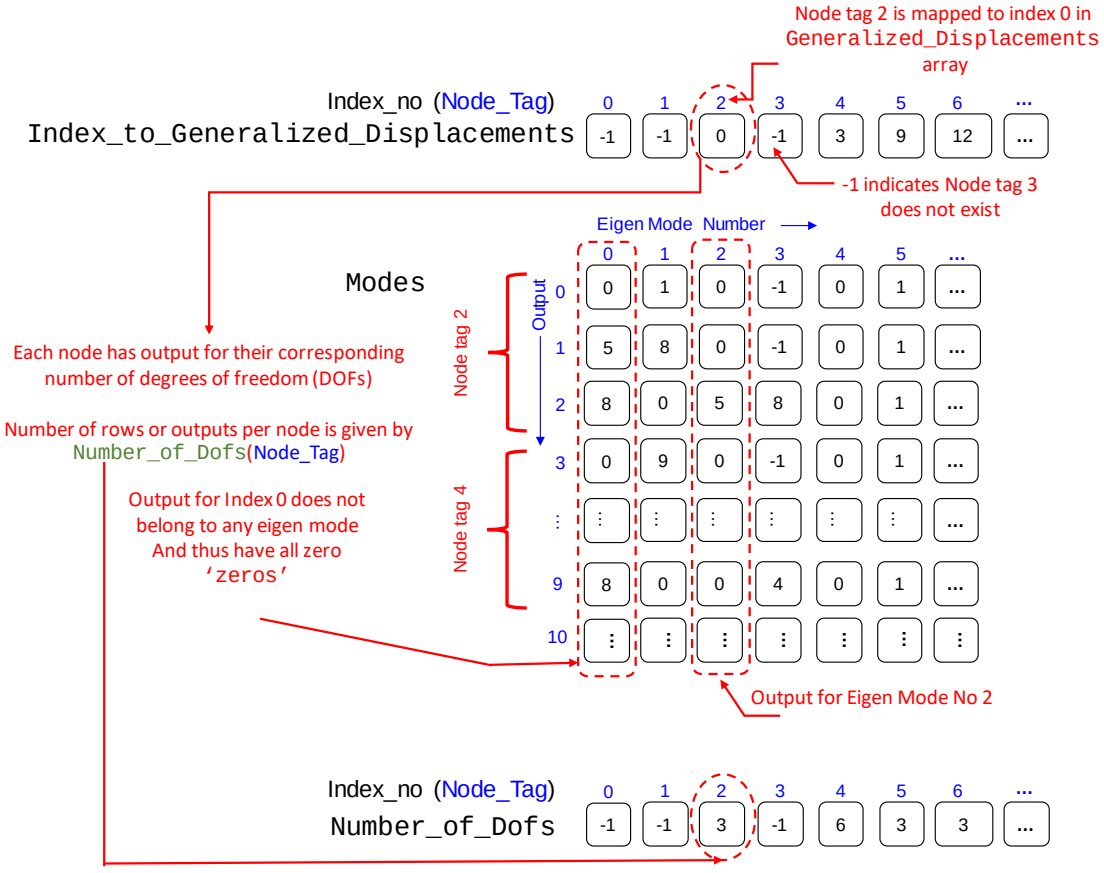


Figure 206.22: Eigen_Mode_Analysis group directory of HDF5 file.

206.5.7 The Material data array

Material is a 1-D string array that stores information about all materials defined in the model. The length of the array is the maximum material tag + 1 (including tag 0). The index of this array corresponds to the material tag. A value of '-1' means that the material tag was not defined. Figure 206.23 shows the Material data array for the example Listing 206.2. Here, index no 1 and index no 2 stores the information corresponding to material tag 1 and 2 respectively.

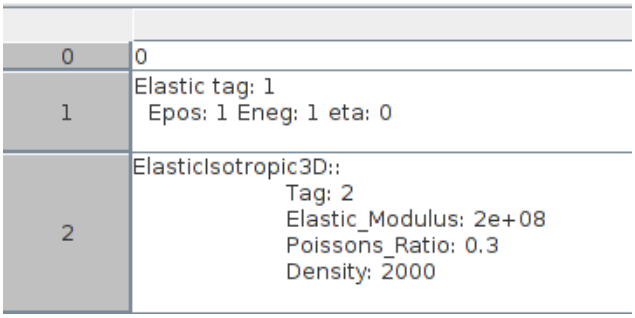


Figure 206.23: `Materials` data array in `Model` directory of HDF5 file.

206.6 Node-specific output format

In Real-ESSI simulator, nodes can be defined with different number of degree of freedoms (DOFs). Nodes with with different dofs can be thought of different types of nodes. As a result, their corresponding output also changes. Here is described, all the different node types available and the output format and their descriptions for each of them. The following subsections would describe the definition of outputs that are expected in `Generalized_Displacements` and `Support_Reactions` data arrays in `Nodes` group of HDF5 output file for node tags of different dof types.

206.6.1 3DOF

Nodes defined with 3DOF type has u_x, u_y, u_z degrees of freedom. They correspond to the displacement degrees of freedom in x, y and z direction respectively. The dof id, generalized_displacement and support reactions for 3dof type node are summarized in the Table 206.6.1.

DOF Id	Description	`Generalized_Displacements`	`Support_Reactions`
0	disp. in x-dir	$u_x[m]$	$F_x[N]$
1	disp. in y-dir	$u_y[m]$	$F_y[N]$
2	disp. in z-dir	$u_z[m]$	$F_z[N]$

Table 206.3: DOF id and output for 3DOF type node

206.6.2 4DOF

Nodes defined with 3DOF type has u_x, u_y, u_z, p degrees of freedom. They correspond to the displacement degrees of freedom in x, y and z direction and pressure respectively. The dof id, generalized_displacement and support reactions for 4dof type node are summarized in the Table 206.6.2. up elements have nodes

with 4 dofs.

DOF Id	Description	Generalized_Displacements	Support_Reactions
0	disp. in x-dir	$u_x[m]$	$F_x[N]$
1	disp. in y-dir	$u_y[m]$	$F_y[N]$
2	disp. in z-dir	$u_z[m]$	$F_z[N]$
3	pressure	$p[Pa]$	$p[Pa]$

Table 206.4: DOF id and output for 4DOF type node

206.6.3 6DOF

Nodes defined with 6DOF type has $u_x, u_y, u_z, r_x, r_y, r_z$ degrees of freedom. They correspond to the displacement and rotational degrees of freedom in x, y and z direction respectively. The dof id, generalized_displacement and support reactions for 6 dof type node are summarized in the Table 206.6.3. Beam and Shell elements have nodes with 6 dofs.

DOF Id	Description	Generalized_Displacements	Support_Reactions
0	disp. in x-dir	$u_x[m]$	$F_x[N]$
1	disp. in y-dir	$u_y[m]$	$F_y[N]$
2	disp. in z-dir	$u_z[m]$	$F_z[N]$
3	rotation about x-axis	$r_x[radian]$	$M_x[N-m]$
4	rotation about y-axis	$r_y[radian]$	$M_y[N-m]$
5	rotation about z-axis	$r_z[radian]$	$M_z[N-m]$

Table 206.5: DOF id and output for 6DOF type node

206.6.4 7DOF

Nodes defined with 6DOF type has $u_x, u_y, u_z, p, U_x, U_y, U_z$ degrees of freedom. They correspond to the solid-displacement , pore-fluid pressure and and fluid-displacement in x, y and z direction respectively. The dof id, generalized_displacement and support reactions for 7dof type node are summarized in the Table 206.6.3. upU elements have nodes with 7 dofs.

DOF Id	Description	Generalized_Displacements	Support_Reactions
0	solid disp. in x-dir	$u_x[m]$	$F^s{}_x[N]$
1	solid disp. in y-dir	$u_y[m]$	$F^s{}_y[N]$
2	solid disp. in z-dir	$u_z[m]$	$F^s{}_z[N]$
3	pore-pressure	$p[Pa]$	$p[Pa]$
4	fluid disp. in x-dir	$U_x[m]$	$F^f{}_x[N]$
5	fluid disp. in y-dir	$U_y[m]$	$F^f{}_y[N]$
6	fluid disp. in z-dir	$U_z[m]$	$F^f{}_z[N]$

Table 206.6: DOF id and output for 7DOF type node

206.7 Element-gauss output format

`Gauss_Outputs` array stores the coordinates of the gauss points declared inside for each element defined in the model. It is a 2D `float` array indexed by the `Index_to_Gauss_Outputs` array and the number of rows is defined by 18 (6 total strain, 6 plastic strain and 6 stress) times `Number_of_Gauss_Points`. The column index is represented by the time step of the simulation. Time index 0 represents initial state conditions, i.e. the state before the start of new stage and end of previous stage.

Figure 206.24 shows how to read `Gauss_Outputs` array for a particular element. In the given example Listing 206.2, element tag 2 and 5 has no gauss points. Since, element tag 4 has 8 gauss points, the total length of gauss output for that element is $8 \times 18 = 144$. The index from which the coordinates information start is 0. gauss values for first 18 index $(0, 1..17)$ corresponds to gauss point 1 and next 18 index $(18..35)$ corresponds to gauss point 2 and so on. For gauss point 1, first 6 index $(0, 1..5)$ corresponds to total strain, next 6 index $(6..11)$ corresponds to plastic strain and next 6 index $(12..17)$ corresponds to stress. Table 206.7 shows how data is stored for each gauss points. The strains are unit less and stress have unit of Pascal Pa.

In the Table 206.7 `start` corresponds to the starting position for the elements output as determined with reference to the `Index_to_Element_Outputs` for the element of interest. To this number we add the corresponding `offset` as determined by each table below and interpret the row according to the meaning established below.

For each gauss outputs there are 6 components of strain tensor, 6 components of plastic-strain tensor and 6 components of stress tensor. This makes a total of $3 \times 6 \times NumGauss$ rows of output per element. The specific meaning of the rows is as follows. For Gauss-point 1 (with starting index given in the `Index_to_Gauss_Outputs` array), the outputs are stored as the following

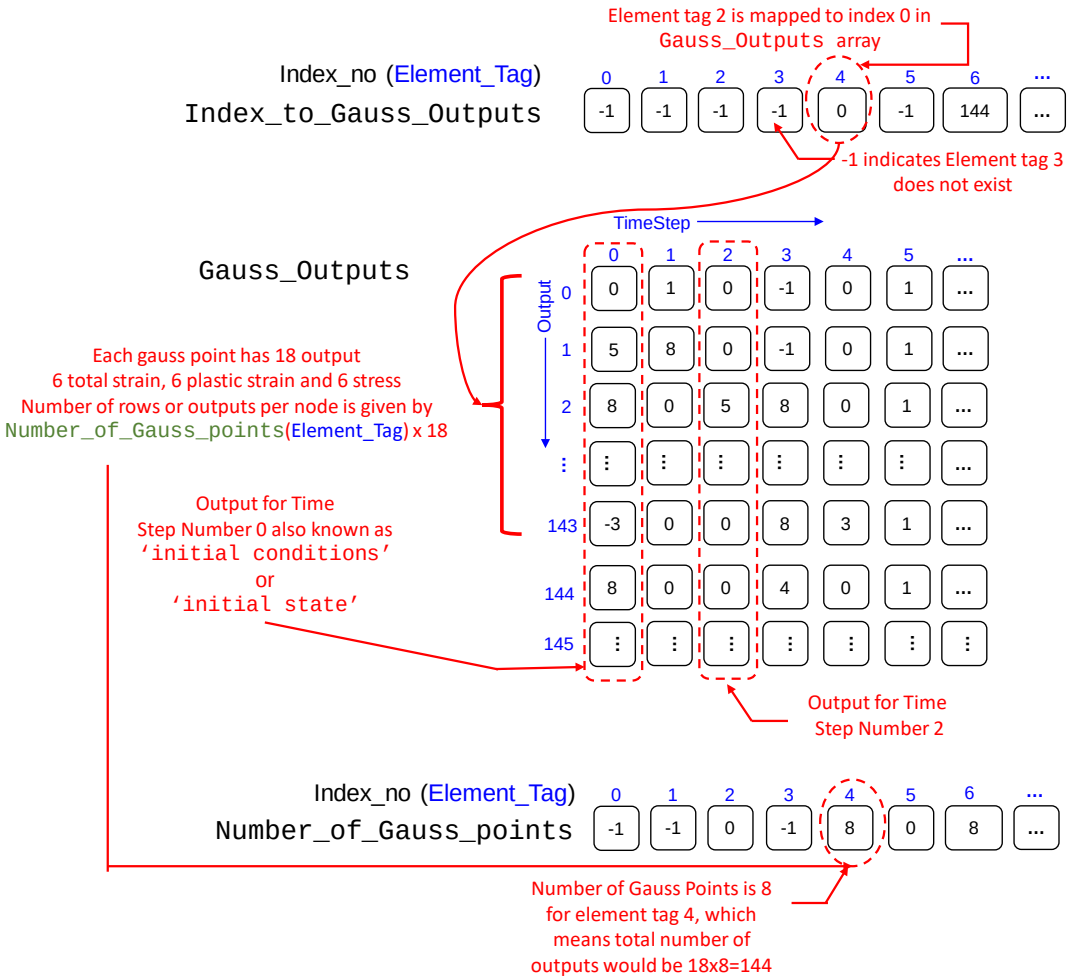


Figure 206.24: Arrays describing gauss output information in `Elements` group directory of HDF5 file.

Position (start+offset)	Content meaning	Position (start+offset)	Content meaning	Position (start+offset)	Content meaning
start+0	$\epsilon_{xx}[]$	start+6	$\epsilon_{xx}^{\text{plastic}}[]$	start+12	$\sigma_{xx}[Pa]$
start+1	$\epsilon_{yy}[]$	start+7	$\epsilon_{yy}^{\text{plastic}}[]$	start+13	$\sigma_{yy}[Pa]$
start+2	$\epsilon_{zz}[]$	start+8	$\epsilon_{zz}^{\text{plastic}}[]$	start+14	$\sigma_{zz}[Pa]$
start+3	$\epsilon_{xy}[]$	start+9	$\epsilon_{xy}^{\text{plastic}}[]$	start+15	$\sigma_{xy}[Pa]$
start+4	$\epsilon_{xz}[]$	start+10	$\epsilon_{xz}^{\text{plastic}}[]$	start+16	$\sigma_{xz}[Pa]$
start+5	$\epsilon_{yz}[]$	start+11	$\epsilon_{yz}^{\text{plastic}}[]$	start+17	$\sigma_{yz}[Pa]$

Table 206.7: Output Format for each gauss point defined inside element

Gauss-point 2 will then start at position 18 through 36 with the same meaning for each row. And so-on for the other Gauss-points.

206.8 Element-specific output format

`Element_Outputs` array stores output for the elements except those stored at gauss points (i.e. stress, total strain and plastic strain). It is a 2D `float` array indexed by the `Index_to_Element_Outputs` array and the number of rows is defined `Number_of_Element_Outputs`. he column index is represented by the time step of the simulation. Time index 0 represents initial state conditions, i.e. the state before the start of new stage and end of previous stage.

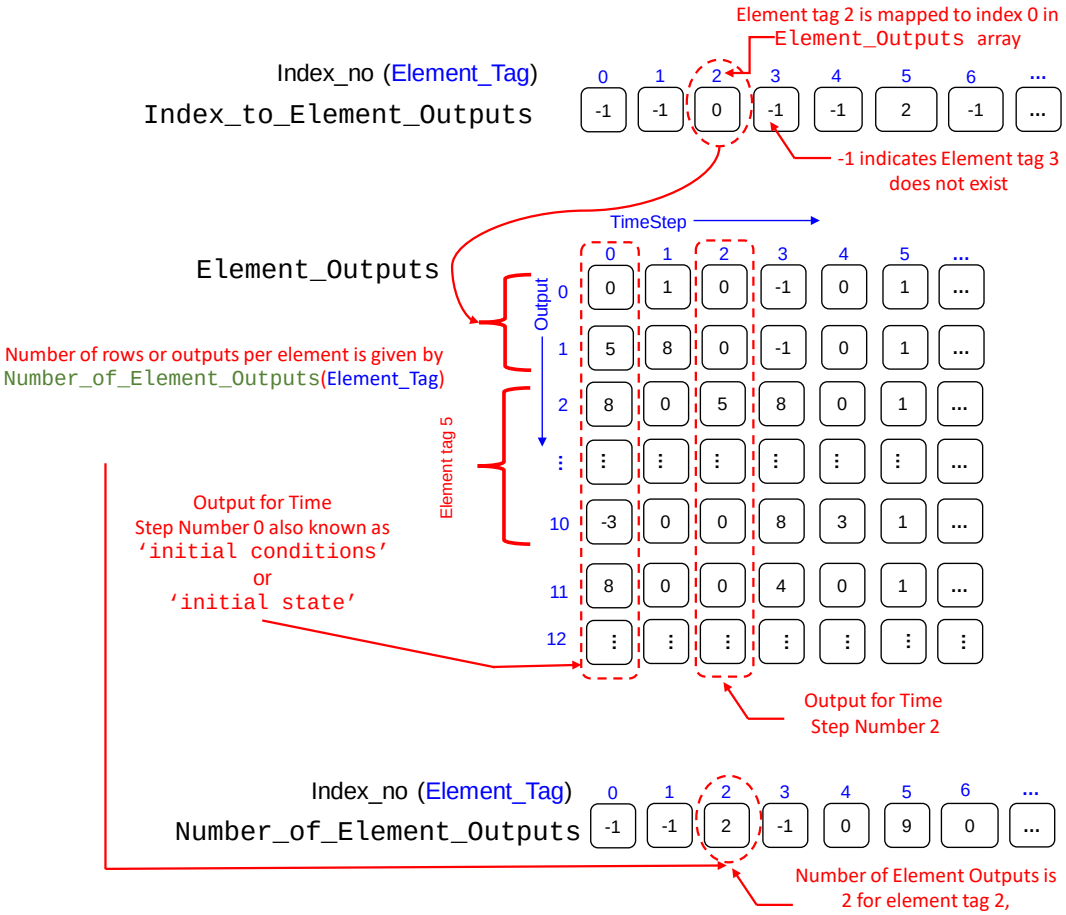


Figure 206.25: Arrays describing output information in `Elements` group directory of HDF5 file.

Figure 206.25 shows how to read `Element_Outputs` array for a particular element and particular time step. In the given example Listing 206.2, element tag 2 has two outputs. Similarly, element tag 5 has 9 outputs. On the other hand element tag 4 has no output. For element 2, the output starts at row index 0 and ends at 1. Similarly, for element tag 4, the output is stored in row index 2..10.

Since the `Element_Outputs` array format depends on the elements present in the model, one must

refer to each element specifically. Each element writes information into the `Element_Outputs` array in a different way. The user can determine the rows of the `Element_Outputs` array that belong to a given element by looking into the `Index_to_Element_Outputs` array which relates element tag to the starting position of that element's output within the `Element_Outputs` array. Additionally, the `Number_of_Element_Outputs` tells the user how many rows after the starting position correspond to the given element output.

The actual meaning of each row is element dependent and is detailed in the following pages. Please note that elements not in this list have no output defined at the moment.

In what follows `start` corresponds to the starting position for the elements output as determined with reference to the `Index_to_Element_Outputs` for the element of interest. To this number we add the corresponding `offset` as determined by each table below and interpret the row according to the meaning established below.

206.8.1 Truss

This element outputs 2 rows in total. First row is the uniaxial change in length while second component is the axial force. Truss does not have any gauss points and thus do not have any gauss outputs. The description of each of these rows are shown in Table 206.8:

Position (start+offset)	Content meaning
start+0	$\Delta L[m]$
start+1	$ForceF[N]$

Table 206.8: Element Output description for Truss

206.8.2 Brick Elements

All the bricks (u, up, upU and variable) have 0 element outputs. But, they do output total strain, plastic strain and stress for all their corresponding number of gauss points involved in elasto-plastic integration. The format in which gauss outputs are stored for each gauss point is described in Section 206.8

206.8.3 ShearBeam

Like brick elements, Shear Beam element also does not have any element output. But it does output for the *one* gauss point it has. Thus, in total it has 18 outputs for the only one gauss point. The format in

which gauss outputs are stored for each gauss point is described in Section 206.8

206.8.4 ElasticBeam

This element outputs 6 components of local nodal displacements at each of its two nodes, and 6 components of end forces at each of its two nodes. Total number of outputs is thus $2 \times 6 \times 2 = 24$ rows per element. The description of each of these rows are shown in Table 206.9:

Position (start+offset)	Content meaning	Position (start+offset)	Content meaning
start+0	$u_x^{\text{local},1}[m]$	start+6	$u_x^{\text{local},2}[m]$
start+1	$u_y^{\text{local},1}[m]$	start+7	$u_y^{\text{local},2}[m]$
start+2	$u_z^{\text{local},1}[m]$	start+8	$u_z^{\text{local},2}[m]$
start+3	$\theta_x^{\text{local},1}[radian]$	start+9	$\theta_x^{\text{local},2}[radian]$
start+4	$\theta_y^{\text{local},1}[radian]$	start+10	$\theta_y^{\text{local},2}[radian]$
start+5	$\theta_z^{\text{local},1}[radian]$	start+11	$\theta_z^{\text{local},2}[radian]$
Position (start+offset)	Content meaning	Position (start+offset)	Content meaning
start+12	$F_x^{\text{local},1}[N]$	start+18	$F_x^{\text{local},2}[N]$
start+13	$F_y^{\text{local},1}[N]$	start+19	$F_y^{\text{local},2}[N]$
start+14	$F_z^{\text{local},1}[N]$	start+20	$F_z^{\text{local},2}[N]$
start+15	$M_x^{\text{local},1}[N-m]$	start+21	$M_x^{\text{local},2}[N-m]$
start+16	$M_y^{\text{local},1}[N-m]$	start+22	$M_y^{\text{local},2}[N-m]$
start+17	$M_z^{\text{local},1}[N-m]$	start+23	$M_z^{\text{local},2}[N-m]$

Table 206.9: Element Output description for Elastic Beam

206.8.5 4NodeShell_ANDES

This element currently does not have any gauss or element outputs.

206.8.6 BeamColumnDispFiber3d

This element outputs 6 components of end forces at each of its two nodes. Each row is described in Table 206.10:

Position (start+offset)	Content meaning	Position (start+offset)	Content meaning
start+0	$F_x^{\text{local},1}[N]$	start+6	$F_x^{\text{local},2}[N]$
start+1	$F_y^{\text{local},1}[N]$	start+7	$F_y^{\text{local},2}[N]$
start+2	$F_z^{\text{local},1}[N]$	start+8	$F_z^{\text{local},2}[N]$
start+3	$M_x^{\text{local},1}[N-m]$	start+9	$M_x^{\text{local},2}[N-m]$
start+4	$M_y^{\text{local},1}[N-m]$	start+10	$M_y^{\text{local},2}[N-m]$
start+5	$M_z^{\text{local},1}[N-m]$	start+11	$M_z^{\text{local},2}[N-m]$

Table 206.10: Element Output description for displacement based fiber beams

206.8.7 Force Based Contact/Interface Elements

This element outputs 9 components: 3 components of gap displacement and 3 components of contact/interface forces and 3 components of incremental slip in the local axis definition. The first two components of gap are transverse components (g_{t1}, g_{t2}), while the third is the normal gap component g_n. Similarly, the first two components of force (F_{t1}, F_{t2}) are transverse (shear on contact/interface plane) while the third is the normal contact/interface force F_n. The last three components are incremental slip $\Delta g^{inc_slip}{}_1, \Delta g^{inc_slip}{}_2$ in the local transverse direction and total uplift Δn in local normal vector direction. If $\Delta n > 0$, there is uplift i.e. loss of contact/interface else it is in contact. Each row is described in Table 206.12:

Position (start+offset)	Content meaning	Position (start+offset)	Content meaning	Position (start+offset)	Content meaning
start+0	$g_{t1}[m]$	start+3	$F_{t1}[N]$	start+6	$\Delta g^{inc_slip}{}_1[m]$
start+1	$g_{t2}[m]$	start+4	$F_{t2}[N]$	start+7	$\Delta g^{inc_slip}{}_2[m]$
start+2	$g_n[m]$	start+5	$F_n[N]$	start+8	$Uplift \Delta g_n[m]$

Table 206.11: Element Output description for force based contact/interface elements

206.8.8 Stress Based Contact/Interface Elements

This element outputs 9 components: 3 components of gap displacement and 3 components of contact/interface forces and 3 components of incremental slip in the local axis definition. The first two components of gap are transverse components (g_{t1}, g_{t2}), while the third is the normal gap component g_n. Similarly, the first two components of stress $(\sigma_{t1}, \sigma_{t2})$ are transverse (shear on contact/interface plane)

while the third is the normal contact/interface stress σ_n. The last three components are incremental slip $\Delta g^{inc_s lip}{}_1, \Delta g^{inc_s lip}{}_2$ in the local transverse direction and total uplift Δn in local normal vector direction. If $\Delta n > 0$, there is uplift i.e. loss of contact/interface else it is in contact. Each row is described in Table 206.12:

Position (start+offset)	Content meaning	Position (start+offset)	Content meaning	Position (start+offset)	Content meaning
start+0	$g_{t1}[m]$	start+3	$\sigma_{t1}[N/m^2]$	start+6	$\Delta g^{inc_s lip}{}_1[m]$
start+1	$g_{t2}[m]$	start+4	$\sigma_{t2}[N/m^2]$	start+7	$\Delta g^{inc_s lip}{}_2[m]$
start+2	$g_n[m]$	start+5	$\sigma_n[N/m^2]$	start+8	$Uplift \Delta g_n[m]$

Table 206.12: Element Output description for stress based contact/interface elements

206.9 Energy Output Format

Energy folder in the output file stores simulation results for energy-related quantities. Energy density quantities are calculated at element integration points. Averaged energy density quantities are calculated for each element. Energy quantities are calculated for each element. Nodal input energy values are calculated at each node. All energy terms are calculated and stored at each time step.

Under the Energy folder, there are currently seven datasets. Four of them contain energy calculation results. The other three are index datasets, which can be used to find energy output for specific elements or nodes.

In general, energy output datasets are 2D float arrays. The column index is represented by the time step of the simulation. The row index represents various energy quantities that are calculated and stored. The three index datasets are used to locate the row index for specific energy quantities.

206.9.1 Input Energy

All types of nodal load, including DRM, are applied at nodes. Elemental loads are automatically converted to nodal loads, then also applied at nodes. Therefore, input energy or input work from all types of external load can be calculated at each node.

Under the Energy folder, the Nodal_Input_Energy dataset stores input energy at each node. Note that this is input energy accumulated from the start of simulation to the current time step. To find the input energy time history at a specific node, find the index in the Index_to_Nodal_Input_Energy dataset then find the corresponding entry in the Nodal_Input_Energy dataset.

For example, here are the steps to get the input energy time history at the node with node tag 73, Go to Index_to_Nodal_Input_Energy, find the index for node 73 to be 105. Then the accumulated input energy at your chosen node is stored at row 105 in Nodal_Input_Energy.

206.9.2 Energy Density Quantity at Gauss Point

Energy_Density_GP dataset stores energy density quantities at each Gauss point. To find the energy density time history at a specific Gauss point, find the index in the Index_to_Energy_Density_GP dataset then find the corresponding entry in the Energy_Density_GP dataset.

For each Gauss point, 12 data slots are occupied in the following order:

- Incremental kinetic energy density
- Accumulated kinetic energy density
- Incremental strain energy density

- Accumulated strain energy density
- Incremental plastic free energy density
- Accumulated plastic free energy density
- Incremental plastic dissipation density
- Accumulated plastic dissipation density
- The last 4 slots are currently empty, reserved for potential future use

Note that "incremental" means the change of energy density during the time step, "accumulated" means current cumulative energy density at the time step.

Here is an example showing how to obtain the energy density evolution at a specific Gauss point. Say you are interested in looking at the accumulated plastic dissipation density at the third Gauss point of an `8NodeBrick` element with element tag 73. Go to `Index_to_Energy_Density_GP`, find the index for element 73 to be 3904. Since each Gauss point occupies 12 slots in `Energy_Density_GP`, the energy outputs for the third Gauss point starts at index 3904+(3-1)*12=3928. Finally, since accumulated plastic dissipation density is the 8th entry among the 12 slots, the row index for the data of your interest is 3928+(8-1) = 3935. This means that the accumulated plastic dissipation density at your chosen location is stored at row 3935 of `Energy_Density_GP`.

206.9.3 Average Energy Density Quantity for Element

`Energy_Density_Element_Average` dataset stores averaged energy density quantities of each element. To find the energy density time history of a specific element, find the index in the `Index_to_Energy_Element` dataset then find the corresponding entry in the `Energy_Density_Element_Average` dataset.

For each element, 12 data slots are occupied in the following order:

- Incremental kinetic energy density
- Accumulated kinetic energy density
- Incremental strain energy density
- Accumulated strain energy density
- Incremental plastic free energy density
- Accumulated plastic free energy density

- Incremental plastic dissipation density
- Accumulated plastic dissipation density
- The last 4 slots are currently empty, reserved for potential future use

Note that "incremental" means the change of energy density during the time step, "accumulated" means current cumulative energy density at the time step.

Here is an example showing how to obtain the energy density evolution of a specific element. Say you are interested in looking at the accumulated plastic dissipation density of an `8NodeBrick` element with element tag 73. Go to `Index_to_Energy_Element`, find the index for element 73 to be 612. Since accumulated plastic dissipation density is the 8th entry among the 12 slots, the row index for the data of your interest is 612+(8-1) = 619. This means that the accumulated plastic dissipation density of your chosen element is stored at row 619 of `Energy_Density_Element_Average`.

206.9.4 Energy Quantity for Element

`Energy_Element` dataset stores averaged energy density quantities of each element. To find the energy time history of a specific element, find the index in the `Index_to_Energy_Element` dataset then find the corresponding entry in the `Energy_Element` dataset.

For each element, 12 data slots are occupied in the following order:

- Incremental kinetic energy
- Accumulated kinetic energy
- Incremental strain energy
- Accumulated strain energy
- Incremental plastic free energy
- Accumulated plastic free energy
- Incremental plastic dissipation
- Accumulated plastic dissipation
- Incremental viscous energy dissipation
- Accumulated viscous energy dissipation
- The last 2 slots are currently empty, reserved for potential future use

Note that "incremental" means the change of energy density during the time step, "accumulated" means current cumulative energy density at the time step.

Here is an example showing how to obtain the energy evolution of a specific element. Say you are interested in looking at the accumulated plastic dissipation of an `8NodeBrick` element with element tag 73. Go to `Index_to_Energy_Element`, find the index for element 73 to be 612. Since accumulated plastic dissipation is the 8th entry among the 12 slots, the row index for the data of your interest is 612+(8-1) = 619. This means that the accumulated plastic dissipation of your chosen element is stored at row 619 of `Energy_Element`.

Chapter 207

Real-ESSI Pre Processing and Model Development Methods

(2010-2015-2017-2020-2021-)

(In collaboration with Prof. Sumeet Kumar Sinha, Dr. Hexiang Wang and Dr. Yuan Feng)

207.1 Introduction

207.2 Model Development Using gmsh

207.2.1 Introduction to gmESSI

The gmESSI, pronounced as *[gm-ESSI]*, is a translator that converts mesh file from gmsh (a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities) to Real-ESSI DSL format. The primary aim of this program is to provide an efficient pre-processing tool to develop Finite Element (FE) models in gmsh and make them interface with various Real-ESSI functionalities. The gmESSI translator package contains the translator, sublime plugin and the manual.

The gmESSI package is available at http://sokocalo.engr.ucdavis.edu/~jeremic/lecture_notes_online_material/_Chapter_SoftwareHardware_Pre_Processing/Real-ESSI_gmESSI.tgz.

The text editor sublime plugin *[gmESSI-Tools]* can be downloaded here: http://sokocalo.engr.ucdavis.edu/~jeremic/lecture_notes_online_material/_Chapter_SoftwareHardware_Pre_Processing/fei-syntax-n-snippets.tar.gz.

207.2.1.1 Getting Started

The translator utilizes the physical and entity group concept of *Gmsh* (http://geuz.org/gmsh/doc/texinfo/gmsh.html) (Geuzaine and Remacle, 2009), which gets imprinted in the mesh ".msh" file. The translator then manipulates these groups to convert the whole mesh to ESSI commands. Thus, making physical groups is the essential, key for conversion. The Translator basically provides some strict syntax for naming these Physical Groups which provides gmESSI information about the elements or (nodes) on which the translation operates. The translator is made so general that any other FEM program can use it with little tweaks to have their own conversion tool. A quick look at some important features of the program are:

- It has a lot of predefined commands which do the conversion at the blink of an eye. These commands make it easier to define elements, boundary conditions, contacts/interfaces, fixities, loads ....
- It provides a python module "gmessi". The users can import this modulue and can extend the functional capability of gmESSI.
- The *[gmESSI-Tools]* sublime plugin makes it easy by providing syntax coloring and auto-text-completion for gmESSI commands. *[gmsh-Tools]* sublime plugin can also be installed for gmsh syntax coloring and auto-completion.

- The translator uses a *mapping.fei* file to check for its command syntax and conversion. A user can easily add a command in *mapping.fei* and it would get reflected automatically in the translation.
- It automatically optimizes the Real-ESSI tags (node, element, load) for space and time efficiency while running simulation.

Installation Process: The Translator have its dependencies on Octave (3.2 or higher), Boost(1.58 or higher), (Python 2.7 or higher). One should make sure to have them before compiling it. On Linux Ubuntu distros the dependencies can be installed as

```
sudo apt-get install liboctave-dev
# Boost version should be higher than 1.48
sudo apt-get install libboost-all-dev
sudo apt-get install python-dev
```

Installation of the gmESSI translator is easy, just follow steps below.

```
## go to folder where you want to store and build gmESSI application

## download the package from main Real-ESSI repository
## this line below should be all one line
## HOWEVER it had to be broken in two lines to be readable
## so please make a single command out of two lines below
##
#
# using curly brackets to help in checking scripts, that rely on these
# brackets being available around URL
#
wget {http://sokocalo.engr.ucdavis.edu/~jeremic/lecture_notes_online_material/\
_Chapter_SoftwareHardware_Pre_Processing/Real-ESSI_gmESSI.tgz}
##
# make directory, move files, expand archive
mkdir Real-ESSI-gmESSI
mv _all_files_gmESSI_.tgz Real-ESSI-gmESSI
cd Real-ESSI-gmESSI
tar -xvzf _all_files_gmESSI_.tgz

## build the package
make # builds the application in curr_dir/build

## install the package
# -- by default the pacakage is installed in /usr/local
make install # installs the package in /usr/local
# -- to change the install directory
make install INSTALL_DIR=install_dir_specified_by_user

############### For installation of gmESSI plugin in sublime ##############
# open sublime-text
```

```
# make sure you have installed package control
# if not then install it first from
# {https://packagecontrol.io/installation}
# go to Preferences->PackageControl->InstallPackage
# search for [gmESSI-Tools] and install it
# also install [gmsh-Tools] following the same steps
# restart sublime
#########################################################################
```

207.2.1.2 Running gmESSI

gmESSI can be invoked from the bash terminal by typing *gmessy*. It can take one or multiple *xyz.gmessi* files as an argument and convert them to Real-ESSI files in their respective simulation directory defined by the user. By default, the 'gmessi' python module is automatically imported and available as 'gmESSI' in '.gmeesi' input file.

gmessy is a top level python script that parses the .gmessi file and categories commands in the following order as

- gmESSI Command: gmESSI Commands are one line commands. They start and end with '[' and ']' respectively. Section 207.2.3 describes the syntax.
- gmESSI Comments: The lines that start with '//' are considered as gmESSI comments. It gets translated and copied to the main file (See Section 207.2.4.6).
- Singular Commands: The lines that start with '!' are directly copied to the main file (See Section 207.2.4.6 and Section 207.2.5.1). Real-ESSI domain specific language (DSL) are written following the exclamation mark '!' sign.
- Python Comments: The lines that start with '#' are considered as python comments.
- Python Commands: Whatever lines left are considered as python commands. This option is only for the advanced user and is not documented to make the manual simple. Only some useful commands required are explained in the manual.

The categorized commands then generates an equivalent python (.py) script, which gets finally run in python interpretor. The generated equivalent python script can be seen by adding '-l' or '–logfile= LOG_FILE' option during execution. It is important to note that nodes, coordinates, element no etc generated from the translator have a precision associated with them. By default the precision is up-to '6' significant digits. The user can change the precision anywhere in the .gmessi file as

```
gmESSI.setPrecision(10);
```

This will set the precision to '10' significant digits. Lowering precision can be helpful in generating same coordinates for contact/interface node pairs. See [Example_4.gmessi] for its usage.

The full description of *gmessy* can be invoked from the terminal as

```
$gmessy --help

usage: gmessy [-h] [-l] [-nm] [-em] [-ne] [--logfile= LOG_FILE]
[--nodemap= NM_FILE] [--elemap= ELM_FILE]
gmessi_filename

positional arguments:
gmessi_filename filename containing semantics of conversion

optional arguments:
-h, --help show this help message and exit
-l generate the log file at the current location
-nm generate the node map file at the current location
-em generate the element map file at the current location
-ne don't carry out the conversion
--logfile= LOG_FILE generate the log file at specified location
--nodemap= NM_FILE generate the node-map (gmsh-to-Real_ESSI) file at specified ↩
    location
--elemap= ELM_FILE generate the element-map (gmsh-to-Real_ESSI) file at ↩
    specified location
```

Since gmESSI optimizes the 'node' and 'element' tag for Real-ESSI, it provides an interface to retrieve the node map and element map containing mapping from gmsh_tag to Real_ESSI_Tag.

Running gmESSI requires, the *.gmessi* input file and the gmsh mesh (.msh) mesh file containing physical groups. Let's go and run an example to see how gmESSI works. [Example_1] can be obtained here.

Alternatively in the gmESSI directory, navigate to the Examples directory and then to Example_1 directory.

```
$cd ./Examples/Example_1
$ls
Example_1.geo # geometry file [gmsh]
Example_1.msh # mesh file [gmsh]
Example_1.gmessi # gmessi input file [gmESSI]
```

Contents of Example_1.gmessi input file: As described above, the .gmessi input file contains gmESSI commands, singular commands and python commands. Also, it can contain comments followed by // or #.

At the beginning of the input file, the simulation directory, main, node, element, load filenames must be specified. Also, before adding any gmESSI command, mesh must be loaded using 'gmESSI.loadGmshFile' command.

```
$ cat Example_1.gmessi

### loading the msh file
gmESSI.loadGmshFile("Example_1.msh")

### Physical Groups defined in the msh file.
#2 2 "Base_Surface"
#2 3 "Top_Surface"
#3 1 "Soil"

### Defining the Simulation Directory
gmESSI.setSimulationDir("./Example_1_ESSI_Simulation")
gmESSI.setMainFile(gmESSI.SimulationDir+ "main.fei")
gmESSI.setNodeFile(gmESSI.SimulationDir+ "node.fei")
gmESSI.setElementFile(gmESSI.SimulationDir+ "element.fei")
gmESSI.setLoadFile(gmESSI.SimulationDir+ "load.fei")


// My new model
! model name "Soil_Block";

[Add_All_Node{ unit:= m, nof_dofs:= 3}]

// Adding Material layer wise and also assigning it to elements
[Vary_Linear_Elastic_Isotropic_3D{Physical_Group#Soil, ElementCommand:= ↩
    [Add_8NodeBrick{}], Density:= 1600+10*(10-z)\ 0 \kg/m^3, ElasticModulus:= ↩
    20e9+10e8*(10-z)\-8\Pa, PoissonRatio:= 0.3}]

! include "node.fei";
! include "element.fei";
! new loading stage "Stage1_Surface_Loading";

# Applying Fixities
[Fix_Dofs{Physical_Group#Base_Surface, all}]

#### For applying Surface load on the Top Surface of the Soil Block
#[Add_8NodeBrick_SurfaceLoad{Physical_Group#1,Physical_Group#3,10*Pa}]

#### For applying Nodal loads to all the nodes of the top surface
[Add_Node_Load_Linear{Physical_Group#Top_Surface, ForceType:= Fx, Mag:= 10*kN}]

#### For applying Self-Weight Load to the soil elements
! add acceleration field # 1 ax = 0*g ay = 0*g az = -1*g ;
! add load #18 to all elements type self_weight use acceleration field # 1;

# Updating the tag inside gmESSI as user entered by himself load tag
gmESSI.setESSITag("load",19)
```

```

! include "load.fei";
! NumStep = 10;
!
! define algorithm With_no_convergence_check;
! define solver UMFPack;
! define load factor increment 1/NumStep;
! simulate NumStep steps using static algorithm;
! bye;
```

Running Example_1 in Terminal:

```
$ gmessy Example_1.gmessi
Message:: newDirectory created as ./Example_1_ESSI_Simulation


Add_All_Node{ unit:= m, nof_dofs:= 3}
   Found!!
   Successfully Converted

Vary_Linear_Elastic_Isotropic_3D{Physical_Group#Soil, ElementCommand:= ↩
    [Add_8NodeBrick{}], Density:= 1600+10*(10-z)\ 0 \kg/m^3, ElasticModulus:= ↩
    20e9+10e8*(10-z)\-8\Pa, PoissonRatio:= 0.3}
   Found!!
   Successfully Converted

Fix_Dofs{Physical_Group#Base_Surface, all}
   Found!!
   Successfully Converted

Add_Node_Load_Linear{Physical_Group#Top_Surface, ForceType:= Fx, Mag:= 10*kN}
   Found!!
   Successfully Converted

********************** Updated New Tag Numbering **********************
damping = 1
displacement = 1
element = 28
field = 1
load = 17
material = 4
motion = 19
node = 65
nodes = 65
Gmsh_Elements = 46
Gmsh_Nodes = 65
```

It must be noted that the terminal only displays information about gmESSI commands. The singular commands are directly copied to the main file. The translator creates a user defined directory *Example_1_ESSI_Simulation* and places

1. node.fei
2. element.fei
3. load.fei
4. main.fei
5. Example_1.msh

The terminal displays the WARNING, ERROR messages and log of command conversions as shown above. At the end, it displays the *Available ESSITag's* numbering, which can be refereed and used for further conversion.

ESSITags are explained later in this manual in Section 207.2.4.8.

The Real-ESSI input files produced can be tweaked a little if required. Once all is set, the model can be run through Real-ESSI Simulator

```
cd Example_1_ESSI_Simulation

### To run ESSI in sequential
# -- assuming sequential executable name is 'essi'
essi -f main.fei

### To run ESSI in parallel
# -- assuming parallel executable name is 'pessi'
mpirun -np 4 pessi -f main.fei
```

207.2.2 Gmsh Physical Groups and Geometrical Entities

207.2.2.1 Geometrical Entities

Geometrical entities are the most elementary group in Gmsh. Each point, line, surface and volume is a geometrical entity and possess a unique identification number. Elementary geometrical entities can then be manipulated in various ways, for example using the *Translate, Rotate, Scale or Symmetry* commands. They can be deleted with the *Delete* command, provided that no higher-dimension entity references them. Example_2.geo shows description of a geometry (.geo) file in gmsh for creating a *cantilever beam*. The files can be downloaded here. Alternatively, it can be located in the gmESSI directory by navigating to the Examples/Example_2 directory.

```
$ cat Example_2.geo
// Creating a point
Point(1) = {0,0,0};

// Dividing the beam length in 5 parts
Extrude (4,0,0) {Point{1}; Layers{5};}

// Dividing the beam width in 2 parts
Extrude (0,1,0) {Line{1}; Layers{2};Recombine;}

// Dividing the beam depth in 2 parts
Extrude (0,0,1) {Surface{5}; Layers{2};Recombine;}
```

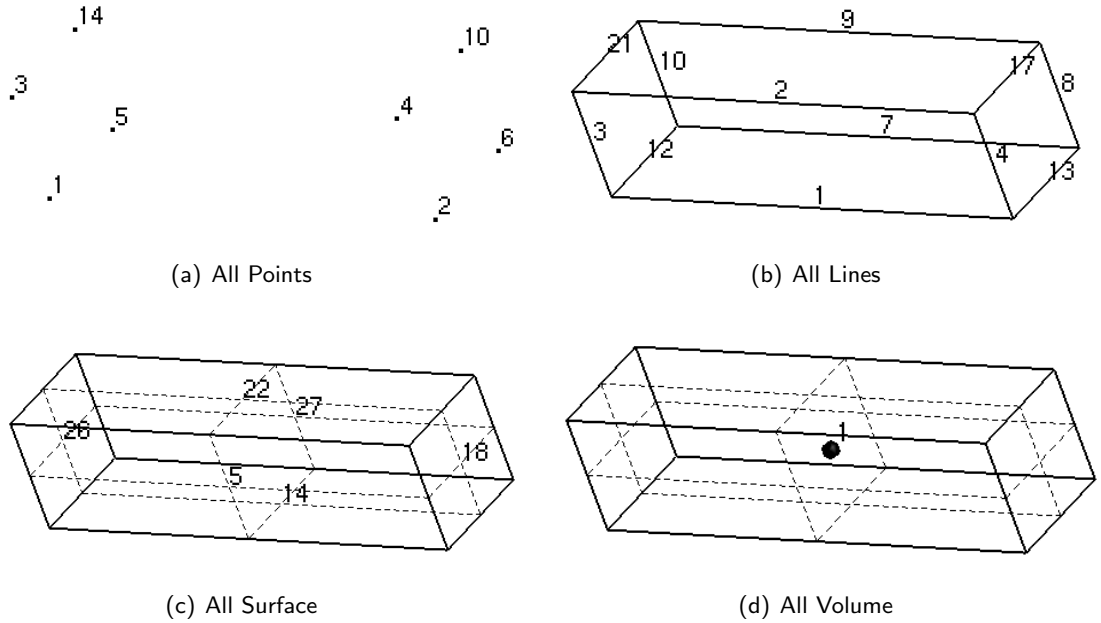
Figure 207.1 shows, the different unique identification number attached to each of the nodes, lines, surface and volume of the geometry of *cantilever beam*. Physical groups can now be created of type {nodes, lines, surface or volume} containing one or more geometrical entities of their respective type.

207.2.2.2 Physical Groups

Physical groups are groups of same type {nodes, lines, surface, volume} of elementary geometrical entities. These Physical Groups cannot be modified by geometry commands. Their only purpose is to assemble elementary entities into larger groups, possibly modifying their orientation, so that they can be referred to by the mesh module as single entities. As is the case with elementary entities, each physical point, physical line, physical surface or physical volume are also assigned a unique identification number.

NOTE:- A geometrical entity has only one elementary entity number but can be a part of many physical groups by sharing their unique identification number.

Below is the continuation of Example_2.geo in Gmsh for creating physical Groups of *cantilever beam*.



(a) All Points (b) All Lines

(c) All Surface (d) All Volume

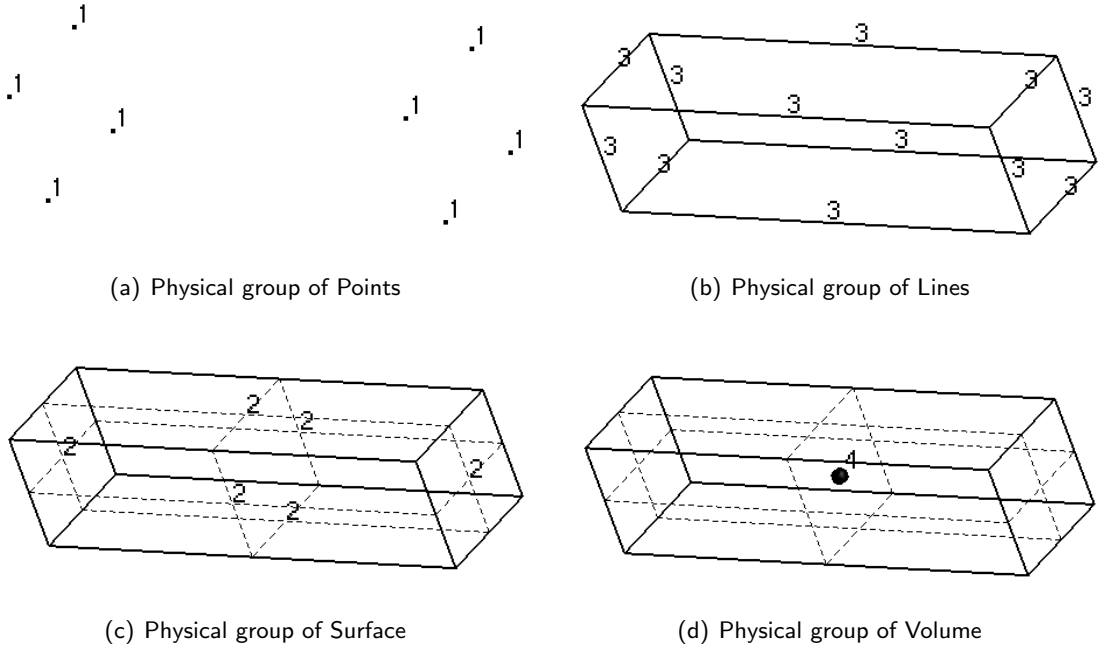
Figure 207.1: Showing Geometrical Entities. Every point, line, surface and volume has an unique identification number assigned to it.

Just for the sake of example, 4 physical groups are created which consist of all points, lines, surface and volume respectively of the *cantilever beam model*. Also physical groups of the surface where fixities and load is applied is created.

```
$ cat Example_2.geo
.....
Physical Point ("All_Points") ={1,2,3,4,5,6,10,14};
Physical Surface("All_Surfaces") = {5,14,22,27,18,26};
Physical Line("All_Lines") ={1,2,3,4,12,13,21,17,7,8,9,10};
Physical Volume("All_Volumes") ={1};
Physical Surface("ApplySurfaceLoad") ={27};
Physical Surface("SurfaceToBeFixed") ={26};
```

In generated mesh (.msh) file, all the geometrical entities have a tag list which contains the ids of the physical groups to which it belongs or is associated. In the above example shown in Figure 207.2, every point, line, surface, volume belongs to only one physical group and thus are showing only one associative number against themselves. Figure 207.3 shows geometrical entities which are part of many physical groups. For example:- the volume shown in Figure 207.3 shows physical group of volumes having id 1 and 7.

The whole idea of creating a Physical Group of points, lines, surfaces and volumes and giving it a unique string name is to allow quick identification and manipulation during gmESSI commands. In Gmsh the



(a) Physical group of Points

(b) Physical group of Lines

(c) Physical group of Surface

(d) Physical group of Volume

Figure 207.2: Showing all 4 Physical Groups with entities numbered by their physical group id's.

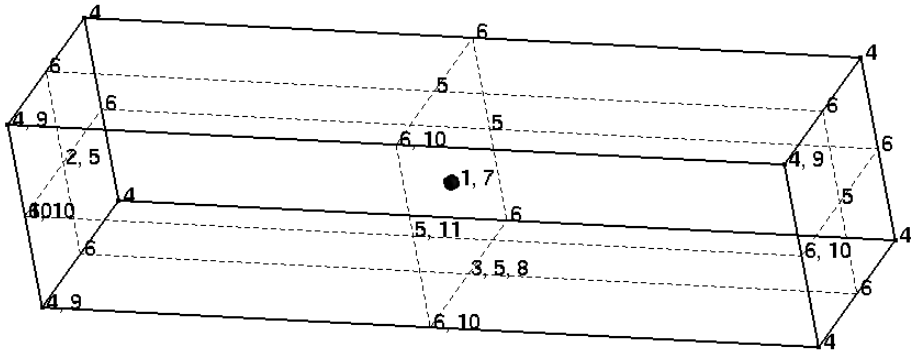


Figure 207.3: Showing geometrical entities associated with more than one physical group.

name of these Physical Group along with their corresponding elements and nodes gets transferred to the mesh .msh file as shown below. Figure 207.4 shows how Gmsh interprets these Physical groups in .msh file.

```
$cat Example_1.msh
.....
$PhysicalNames
6
0 1 "All_Points"
1 3 "All_Lines"
2 2 "All_Surfaces"
```

```
2 5 "ApplySurfaceLoad"
2 6 "SurfaceToBeFixed"
3 4 "All_Volumes"
$EndPhysicalNames
......
```

NOTE:- While creating a physical group in Gmsh, only the information (nodes and elements) of that physical group gets written in the .msh file and rest are not written. So one must be careful to create physical groups of all entities which is needed during post-processing or conversion. More information about Gmsh syntax, physical groups, commands, .msh file, save options, is available at the main online documentation web site: http://geuz.org/gmsh/doc/texinfo/gmsh.html

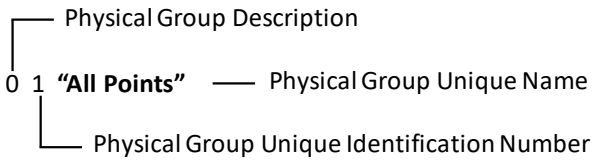


Figure 207.4: Description showing how gmsh interprets the Physical Groups.

- Physical Group Description :: Gmsh uses it to identify the type of physical group. $0, 1, 2$ *and* 3 represents the physical group of geometric points, lines, surface and volume respectively.
- Physical Group Unique Identification Number :: It is an unique identification number automatically assigned to each physical group by gmsh.
- Physical Group Unique Name :: It is also the same as *Physical Group Unique Identification Number* but the difference is that it is not automatic but defined by the user and that too in the form of string.

The gmESSI Translator utilizes the property of naming the physical group as "string" to get gmESSI commands from the user along with specific physical group on which it is operated. Below in shown [Example_2.gmessi] input file for a *Cantilever analysis*. It shows how to write gmESSI commands with physical group information on which it is operated. gmESSI utilizes the mesh (.msh) file to get the respective physical group and translated it to ESSI input (.fei) files.

```
$ cat Example_2.gmessi

gmESSI.loadGmshFile("Example_2.msh")
```

```
4
5  ####### Physical Groups Available in Example_2.msh file
6  #0 1 "All_Points"
7  #1 3 "All_Lines"
8  #2 2 "All_Surfaces"
9  #2 5 "ApplySurfaceLoad"
10 #2 6 "SurfaceToBeFixed"
11 #3 4 "All_Volumes"
12
13 ############ Important!! to set the file names ###################
14 gmESSI.setSimulationDir("./Example_2_ESSI_Simulation")
15 gmESSI.setMainFile(gmESSI.SimulationDir+ "main.fei")
16 gmESSI.setNodeFile(gmESSI.SimulationDir+ "node.fei")
17 gmESSI.setElementFile(gmESSI.SimulationDir+ "element.fei")
18 gmESSI.setLoadFile(gmESSI.SimulationDir+ "load.fei")
19
20 // My new model
21 ! model name "Cantilever_Analysis";
22
23 [Add_All_Node{Unit:= m, NumDofs:= 3}]
24
25 // Adding Material
26 ! add material 1 type linear_elastic_isotropic_3d mass_density = 2000*kg/m^3 ↩
       elastic_modulus = 200*MPa poisson_ratio = 0.2;
27
28 [Add_8NodeBrick{Physical_Group#All_Volumes, material_no:= 1}]
29 [Fix_Dofs{Physical_Group#SurfaceToBeFixed, all}]
30
31 ! include "node.fei";
32 ! include "element.fei";
33
34 ! new loading stage "Stage1_Uniform_Surface_Load";
35
36 # Adding Surface Load
37 #[Add_8NodeBrick_SurfaceLoad{Physical_Group#All_Volumes, ↩
       Physical_Group#ApplySurfaceLoad, -10*Pa}]
38 [Add_Node_Load_Linear{Physical_Group#ApplySurfaceLoad, ForceType:= Fz, Mag:= ↩
       -10*kN}]
39
40 ! include "load.fei";
41 ! define algorithm With_no_convergence_check;
42 ! define solver UMFPack;
43 ! define load factor increment 1;
44 ! simulate 10 steps using static algorithm;
45 ! bye;
```

NOTE:- The first command in [.gmessi] file should be to load the mesh (.msh) file. The syntax to load the gmsh generated mesh file is

```
1 gmESSI.LoadGmshFile("meshfile.msh")
```

The gmESSI translator reads the command *[Add_All_Node{ Unit:= m, NumDofs:= 3}]* and adds all the nodes from mesh file to ESSI input files. Similarly it translates all the other commands as well.

207.2.3 gmESSI Command Description

gmESSI Translator as said above utilizes the naming of the physical groups to get commands from the user and then carry out the conversion by acting on the defined physical group.

207.2.3.1 gmESSI Syntax

gmESSI follows strict syntax. gmESSI parses the physical group name string in mesh (.msh) file. Let us have a quick look at the syntax of physical group name.

Physical Group Names : Physical group names are created inside gmsh geometry file. gmESSI follows special syntax as described below.

1. Physical group names used in gmsh should be unique for gmESSI to identify them during post processing.
2. Physical group names should not contain any space
3. Physical group tags can be any alphanumeric sequence but should not contain any of these [ ]$ literals in their names. Example "Physical_Group_1"

gmESSI Command Syntax : gmESSI translator commands are always enclosed between opening/closing square brackets [and] respectively. A typical gmESSI command syntax is shown in Figure 207.5

Figure 207.5: gmESSI command description.

- Command Name : Just as regular function gmESSI Commands have a name and take arguments. The names are usually self explanatory of its function like *Add_8NodeBrick*{...}, *Free_Dofs*{...} .. *etc*
- Physical Group Argument : Usually the gmESSI commands have first argument as physical group . For Example:- *Add_8NodeBrick*{*PhysicalGroup#5,...*}, *Add_8NodeBrick*{*PhysicalGroup#All_Volumes,...*}, *Free_Dofs*{*PhysicalGroup#4,...*},.. *etc*.

  Physical Group Id can be the gmsh unique string or number representing that physical group (as shown in .msh file).

- Arguments : Arguments as always are separated by comma ','.
  - Argument Tag The arguments of gmESSI commands can also have tags associated with them so that it becomes easy for the user to interpret the argument and make changes in future. The tag and the argument is separated by :=. Tag itself has no meaning but it serves as an important information center for user. An example is shown below to show how tags are applied.

  - gmESSI command having arguments without tags
    1. [Add_Node_Load_Linear{Physical_Group#ApplySurfaceLoad, Fz, -10*kN}]
  - gmESSI command having arguments with tags
    1. [Add_Node_Load_Linear{Physical_Group#ApplySurfaceLoad, ForceType:= Fz, Mag:= -10*kN}]
    2. [Add_Node_Load_Linear{Physical_Group#ApplySurfaceLoad, ForceType:= Fz, -10*kN}]
    3. [Add_Node_Load_Linear{Physical_Group#ApplySurfaceLoad, Force_Direction:= Fz, Strength:= -10*kN}]

It can be seen from above examples that the tags are optional and also the user can put their own tag names. The sublime plugin [*gmESSI-Tools*] comes with elaborative tags for the parameters and a lot more with syntax coloring and text-completion for gmESSI commands. It is encouraged to use the plugin and take its advantage.

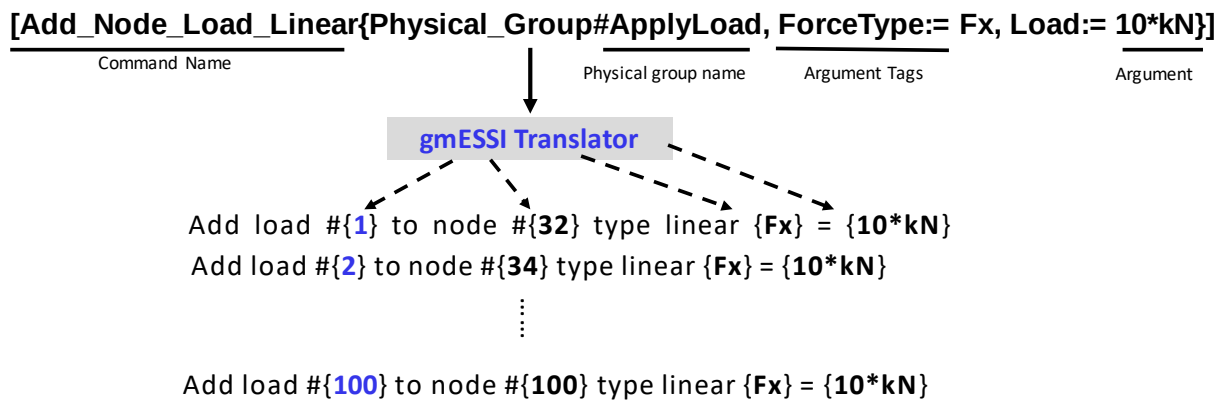


Figure 207.6: gmESSI conversion description.

Figure 207.6 shows the illustration how gmESSI works. Load gets added to all the nodes of the physical group 'ApplyLoad'. gmESSI translator automatically assigns the unique load tag sequentially. It

retrieves the node tag from the physical group. Rest of the information like 'ForceType' and 'Magnitude' is obtained from the arguments.

Most of the time these arguments are dummy which means that they just get copied to their equivalent ESSI command at their respective places. These arguments thus have a "string" data-type. For example: the command Add_Node_Load_Linear{Physical_Group#ApplySurfaceLoad, Fz, -10*kN} is equivalent to the Real-ESSI command add load #{} to node #{} type linear {} = {}. Fz and -10*kN goes to their respective position directly through the translator as shown in the Figure 207.6 load number 1 and node number 32 are computed by the translator and then inserted in the ESSI command.

NOTE:- The gmESSI Translator does not provide syntax checking for those dummy arguments. It means that, whatever is written gets copied at the respective position in the equivalent ESSI command, so the one must be careful with what they are writing in these arguments. For Example the command Add_Node_Load_Linear{Physical_Group#Id, ForceDirection, Magnitude} based on the arguments can get converted as

1. [Add_Node_Load_Linear{Physical_Group#ApplySurfaceLoad, ForceType:= Fz, Mag:= -10*kN}]
   —> add load #1 to node #32 type linear Fz = -10*kN
   —> add load #2 to node #33 type linear Fz = -10*kN
   .....
   —> add load #100 to node #100 type linear Fz = -10*kN
2. [Add_Node_Load_Linear{Physical_Group#ApplySurfaceLoad, ForceType:= Fz, Mag:= -10}]
   —> add load #1 to node #32 type linear Fz = -10
   —> add load #2 to node #33 type linear Fz = -10
   .....
   —> add load #100 to node #100 type linear Fz = -10
3. [Add_Node_Load_Linear{Physical_Group#ApplySurfaceLoad, ForceType:= Ft, Mag:= -10*kN}]
   —> add load #1 to node #32 type linear Ft = -10*kN
   —> add load #2 to node #33 type linear Ft = -10*kN
   .....
   —> add load #100 to node #100 type linear Ft = -10*kN

All the above conversions are correct. But only conversion (1.) is correct as an input for Real-ESSI Simulator because force direction is one of *Fx,Fy,Fz* and magnitude *10*kN* has *proper units*. So one

must be very careful while writing the arguments.

Note: Some of the arguments are not string but represents numerical quantities, which are manipulated by the translator during conversion. Thus, the one must supply only numbers without any alphabets else it would lead an unexpected termination of program. These arguments corresponds to *Special Commands* such as *Connect Command* and *Variational Commands*. The manual talks about them later in Section 207.2.5.7.

207.2.3.2 gmESSI Command's Physical Group

As iterated earlier, gmESSI commands operates on physical groups. The gmESSI command usually have their first argument as physical on which it operates. The gmESSI syntax allows the users to operates it's command on specific physical groups. The user specifies the group by including an argument Physical_Group#Tag in front of the gmESSI commands describing the command. The *tag* can be either *Physical_Group_Id, Physical_Group_Name*. Let's look at some of them

- [Add_Node_Load_Linear{Physical_Group#5,Fz,-10*kN}] operates on physical group 5
- [Add_8NodeBrick{Physical_Group#All_Volumes, 1}] operates on physical group which has string_tag as All_Volumes

For example in reference to *[Example_2.gmessi]* Physical_Group#All_Volumes or Physical_Group#4 refers the same physical group.

A physical group is a group of point, line, surface or volume defined by the user which contains all the geometrical entities that falls under that domain/group. Figure 207.2 shows physical groups.

207.2.4 gmESSI Output

gmESSI Translator translates the gmESSI commands operated on mesh (.msh) file to different ESSI input (node, element, load and main) (.fei) files and put them in user-defined directory. It also updates the mesh (.msh) file and puts it in the same directory. The *log of translation, errors and warnings* are displayed on the terminal. Below is the demonstration of log messages one by one using *[Example_2.gmessi]* with mesh-file name *Example_2.msh*. The folders and Reall ESSI input (.fei) files that are created by the translator for Example_2.gmessi input file are.

207.2.4.1 Directory Example_2_ESSI_Simulation

gmESSI Translator creates simulation directory as specified by the user. The user is expected to create the necessary node (Section 207.2.4.4), element (Section 207.2.4.3) , load (Section 207.2.4.5) and main (Section 207.2.4.6) file to that directory. The user is expected to provide the directory and filenames before executing any gmESSI command. In case the directory already exists a warning messages is shown on the terminal and a new directory following the original name with '_n' (n is number) is created. A new Real-ESSI simulation directory is assigned by the following command

```
gmESSI.setSimulationDir("./Example_2_ESSI_Simulation", overwrite\_mode)
```

where, 'overwrite_mode=0' means that in case of already existing folder, a new directory following the original name with '_n' (n is number) is created. 'overwrite_mode=1' would not check for any conflicts and use the same directory as specified by user. For example:- running [Example_.gmessi] file would produce the following message.

```
$ gmessy Example_2.gmessi
Files converted to Examples/Example_2_ESSI_Simulation
```

Again, running the same example would produce the following message as shown below. In [Example_2.gmessi] overwrite is turned off and thats why it creates new-non conflicting directory by appending $_1$ to end.

```
$ gmessy Example_2.gmessi
Message:: newDirectory created as ./Example_2_ESSI_Simulation_1
```

The execution of *gmessy XYZ.gmessi* produces warnings/errors in the following situations.

- ERROR:: Please Enter the gmessi File :: It occurs if the user does not give a filename. The possible situation for getting this error is

```
$ gmessy
```

- ERROR:: The program failed to open the file XYZ.msh It occurs if the given file or one of the files in the argument does not exist or fails to open because of some reason.
- WARNING::Directory Already Present.The contents of the Folder may get changed :: It occurs when users translates the mesh file file XYZ.msh in overwrite mode and the corresponding folder XYZ_ESSI_Simulation already exists at the execution location.
- Files converted to Examples/Example_2_ESSI_Simulation :: The message refers to the location of the folder where the translations have been saved.

207.2.4.2 Translation Log Terminal

gmESSI Translator displays the log of translation of gmESSI commands to corresponding *Real-ESSI commands* on the terminal. Proper *Errors Messages* and *Warnings* are echoed to the user. The execution of the commands are sequential which means the commands written first are executed first and similarly their success and failure is also echoed first. Let us look at this aspect with Example_2.gmessi.

```
$cat Example_2.gmessi
......
! add material 1 type linear_elastic_isotropic_3d mass_density = 2000*kg/m^3 ↩
   elastic_modulus = 200*MPa poisson_ratio = 0.2;

[Add_8NodeBrick{Physical_Group#All_Volumes, material_no:= 1}]

[Fix_Dofs{Physical_Group#SurfaceToBeFixed, all}]

! include "node.fei";
! include "element.fei";
......
```

Here, the sequence of execution of commands is '! add material # 1 type linear_elastic_isotropic_3d mass_density $= 2000 * kg/m^3$ elastic_modulus $= 200 * MPa$ poisson_ratio $= 0.2$; ', [Add_8NodeBrick{ Physical_Group#All_Volumes, material_no:= 1}], [Fix_Dofs{ Physical_Group#SurfaceToBeFixed, all}] and '! include "node.fei";'. Notice that the same order gets reflected in the translation log on the terminal as shown below. Also, it must be noted that the commands followed by '!' or '//' or '#' or python commands do not have any log messages corresponding to them.

It must be noted that the lines following '!' are directly copied to the main (Section 207.2.4.6). Usually Real-ESSI domain specific language that does not operate/require any physical group should be written following exclamation '!' sign.

```
$ gmessy ./Example_2.gmessi
......
```

```
3
4  Add_8NodeBrick{Physical_Group#All_Volumes, material_no:= 1}
5  Found!!
6  Successfully Converted
7
8  Fix_Dofs{Physical_Group#SurfaceToBeFixed, all}
9  Found!!
10 Successfully Converted
```

Apart from displaying the log details on the terminal, similar log is added for each translation of gmESSI commands in their respective files in which they are translated. In these files, each successful translation is enclosed between corresponding *RespectiveGmESSICommand Begins* and *RespectiveGmESSICommand Ends*. The same is shown below through the contents of node.fei. Notice that all the translations are enclosed between Begins and Ends Tag.

```
1  $ cat Examples/Example_2_ESSI_Simulation/node.fei
2
3  //*****************************************************************
4  // Add_All_Node{Unit:= m, NumDofs:= 3}Starts
5  //*****************************************************************
6
7  add node # 1 at (0.000000*m,0.000000*m,0.000000*m) with 3 dofs;
8  add node # 2 at (4.000000*m,0.000000*m,0.000000*m) with 3 dofs;
9  add node # 3 at (0.000000*m,1.000000*m,0.000000*m) with 3 dofs;
10 ..........................................................
11
12 //*****************************************************************
13 // Add_All_Node{Unit:= m, NumDofs:= 3}Ends
14 //*****************************************************************
```

```
1  $ cat Examples/Example_2_ESSI_Simulation/element.fei
2
3  //*********************************************************************
4  // Add_8NodeBrick{Physical_Group#All_Volumes, material_no:= 1}Starts
5  //*********************************************************************
6
7  add element #1 type 8NodeBrick with nodes (51,46,29,37,33,17,1,9) use material #1;
8  add element #2 type 8NodeBrick with nodes (47,28,5,19,51,46,29,37) use material ↩
       #1;
9  add element #3 type 8NodeBrick with nodes (42,32,46,51,13,3,17,33) use material ↩
       #1;
10 ..........................................................
11
12 //*********************************************************************
13 // Add_8NodeBrick{Physical_Group#All_Volumes, material_no:= 1}Ends
14 //*********************************************************************
```

NOTE:- The ordering/sequence of commands in ESSI analysis file is important and so the user must

make sure that the translations are made in the same order or if not the user should change it manually by (cut/copy/paste) in *node.fei, load.fei and main.fei* files before execution.

Having given a short description of the other translation log/error messages. Let us look more closely one by one and understand the messages,errors and warnings prompted on the terminal.

- Found!! : This message in front of the gmESSI command as shown above on translation log in the terminal means that, the corresponding command was found in the gmESSI Command Library.
- Successfully Converted : As the message itself describes, it occurs if the command has been successfully translated.
- Not Found!! : It occurs if the gmESSI Translator could not find the arbitrary command XYZ in the gmESSI Command library. Example:- *Loading{Fx,10*kN} NotFound!!*
- WARNING:: Execution of the command escaped. The Gmessi command XYZ could not be found : The gmESSI Translator does not terminate the translation if a command is not found, instead gives this warning message following the *Not Found!! Error*.
- Error:: The command XYZ has a syntax error in Physical_Group# tag : It occurs if there is a syntax error in Physical_Group# argument. The correct represent ion for Physical group Tags is Physical_Group#n, where n is the group id as 1,2,3.. etc. Examples of improper representation are *Phy#2, Physical#Node, ..*
- Warning:: The command XYZ failed to convert as there is no such Physical Group :: It occurs if one of the arguments in the command is Physical_Group# and the specified physical group by the user does not exists in the .msh file.
- Warning:: The command XYZ could not find any nodes/elements on which it operates : It occurs if for a specified command, the required element types for translation could not be found in the specified Physical group. For Examples:- [Add_8NodeBrick{Physical_Group#1,1}] would give this warning as the Physical_Group#1 being a Physical line group does not contain any 8-Noded Brick elements on which this command operates.
- ERROR:: Gmsh File has invalid symbols in Node Section. Unable to convert string to integer in Gmsh File : It occurs if there is perhaps a string inside the Nodes section of .msh file.
- ERROR:: The command XYZ has a syntax errors :: It occurs if the specified command by the user contain any syntax errors caught while parsing the command.

- ERROR:: Gmsh File has invalid symbols in Element Section. Unable to convert string to integer in Gmsh File : It occurs if there is perhaps a string inside the Element section of .msh file.

207.2.4.3 Element File (element.fei)

Element file *element.fei* is one of four parts of Real-ESSI input file that contains the translation of commands related to only initialization of elements of the FEM mesh. Generally, all the conversions from Elemental Command (Section 207.2.5.5) are written to element file.

A new analysis element file is assigned by the following python command

```
gmESSI.setElementFile(gmESSI.SimulationDir+ "element.fei")
```

where, 'gmESSI.SimulationDir' returns the Real-ESSI Simulation directory specified by the user (see section 207.2.4.1).

207.2.4.4 Node File (node.fei)

Node file *node.fei* is one of four parts of Real-ESSI input file that contains the translation of commands related to only initialization of nodes of the FEM mesh. All the conversions from Add Node Command (Section 207.2.5.2) are written to node file.

A new analysis node file is assigned by the following python command

```
gmESSI.setNodeFile(gmESSI.SimulationDir+ "node.fei")
```

where, 'gmESSI.SimulationDir' returns the Real-ESSI Simulation directory specified by the user (see section 207.2.4.1).

207.2.4.5 Load File (load.fei)

Load file *load.fei* contains the translation of commands related to the load and boundary conditions on the structure, for example declaration of fixities, boundary conditions, tied/connected nodes, nodal loads, surface loads etc....

A new load file is assigned by the following python command

```
gmESSI.setLoadFile(gmESSI.SimulationDir+ "load.fei")
```

where, 'gmESSI.SimulationDir' returns the Real-ESSI Simulation directory specified by the user (see section 207.2.4.1).

207.2.4.6 Analysis File (main.fei)

Analysis file *main.fei* is the main file which is run on Real-ESSI Simulator. The main file must include load, node and element file through *include 'filename.fei'* command.

A new analysis main file is assigned by the following python command

```
gmESSI.setMainFile(gmESSI.SimulationDir+ "main.fei")
```

where, 'gmESSI.SimulationDir' returns the Real-ESSI Simulation directory specified by the user (see section 207.2.4.1). A typical analysis file after conversion looks like the following.

```
$ cat Examples/Example_2_ESSI_Simulation/Example_2_analysis.fei

// My new model
 model name "Cantilever_Analysis";

// Adding Material
add material 1 type linear_elastic_isotropic_3d mass_density = 2000*kg/m^3 ↩
    elastic_modulus = 200*MPa poisson_ratio = 0.2;

include "node.fei";
include "element.fei";

new loading stage "Stage1_Uniform_Surface_Load";

include "load.fei";
 define algorithm With_no_convergence_check;
 define solver UMFPack;
 define load factor increment 1;
 simulate 10 steps using static algorithm;
bye;
```

The user can now add solver, time steps and even rearrange the file structure accordingly to Real-ESSI syntax.

NOTE: Real-ESSI Interpreter is sequential and follows certain ordering in commands like materials should be declared before assigning to elements, main-follower nodes can be assigned only when both nodes are declared .. etc.. One should be careful with the order in which conversions are made and if necessary should change it manually by (cut/copy/paste) later in the files geometry.fei, load.fei and analysis.fei or use the python module discussed later before running in ESSI.
Please refer to the Real-ESSI manual for more details on the ordering of the commands.

207.2.4.7 Mesh File (XYZ.msh)

Mesh file *XYZ.msh* is the input required by the translator. The translator updates the mesh file with users addition. For example:- if Connect-Command (Section 207.2.5.8) is used, the file contains additional physical group, nodes and 2-noded elements. The Connect Command is discussed in the more detail later in Section 207.2.5.8.

207.2.4.8 Updated ESSI Tags Terminal

Updated ESSI Tags refers to the new tag numbering reference associated with ESSI Tags. ESSI has tag numberings associated for damping, displacement, element, field, load, material, and node/nodes. For example in Real-ESSI Command *add node # 1 at (x,y,z) with 3 dofs*, node is a tag and requires a new number like 1 to be associated with that node. The translator displays the new numberings available for each ESSI Tag so that the user is made aware of new numberings for manually specifying an ESSI command after the translation.

gmESSI also provides a python command to set the ESSI Tag. The command is

```
gmESSI.setESSITag(ESSI_Tag_Name,Tag)
```

where,

- ESSI_Tag_Name : It refers to a string representing to the Real-ESSI tag such as 'node', 'element', 'field'...etc
- Tag : It refers to an integer representing the next available tag.

NOTE : If user is writing its own Real-ESSI domain specific language (DSL), it is expected that the user will update the corresponding Real-ESSI tag used in that DSL. Otherwise, gmESSI would not be able to know the updated available tags. See Example_1.gmessi for its usage.

```
$ gmessy Example_2.gmessi
.....................
******* Updated New Tag Numbering ********
Damping = 1
displacement = 1
element = 21
field = 1
load = 19
material = 2
motion = 1
node = 55
nodes = 55
Gmsh_Elements = 127
Gmsh_Nodes = 55
```

207.2.5 gmESSI Commands

Having the knowledge about the syntax, output files, errors and warnings, its time to move on to different types of commands that gmESSI offers. it provide commands operated on physical group to allow conversion for to equivalent Real-ESSI commands. There are also some special command that gmESSI supports. For simplicity, the commands are categorized on the basis of their operation on nodes/elements. As stated earlier, the commands are translated to one of the four files *node.fei, element.fei, load.fei and main.fei*. Let us look at them closely one by one along with all its supported commands.

207.2.5.1 Singular Commands

Singular Commands does not require any physical group to operate. All the text following exclamation mark '!' are copied directly to the *main.fei* (Section 207.2.4.6). For Example:- ' ! include 'load.fei'; ' is translated as 'include "load.fei" ' in main.fei analysis file. See [Example_1.gmessi] for its usage.

Note:- Real-ESSI DSL/commands must be followed by the exclamation mark '!'.

207.2.5.2 Add Node Commands

Add Node Commands have only two commands. *[Add_All_Node{unit,nof_dofs}]* adds all the nodes generated in mesh (.msh) file to 'node.fei' file. Whereas, *[Add_Node{Unit,NumDofs}]* add all the nodes of only specified physical group by the user. These commands operates on all the nodes of the physical group and generate an equivalent Real-ESSI DSL for each of them.

NOTE:- Every Add Node commands get translated into the *node.fei* (Section 207.2.4.4).

- gmESSI : [Add_Node{PhysicalGroup , Unit , NumDofs}]
  translates to series of
  Real-ESSI DSL : add node # < . > at (< L >,< L >,< L >) with < . > dofs;
  operated over all the nodes defined in the gmsh '.msh' file.

- gmESSI : [Add_All_Node{Unit , NumDofs}]
  translates to series of
  Real-ESSI DSL : add node # < . > at (< L >,< L >,< L >) with < . > dofs;
  operated over all the nodes of the defined physical group

207.2.5.3 Nodal Commands : Operates On All Nodes of the defined Physical Group

Nodal commands operates on all the nodes of the physical group defined by the user. For example:- [Fix_Dofs{Physical_Group#Lateral_Surface,ux}] would fix ux degree of freedom of all the nodes of physical group 'Lateral_Surface'. It will generate equivalent Real-ESSI DSL 'fix node # < . > dof < . >' and apply to all the nodes of that physical group. Figure 207.6 shows how gmESSI operated on physical groups.

As earlier stated, that the arguments of gmESSI commands are dummy and gets copied directly to the ESSI equivalent command, so one must be very much aware while writing the arguments to the commands. The arguments should be filled with values of the corresponding ESSI command along with required units if any. For more details about the values to the arguments, please refer to ESSI Manual.

NOTE:- Every Nodal command gets translated to the *load.fei* file (Section 207.2.4.5).

The different commands under this category and their corresponding Real-ESSI commands are listed below

1. gmESSI : [Add_Nodes_To_Physical_Group{PhysicalGroup , Physical_Node_Group_String}]
   translates to series of
   Real-ESSI DSL : add nodes (< . >) to [physical_node_group] "*string*";
   operated over all the nodes of the defined physical group

2. gmESSI : [Add_Self_Weight_To_Node{PhysicalGroup , field#1}]
   translates to series of
   Real-ESSI DSL : add load # < . > to node # < . > type [self_weight] use acceleration field # < . >;
   operated over all the nodes of the defined physical group

3. gmESSI : [Add_Node_Load_Linear{PhysicalGroup , Force_Type , Magnitude}]
   translates to series of
   Real-ESSI DSL : add load # < . > to node # < . > type [linear] [FORCETYPE] = < *forceormoment* >; //[FORCETYPE] = [Fx] [Fy] [Fz] [Mx] [My] [Mz] [F_fluid_x] [F_fluid_y] [F_fluid_z]
   operated over all the nodes of the defined physical group

4. gmESSI : [Add_Node_Load_Path_Time_Series{PhysicalGroup , Force_Type , Magnitude , Series_File}]
   translates to series of
   Real-ESSI DSL : add load # < . > to node # < . > type [path_time_series] [FORCETYPE] =

$< forceormoment >$ series_file $=$ "$string$";

operated over all the nodes of the defined physical group

5. gmESSI : [Add_Node_Load_Path_Series{PhysicalGroup , Force_Type , Magnitude , Time_Step , Series_File}]
   translates to series of
   Real-ESSI DSL : add load # < . > to node # < . > type [path_series] [FORCETYPE] $=$ $< forceormoment >$ time_step $= < T >$ series_file $=$ "$string$";
   operated over all the nodes of the defined physical group

6. gmESSI : [Add_Node_Load_From_Reaction{PhysicalGroup}]
   translates to series of
   Real-ESSI DSL : add load # < . > to node # < . > type [from_reactions];
   operated over all the nodes of the defined physical group

7. gmESSI : [Add_Node_Load_Imposed_Motion_Time_Series{PhysicalGroup , Dof_Type , Time_Step , Disp_Scale , Disp_File , Vel_Scale , Vel_File , Acc_Scale , Acc_File}]
   translates to series of
   Real-ESSI DSL : add imposed motion # < . > to node # < . > dof $< DOFTYPE >$ time_step $= < T >$ displacement_scale_unit $= < L >$ displacement_file $=$ "$string$" velocity_scale_unit $= < L/T >$ velocity_file $=$ "$string$" acceleration_scale_unit $= < L/T^2 >$ acceleration_file $=$ "$string$";
   operated over all the nodes of the defined physical group

8. gmESSI : [Add_Node_Load_Imposed_Motion_Time_Series{PhysicalGroup , Dof_Type , Time_Step , Disp_Scale , Disp_File , Vel_Scale , Vel_File , Acc_Scale , Acc_File}]
   translates to series of
   Real-ESSI DSL : add load # < . > type imposed motion to node # < . > dof $< DOFTYPE >$ time_step $= < T >$ displacement_scale_unit $= < L >$ displacement_file $=$ "$string$" velocity_scale_unit $= < L/T >$ velocity_file $=$ "$string$" acceleration_scale_unit $= < L/T^2 >$ acceleration_file $=$ "$string$";
   operated over all the nodes of the defined physical group

9. gmESSI : [Add_Node_Load_Imposed_Motion_Series{PhysicalGroup , Dof_Type , Disp_Scale , Disp_File , Vel_Scale , Vel_File , Acc_Scale , Acc_File}]
   translates to series of
   Real-ESSI DSL : add imposed motion # < . > to node # < . > dof $< DOFTYPE >$ displacement_scale_unit $= < L >$ displacement_file $=$ "$string$" velocity_scale_unit $= < L/T >$ velocity_file

$=$ "*string*" acceleration_scale_unit $=< L/T^2 >$ acceleration_file $=$ "*string*";
operated over all the nodes of the defined physical group

10. gmESSI : [Add_Node_Load_Imposed_Motion_Time_Series{PhysicalGroup , Dof_Type , Time_Step , Disp_Scale , Disp_File , Vel_Scale , Vel_File , Acc_Scl , Acc_File}]
translates to series of
Real-ESSI DSL : add load $\#< . >$ type imposed motion to node $\#< . >$ dof $< DOFTYPE >$ displacement_scale_unit $=< L >$ displacement_file $=$ "*string*" velocity_scale_unit $=< L/T >$ velocity_file $=$ "*string*" acceleration_scale_unit $=< L/T^2 >$ acceleration_file $=$ "*string*";
operated over all the nodes of the defined physical group

11. gmESSI : [Add_Damping_To_Node{PhysicalGroup , damping#1}]
translates to series of
Real-ESSI DSL : add damping $\#< . >$ to node $\#< . >$;
operated over all the nodes of the defined physical group

12. gmESSI : [Add_Mass_To_Node{PhysicalGroup , MassX , MassY , MassZ}]
translates to series of
Real-ESSI DSL : add mass to node $\#< . >$ mx $=< M >$ my $=< M >$ mz $=< M >$;
operated over all the nodes of the defined physical group

13. gmESSI : [Add_Beam_Mass_To_Node{PhysicalGroup , MassX , MassY , MassZ , ImassX , ImassY , ImassZ}]
translates to series of
Real-ESSI DSL : add mass to node $\#< . >$ mx $=< M >$ my $=< M >$ mz $=< M >$ Imx $=< ML^2 >$ Imy $=< ML^2 >$ Imz $=< ML^2 >$;
operated over all the nodes of the defined physical group

14. gmESSI : [Fix_Dofs{PhysicalGroup , Dof_Types}]
translates to series of
Real-ESSI DSL : fix node $\#< . >$ dofs $< DofTypes >$;
operated over all the nodes of the defined physical group

15. gmESSI : [Free_Dofs{PhysicalGroup , Dof_Types}]
translates to series of
Real-ESSI DSL : free node $\#< . >$ dofs $< . >$;
operated over all the nodes of the defined physical group

16. gmESSI : [Remove_Node{PhysicalGroup}]
    translates to series of
    Real-ESSI DSL : remove node # < . >;
    operated over all the nodes of the defined physical group

17. gmESSI : [Remove_Equal_Dof_Constrain{PhysicalGroup}]
    translates to series of
    Real-ESSI DSL : remove constraint [equal_dof] node # < . >;
    operated over all the nodes of the defined physical group

18. gmESSI : [Remove_Displacement_From_Node{PhysicalGroup}]
    translates to series of
    Real-ESSI DSL : remove displacement from node # < . >;
    operated over all the nodes of the defined physical group

207.2.5.4 General Elemental Commands : Operates On All Elements of the defined Physical Group

General Elemental Commands operates on all the elements of a physical group. The translations are written in *load.fei* file. For example:- [Add_SelfWeight_To_Element{Physical_Group#Soil,Field:= 1}] would add self-weight to all the elements of the physical group 'Soil' along the field#1 direction using series of equivalent Real-ESSI DSL 'add load # < . > to element # < . > type [self_weight] use acceleration field # < . >;'

The different commands under this category and their corresponding ESSI commands are listed below

1. gmESSI : [Add_Elements_To_Physical_Group{PhysicalGroup , Physical_Element_Group_String}]
   translates to series of
   Real-ESSI DSL : add elements (< . >) to [physical_element_group] "*string*";
   operated over all the nodes of the defined physical group

2. gmESSI : [Add_Self_Weight_To_Element{PhysicalGroup , field#1}]
   translates to series of
   Real-ESSI DSL : add load # < . > to element # < . > type [self_weight] use acceleration field # < . >;
   operated over all the nodes of the defined physical group

3. gmESSI : [Add_Damping_To_Element{PhysicalGroup , damping#1}]
   translates to series of
   Real-ESSI DSL : add damping # < . > to element # < . >;
   operated over all the nodes of the defined physical group

4. gmESSI : [Remove_Element{PhysicalGroup}]
   translates to series of
   Real-ESSI DSL : remove element # < . >;
   operated over all the nodes of the defined physical group

5. gmESSI : [Remove_Strain_From_Element{PhysicalGroup}]
   translates to series of
   Real-ESSI DSL : remove strain from element # < . >;
   operated over all the nodes of the defined physical group

207.2.5.5 Elemental Commands : Operates On All Elements of the defined Physical Group

Elemental Commands operates only to specific elements of a physical group. The translations are written in *element.fei* file. For example:- [Add_8NodeBrick{Physical_Group#Soil,1}] would initialize all the hexahedron elements of physical group 'Soil' to equivalent Real-ESSI commands for defining 8-noded bricks elements 'add element # < . > type [8NodeBrick] with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . >;'. Figure 207.6 shows how gmESSI operated on physical groups. The different commands under this category and their corresponding ESSI commands are listed below

1. gmESSI : [Add_20NodeBrick{PhysicalGroup , Num_Gauss_Points , material#1}]
   translates to series of
   Real-ESSI DSL : add element # < . > type [20NodeBrick] with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . >;
   operated over all the elements of the defined physical group
2. gmESSI : [Add_20NodeBrick_Variable_GaussPoints{PhysicalGroup , Num_Gauss_Points , material#1}]
   translates to series of
   Real-ESSI DSL : add element # < . > type [20NodeBrick] using < . > Gauss points each direction with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . >;
   operated over all the elements of the defined physical group
3. gmESSI : [Add_20NodeBrick_upU{PhysicalGroup , material#1 , Porosity , Alpha , Solid_Density , Fluid_Density , Perm_X , Perm_Y , Perm_Z , Solid_Bulk_Modulus , Fluid_Bulk_Modulus}]
   translates to series of
   Real-ESSI DSL : add element # < . > type [20NodeBrick_upU] with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . > and porosity = < . > alpha = < . > rho_s = $< M/L^3 >$ rho_f = $< M/L^3 >$ k_x = $< L^3T/M >$ k_y = $< L^3T/M >$ k_z = $< L^3T/M >$ K_s = $< stress >$ K_f = $< stress >$;
   operated over all the elements of the defined physical group
4. gmESSI : [Add_20NodeBrick_upU_Variable_GaussPoints{PhysicalGroup , Num_Gauss_Points , material#1 , Porosity , Alpha , Solid_Density , Fluid_Density , Perm_X , Perm_Y , Perm_Z , Solid_Bulk_Modulus

, Fluid_Bulk_Modulus}]

translates to series of

Real-ESSI DSL : add element # < . > type [20NodeBrick_upU] using < . > Gauss points each direction with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . > and porosity = < . > alpha = < . > rho_s = $< M/L^3 >$ rho_f = $< M/L^3 >$ k_x = $< L^3T/M >$ k_y = $< L^3T/M >$ k_z = $< L^3T/M >$ K_s = $< stress >$ K_f = $< stress >$;

operated over all the elements of the defined physical group

5. gmESSI : [Add_20NodeBrick_up{PhysicalGroup , material#1 , Porosity , Alpha , Solid_Density , Fluid_Density , Perm_X , Perm_Y , Perm_Z , Solid_Bulk_Modulus , Fluid_Bulk_Modulus}]

   translates to series of

   Real-ESSI DSL : add element # < . > type [20NodeBrick_up] with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . > and porosity = < . > alpha = < . > rho_s = $< M/L^3 >$ rho_f = $< M/L^3 >$ k_x = $< L^3T/M >$ k_y = $< L^3T/M >$ k_z = $< L^3T/M >$ K_s = $< stress >$ K_f = $< stress >$;

   operated over all the elements of the defined physical group

6. gmESSI : [Add_20NodeBrick_up_Variable_GaussPoints{PhysicalGroup , Num_Gauss_Points , material#1 , Porosity , Alpha , Solid_Density , Fluid_Density , Perm_X , Perm_Y , Perm_Z , Solid_Bulk_Modulus , Fluid_Bulk_Modulus}]

   translates to series of

   Real-ESSI DSL : add element # < . > type [20NodeBrick_up] using < . > Gauss points each direction with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . > and porosity = < . > alpha = < . > rho_s = $< M/L^3 >$ rho_f = $< M/L^3 >$ k_x = $< L^3T/M >$ k_y = $< L^3T/M >$ k_z = $< L^3T/M >$ K_s = $< stress >$ K_f = $< stress >$;

   operated over all the elements of the defined physical group

7. gmESSI : [Add_27NodeBrick{PhysicalGroup , material#1}]

   translates to series of

   Real-ESSI DSL : add element # < . > type [27NodeBrick] with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . >;

operated over all the elements of the defined physical group

8. gmESSI : [Add_27NodeBrick_upU{PhysicalGroup , material#1 , Porosity , Alpha , Solid_Density , Fluid_Density , Perm_X , Perm_Y , Perm_Z , Solid_Bulk_Modulus , Fluid_Bulk_Modulus}]
   translates to series of
   Real-ESSI DSL : add element # $<.>$ type [27NodeBrick_upU] with nodes ($<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$) use material # $<.>$ and porosity = $<.>$ alpha = $<.>$ rho_s = $< M/L^3 >$ rho_f = $< M/L^3 >$ k_x = $< L^3T/M >$ k_y = $< L^3T/M >$ k_z = $< L^3T/M >$ K_s = $< stress >$ K_f = $< stress >$;
   operated over all the elements of the defined physical group

9. gmESSI : [Add_27NodeBrick_upU_Variable_GaussPoints{PhysicalGroup , NumGaussPoints , material#1 , Porosity , Alpha , Solid_Density , Fluid_Density , Perm_X , Perm_Y , Perm_Z , Solid_Bulk_Modulus , Fluid_Bulk_Modulus}]
   translates to series of
   Real-ESSI DSL : add element # $<.>$ type [27NodeBrick_upU] using $<.>$ Gauss points each direction with nodes ($<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$) use material # $<.>$ and porosity = $<.>$ alpha = $<.>$ rho_s = $< M/L^3 >$ rho_f = $< M/L^3 >$ k_x = $< L^3T/M >$ k_y = $< L^3T/M >$ k_z = $< L^3T/M >$ K_s = $< stress >$ K_f = $< stress >$;
   operated over all the elements of the defined physical group

10. gmESSI : [Add_27NodeBrick_up{PhysicalGroup , material#1 , Porosity , Alpha , Solid_Density , Fluid_Density , Perm_X , Perm_Y , Perm_Z , Solid_Bulk_Modulus , Fluid_Bulk_Modulus}]
    translates to series of
    Real-ESSI DSL : add element # $<.>$ type [27NodeBrick_up] with nodes ($<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$, $<.>$) use material # $<.>$ and porosity = $<.>$ alpha = $<.>$ rho_s = $< M/L^3 >$ rho_f = $< M/L^3 >$ k_x = $< L^3T/M >$ k_y = $< L^3T/M >$ k_z = $< L^3T/M >$ K_s = $< stress >$ K_f = $< stress >$;
    operated over all the elements of the defined physical group

11. gmESSI : [Add_27NodeBrick_up_Variable_GaussPoints{PhysicalGroup , Num_Gauss_Points , material#1 , Porosity , Alpha , Solid_Density , Fluid_Density , Perm_X , Perm_Y , Perm_Z , Solid_Bulk_Modulus

, Fluid_Bulk_Modulus}]

translates to series of

Real-ESSI DSL : add element # < . > type [27NodeBrick_up] using < . > Gauss points each direction with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . > and porosity = < . > alpha = < . > rho_s = $< M/L^3 >$ rho_f = $< M/L^3 >$ k_x = $< L^3T/M >$ k_y = $< L^3T/M >$ k_z = $< L^3T/M >$ K_s = $< stress >$ K_f = $< stress >$;

operated over all the elements of the defined physical group

12. gmESSI : [Add_Equal_Dof{PhysicalGroup , Dof_Type}]

translates to series of

Real-ESSI DSL : add constraint [equal_dof] with master node # < . > and slave node # < . > dof to constrain < . >;

operated over all the elements of the defined physical group

13. gmESSI : [Add_Equal_Dof{PhysicalGroup , Master_Dof ,]

translates to series of

Real-ESSI DSL : add constraint [equal_dof] with node # < . > dof < . > master and node # < . > dof < . > slave;

operated over all the elements of the defined physical group

14. gmESSI : [Add_ShearBeam{PhysicalGroup , CrossSection , material#1}]

translates to series of

Real-ESSI DSL : add element # < . > type [ShearBeam] with nodes (< . >, < . >) cross_section = $< l^2 >$ use material # < . >;

operated over all the elements of the defined physical group

15. gmESSI : [Add_DispBeamColumn3D{PhysicalGroup , Num_Integr_Points , Section_Number , Density , XZ_Plane_Vect_x , XZ_Plane_Vect_y , XZ_Plane_Vect_z , Joint1_Offset_x , Joint1_Offset_y , J1_z , Joint2_Offset_x , J2_y , J2_Offset_z}]

translates to series of

Real-ESSI DSL : add element # < . > type [BeamColumnDispFiber3d] with nodes (< . >, < . >) number_of_integration_points = < . > section_number = < . > mass_density = $< M/L^3 >$ xz_plane_vector = (< . >, < . >, < . >) joint_1_offset = ($< L >$, $< L >$, $< L >$) joint_2_offset = ($< L >$, $< L >$, $< L >$);

operated over all the elements of the defined physical group

16. gmESSI : [Add_Beam_Elastic{PhysicalGroup , Cross_Section , Elastic_Modulus , Shear_Modulus , Jx , Iy , Iz , Density , XZ_PlaneVect_x , XZ_PlaneVect_y , XZ_Plane_Vect_z , Joint1_Offset_x , Joint1_y , Joint1_Offset_z , Joint2_Offset_x , Joint2_Offset_y , J2_Offset_z}]
translates to series of
Real-ESSI DSL : add element # < . > type [beam_elastic] with nodes (< . >, < . >) cross_section = $< area >$ elastic_modulus = $< F/L^2 >$ shear_modulus = $< F/L^2 >$ torsion_Jx = $< length^4 >$ bending_Iy = $< length^4 >$ bending_Iz = $< length^4 >$ mass_density = $< M/L^3 >$ xz_plane_vector = (< . >, < . >, < . >) joint_1_offset = (< L >, < L >, < L >) joint_2_offset = (< L >, < L >, < L >);
operated over all the elements of the defined physical group

17. gmESSI : [Add_Beam_Elastic_LumpedMass{PhysicalGroup , Cross_Section , Elastic_Modulus , Shear_Modulus , Jx , Iy , Iz , Density , XZ_Plane_Vect_x , XZ_Plane_Vect_y , XZ_Plane_Vect_z , Joint1_Offset_x , Joint1_Offset_y , Joint1_Offset_z , Joint2_Offset_x , Joint2_Offset_y , Joint2_Offset_z}]
translates to series of
Real-ESSI DSL : add element # < . > type [beam_elastic_lumped_mass] with nodes (< . >, < . >) cross_section = $< area >$ elastic_modulus = $< F/L^2 >$ shear_modulus = $< F/L^2 >$ torsion_Jx = $< length^4 >$ bending_Iy = $< length^4 >$ bending_Iz = $< length^4 >$ mass_density = $< M/L^3 >$ xz_plane_vector = (< . >, < . >, < . >) joint_1_offset = (< L >, < L >, < L >) joint_2_offset = (< L >, < L >, < L >);
operated over all the elements of the defined physical group

18. gmESSI : [Add_Beam_DisplacementBased{PhysicalGroup , Num_Integration_Points , Section_Number , Density}]
translates to series of
Real-ESSI DSL : add element # < . > type [beam_displacement_based] with nodes (< . >, < . >) with # < . > integration_points use section # < . > mass_density = $< M/L^3 >$ IntegrationRule = "" xz_plane_vector = (< . >, < . >, < . >) joint_1_offset = (< L >, < L >, < L >) joint_2_offset = (< L >, < L >, < L >);
operated over all the elements of the defined physical group

19. gmESSI : [Add_HardContact{PhysicalGroup , Normal_Stiffness , Tangential_Stiffness , Normal_Damping , Tangential_Damping , Friction_Ratio , Norm_Vect_x , Norm_Vect_y , Norm_Vect_z}]
translates to series of

Real-ESSI DSL : add element # < . > type [HardContact] with nodes (< . >, < . >) normal_stiffness = < F/L > tangential_stiffness = < F/L > normal_damping = < F/L > tangential_damping = < F/L > friction_ratio = < . > contact_plane_vector = (< . >, < . >, < . >);
operated over all the elements of the defined physical group

20. gmESSI : [Add_CoupledHardContact{PhysicalGroup , Normal_Stiffness , Tangential_Stiffness , Normal_Damping , Tangential_Damping , Friction_Ratio , Norm_Vect_x , Norm_Vect_y , Norm_Vect_z}]
translates to series of
Real-ESSI DSL : add element # < . > type [CoupledHardContact] with nodes (< . >, < . >) normal_stiffness = < F/L > normal_penalty_stiffness = < F/L > tangential_stiffness = < F/L > normal_damping = < F/L > tangential_damping = < F/L > friction_ratio = < . > contact_plane_vector = (< . >, < . >, < . >);
operated over all the elements of the defined physical group

21. gmESSI : [Add_SoftContact{PhysicalGroup , Initial_Normal_Stiffness , Stiffning_Rate , Maximum_Normal_Stiffness , Tangential_Stiffness , Normal_Damping , Tangential_Damping , Friction_Ratio , Norm_Vect_x , Norm_Vect_y , Norm_Vect_z}]
translates to series of
Real-ESSI DSL : add element # < . > type [SoftContact] with nodes (< . >, < . >) initial_normal_stiffness = < F/L > stiffening_rate = < $1/L$ > max_normal_stiffness = < F/L > tangential_stiffness = < F/L > normal_damping = < F/L > tangential_damping = < F/L > friction_ratio = < . > contact_plane_vector = (< . >, < . >, < . >);
operated over all the elements of the defined physical group

22. gmESSI : [Add_CoupledSoftContact{PhysicalGroup , Initial_Normal_Stiffness , Stiffning_rate , Maximum_Normal_Stiffness , Tangential_Stiffness , Normal_Damping , Tangential_Damping , Friction_Ratio , Norm_Vect_x , Norm_Vect_y , Norm_Vect_z}]
translates to series of
Real-ESSI DSL : add element # < . > type [CoupledSoftContact] with nodes (< . >, < . >) initial_normal_stiffness = < F/L > stiffening_rate = < $1/L$ > max_normal_stiffness = < F/L > tangential_stiffness = < F/L > normal_damping = < F/L > tangential_damping = < F/L > friction_ratio = < . > contact_plane_vector = (< . >, < . >, < . >);
operated over all the elements of the defined physical group

23. gmESSI : [Add_Truss{PhysicalGroup , material#1 , Cross_Sectin , Density}]
translates to series of

Real-ESSI DSL : add element # < . > type [truss] with nodes (< . >, < . >) use material # < . > cross_section = $< length^2 >$ mass_density = $< M/L^3 >$;
operated over all the elements of the defined physical group

24. gmESSI : [Add_8NodeBrick{PhysicalGroup , material#1}]
translates to series of
Real-ESSI DSL : add element # < . > type [8NodeBrick] with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . >;
operated over all the elements of the defined physical group

25. gmESSI : [Add_Cosserat8NodeBrick{PhysicalGroup , material#1}]
translates to series of
Real-ESSI DSL : add element # < . > type [Cosserat8NodeBrick] with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . >;
operated over all the elements of the defined physical group

26. gmESSI : [Add_8NodeBrick_Variable_GaussPoints{PhysicalGroup , NumGaussPoints , material#1}]
translates to series of
Real-ESSI DSL : add element # < . > type [8NodeBrick] using < . > Gauss points each direction with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . >;
operated over all the elements of the defined physical group

27. gmESSI : [Add_8NodeBrick_upU{PhysicalGroup , material#1 , Porosity , Alpha , Solid_Density , Fluid_Density , Perm_X , Perm_Y , Perm_Z , Solid_Bulk_Modulus , Fluid_Bulk_Modulus}]
translates to series of
Real-ESSI DSL : add element # < . > type [8NodeBrick_upU] with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . > porosity = < . > alpha = < . > rho_s = $< M/L^3 >$ rho_f = $< M/L^3 >$ k_x = $< L^3T/M >$ k_y = $< L^3T/M >$ k_z = $< L^3T/M >$ K_s = $< stress >$ K_f = $< stress >$;
operated over all the elements of the defined physical group

28. gmESSI : [Add_8NodeBrick_upU_Variable_GaussPoints{PhysicalGroup , Num_Gauss_Points , material#1 , Porosity , Alpha , Solid_Density , Fluid_Density , Perm_X , Perm_Y , Perm_Z , Solid_Bulk_Modulus , Fluid_Bulk_Modulus}]
translates to series of
Real-ESSI DSL : add element # < . > type [8NodeBrick_upU] using < . > Gauss points each direction with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material #

$< . >$ porosity $= < . >$ alpha $= < . >$ rho_s $= < M/L^3 >$ rho_f $= < M/L^3 >$ k_x $= < L^3T/M >$ k_y $= < L^3T/M >$ k_z $= < L^3T/M >$ K_s $= < stress >$ K_f $= < stress >$;

operated over all the elements of the defined physical group

29. gmESSI : [Add_8NodeBrick_up{PhysicalGroup , material#1 , Porosity , Alpha , Solid_Density , Fluid_Density , Perm_X , Perm_Y , Perm_Z , Solid_Bulk_Modulus , Fluid_Bulk_Modulus}]

    translates to series of

    Real-ESSI DSL : add element $# < . >$ type [8NodeBrick_up] with nodes ($< . >$, $< . >$, $< . >$, $< . >$, $< . >$, $< . >$, $< . >$, $< . >$) use material $# < . >$ porosity $= < . >$ alpha $= < . >$ rho_s $= < M/L^3 >$ rho_f $= < M/L^3 >$ k_x $= < L^3T/M >$ k_y $= < L^3T/M >$ k_z $= < L^3T/M >$ K_s $= < stress >$ K_f $= < stress >$;

    operated over all the elements of the defined physical group

30. gmESSI : [Add_8NodeBrick_up_Variable_GaussPoints{PhysicalGroup , Num_Gauss_Points , material#1 , Porosity , Alpha , Solid_Density , Fluid_Density , Perm_X , Perm_Y , Perm_Z , Solid_Bulk_Modulus , Fluid_Bulk_Modulus}]

    translates to series of

    Real-ESSI DSL : add element $# < . >$ type [8NodeBrick_up] using $< . >$ Gauss points each direction with nodes ($< . >$, $< . >$, $< . >$, $< . >$, $< . >$, $< . >$, $< . >$, $< . >$) use material $# < . >$ porosity $= < . >$ alpha $= < . >$ rho_s $= < M/L^3 >$ rho_f $= < M/L^3 >$ k_x $= < L^3T/M >$ k_y $= < L^3T/M >$ k_z $= < L^3T/M >$ K_s $= < stress >$ K_f $= < stress >$;

    operated over all the elements of the defined physical group

207.2.5.6 Elemental Compound Commands : Operates On All Surface Elements of the defined Physical Group [Surface Loads]

Elemental Compound Commands operates on two physical groups, one for surface and another for the element on which surface is present. It is used mainly for adding surface loads, which require surface number as well as element no in Real-ESSI DSL. For example:- [Add_8NodeBrick_SurfaceLoad{Physical_Group#Volume, Physical_Group#Surface, 10*Pa}] would initialize surface load of $10Pa$ on surfaces defined by physical_group 'Surface' on elements defined by physical_group 'Volume'.

The different commands under this category and their corresponding ESSI commands are listed below

NOTE:- Every Elemental commands get translated into the *load.fei* (Section 207.2.4.5).

1. gmESSI : [Add_20NodeBrick_SurfaceLoad{PhysicalGroup#Volume , PhysicalGroup#Surface , Pressure}]
   translates to series of
   Real-ESSI DSL : add load # < . > to element # < . > type [surface] at nodes (< . > , < . > , < . > , < . >, < . >, < . >, < . >, < . >) with magnitude $< Pa >$;
   operated over all the elements of the defined physical group

2. gmESSI : [Add_20NodeBrick_SurfaceLoad{PhysicalGroup#Volume , PhysicalGroup#Surface , Press1 , Press2 , Press3 , Press4 , Press5 , Press6 , Press7 , Press8}]
   translates to series of
   Real-ESSI DSL : add load # < . > to element # < . > type [surface] at nodes (< . > , < . > , < . > , < . >, < . >, < . >, < . >, < . >) with magnitudes ($< Pa >$, $< Pa >$, $< Pa >$, $< Pa >$, $< Pa >$, $< Pa >$, $< Pa >$, $< Pa >$);
   operated over all the elements of the defined physical group

3. gmESSI : [Add_27NodeBrick_SurfaceLoad{PhysicalGroup#Volume , PhysicalGroup#Surface , Pressure}]
   translates to series of
   Real-ESSI DSL : add load # < . > to element # < . > type [surface] at nodes (< . > , < . > , < . > , < . >, < . >, < . >, < . >, < . >) with magnitude $< Pa >$;
   operated over all the elements of the defined physical group

4. gmESSI : [Add_27NodeBrick_SurfaceLoad{PhysicalGroup#Volume , PhysicalGroup#Surface , Press1 , Press2 , Press3 , Press4 , Press5 , Press6 , Press7 , Press8 , Press9}]
   translates to series of

Real-ESSI DSL : add load # < . > to element # < . > type [surface] at nodes (< . > , < . > , < . > , < . >, < . >, < . >, < . >, < . >) with magnitudes (< Pa > , < Pa > , < Pa > , < Pa >, < Pa >, < Pa >, < Pa >, < Pa >, < Pa >);
operated over all the elements of the defined physical group

5. gmESSI : [Add_8NodeBrick_SurfaceLoad{PhysicalGroup#Volume , PhysicalGroup#Surface , Pressure}]
translates to series of
Real-ESSI DSL : add load # < . > to element # < . > type [surface] at nodes (< . > , < . > , < . > , < . >) with magnitude < Pa >;
operated over all the elements of the defined physical group

6. gmESSI : [Add_8NodeBrick_SurfaceLoad{PhysicalGroup#Volume , PhysicalGroup#Surface , Press1 , Press2 , Press3 , Press4}]
translates to series of
Real-ESSI DSL : add load # < . > to element # < . > type [surface] at nodes (< . > , < . > , < . > , < . >) with magnitudes (< Pa > , < Pa > , < Pa > , < Pa >);
operated over all the elements of the defined physical group

207.2.5.7 Special Commands

The translator supports some special commands to perform some special functions that are regularly required in simulations. It supports the Connect Command (Section 207.2.5.8) allows to join or create nodes between two physical groups.

207.2.5.8 Connect Command

Connect Commands creates/find layers of 2-noded elements between any two parallel geometrical physical entities like two lines, two surface or two volumes and creates a physical group of those elements and updates this information in the *XYZ.msh* file. Since gmsh does not include the feature of defining or creating 2-noded elements after the mesh creation, this command can be very useful in that case. For example;- defining contacts/interfaces, embedded piles, boundary conditions, connections etc. The command syntax for connect command is
gmESSI :: [Connect{Physical_Group#tag_From , Physical_Group#tag_To, Physical_Group#tag_Between, dir_vect, mag, no_times, $algo\_(find|create)$, tolerance,New_Physical_Group_Name}]

- Physical_Group#tag_From :: It defines the starting nodes
- Physical_Group#tag_To :: It defines the set of end nodes
- Physical_Group#tag_Between:: It defines the set of nodes where the intermediary nodes can be found, while searching. While creating nodes, it does not play any role.
- dir_vect :: It defines the direction in which the user wants to create or find the nodes. The direction vector argument is given as {x_comp \y_comp \z_comp}. Example:- {0 \0 \-1} , {1 \1 \0} .. etc.
- mag :: It defines the length of each 2-noded line elements
- no_times :: It defines number of layers of 2-noded elements, the user want to create/find
- algo_(find/create) :: It defines the algo which is either 0 or 1 meaning whether to find or create the intermediary node
- tolerance :: It defines the tolerance is required to finding the nodes. It should be less than the minimum of the distance of neighboring nodes.
- New_Physical_Group_Name :: This argument enables the user to give a name to the 2-noded new-physical group formed

Figure 207.7 graphically describes arguments of connect command.
This command updates and creates additional nodes and 2-noded elements and also assigns a physical group name "$New_Physical_Group_Name$". gmESSI automatically adds the next id available to the new physical group. The user can then manipulate this newly created physical group with any other gmESSI commands.

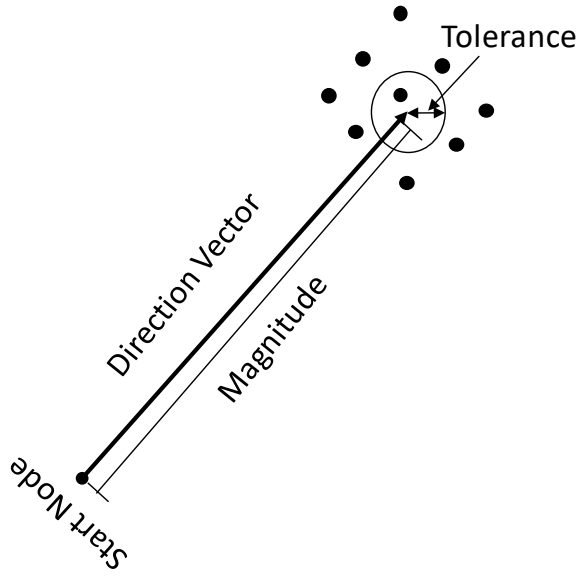


Figure 207.7: Pictorial representation of working of connect command.

The working of this command would be more clear through examples. [Example_3] can be downloaded here. [Example_4] can be downloaded here. These examples describes two situation one where new nodes are to be created and the other where already present nodes needs to be found respectively. In both the cases 2-noded line elements are always created. The examples can also be alternatively located in Examples folder of gmESSI directory

[Example_3] is a simple example where a tower of certain height above ground surface and also its base embedded in soil is modeled. It starts with a mesh file that creates a node for tower at a certain height and then using algorithm -'create' new nodes are created at certain intervals to generate the beam elements. On the other hand, the embedded beam is created by "-find" algorithm. Let us look at the [Example_3.geo] file.

```
$ cat Example_3.geo

// Size of the soil block in meter
Size = 10;

// Height of the Tower in meter
Height = 6;

// Mesh Size of the soil block
Mesh_Size = 1;

// Adding Points and extruding
Point(1)={-Size/2,-Size/2,-Size/2};
```

```
Extrude{Size,0,0}{Point{1};Layers{Size/Mesh_Size};Recombine;}
Extrude{0,Size,0}{Line{1};Layers{Size/Mesh_Size};Recombine;}
Extrude{0,0,Size}{Surface{5};Layers{Size/Mesh_Size};Recombine;}

// Make the tower located at height 6 m from the ground surface
Tower = newp;
Point (Tower) = {0,0,Size/2+Height};

//// Create Physical Groups
Physical Volume ("Soil") = {1};
Physical Surface ("Soil_Base_Surface") ={5};
Physical Surface ("Soil_Top_Surface") ={27};
Physical Point ("Tower") = {Tower};
```

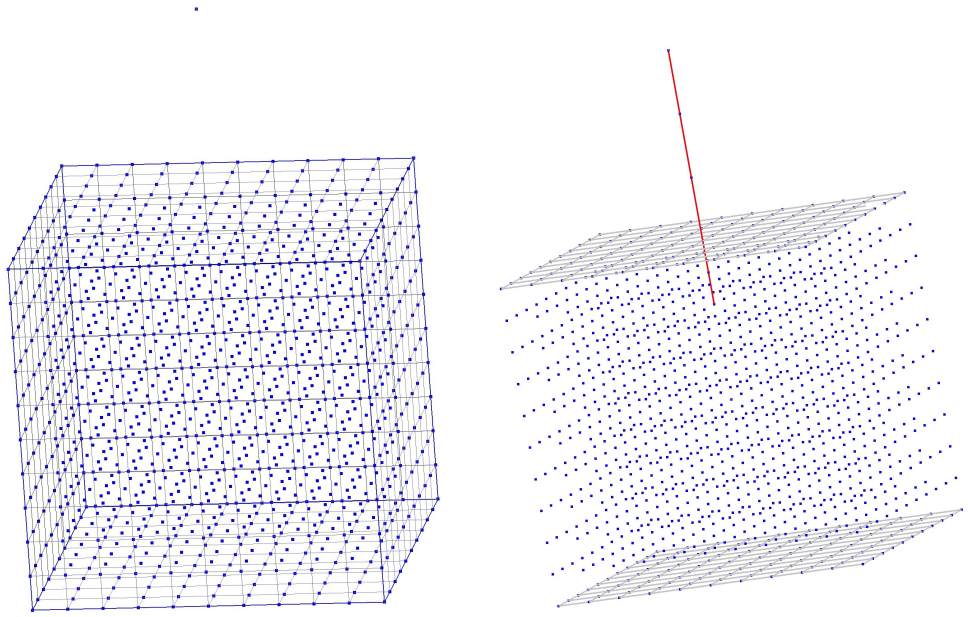
Running [Example_3.gmessi] with the .msh output of geometry file would produces additional nodes and elements as shown in Figure 207.8. An excerpt showing use of connect command with create algo in [Example_3.gmessi] is shown below. The effect of the command is shown in Figure 207.8.

```
$ cat Example_3.gmessi
..........
[Connect{Physical_Group#Tower, Physical_Group#Soil_Top_Surface, ↩
    Physical_Group#Soil_Top_Surface, dv1:= 0 \ 0 \ -1, mag:= 2, Tolerance:= 0, ↩
    algo:= create, noT:= 3, PhysicalGroupName:= Tower_Beam_Above_Ground}]
..............
```

The terminal displays the information about number of elements and nodes created and also displays the information about the new physical group information i.e id and name. The new physical group creation can be seen in the [Example_3.gmsh] in *Example_3_ESSI_Simulation* folder. The terminal message and mesh file is shown below. It also displays error message if more than one node is found in the tolerance provided.

```
$ gmessy Example_3.gmessi
New Physical Group "Tower_Beam_Above_Ground" having id 5 consisting of 4 Nodes ↩
    and 3 2-noded elements created
```

```
$ cat Example_3_ESSI_Simulation\Example_3.msh
............
$PhysicalNames
7
0 4 "Tower"
2 2 "Soil_Base_Surface"
2 3 "Soil_Top_Surface"
3 1 "Soil"
1 5 "Tower_Beam_Above_Ground"
3 6 "TowerBaseNode"
1 7 "Tower_Embedded_Beam"
$EndPhysicalNames
```



(a) Initial mesh file Example_3.msh generated by gmsh. The dir vector is in Z axis 0,0,-1

(b) Final mesh after gmESSI

Figure 207.8: Example 3 Contact Problem. (b) shows the nodes and elements generated by gmESSI Translator.

```
13 ..........
```

[Example_4] describes a foundation on soil problem with contact/interface between them. The contact element is created with the help of comment command using algo "-find". Let us look at the [Example_4.geo] file.

```
1  $ cat Example_4.geo
2
3  // Size of the soil block
4  Size = 1;
5
6  // Thickness of Foundation
7  Thick = 0.1;
8  Foundation_Layers = 2;
9
10 //// Mesh Size of the block
11 Mesh_Size = 0.2;
12
13 // Adding Points an1 extruding
14 Point(1)={-Size/2,-Size/2,-Size/2};
15 Extrude{Size,0,0}{Point{1};Layers{Size/Mesh_Size};Recombine;}
```

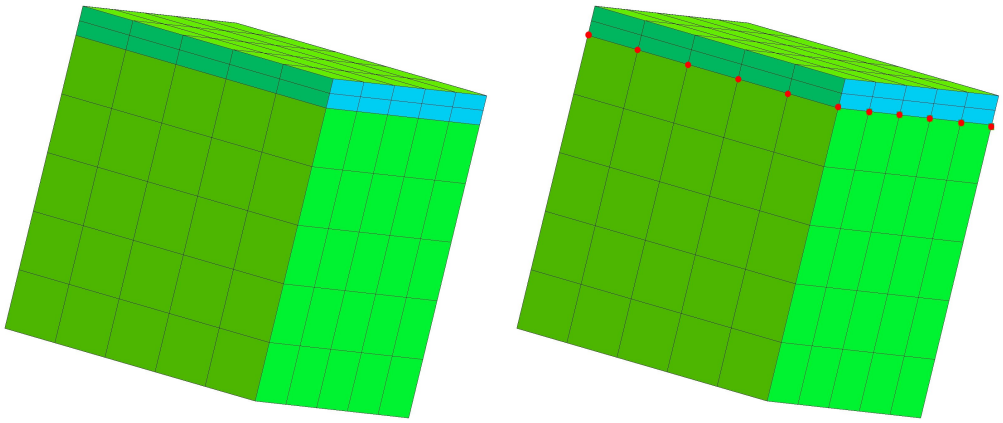
```
Extrude{0,Size,0}{Line{1};Layers{Size/Mesh_Size};Recombine;}
Extrude{0,0,Size}{Surface{5};Layers{Size/Mesh_Size};Recombine;}

// Make sure in Tools -> Geometry -> General
// Geometry tolerance is set smaller than Epsilon
// such as Geometry tolerance = 1e-14

Epsilon = 1e-8;
Translate {0, 0, Epsilon} {Duplicata{Surface{27};}}
Transfinite Line {29,30,31,32} = Size/Mesh_Size +1;
Transfinite Surface {28};
Recombine Surface {28};

//// Extruding the surface to foundation thickness
Extrude{0,0,Thick}{Surface{28};Layers{Foundation_Layers};Recombine;}

//// Create Physical Groups
Physical Volume ("Soil") = {1};
Physical Surface ("Soil_Base_Surface") ={5};
Physical Surface ("Soil_Top_Surface") ={27};
Physical Surface ("Foundation_Base_Surface")={28};
Physical Surface ("Foundation_Top_Surface") ={54};
Physical Volume ("Foundation") = {2};


Physical Surface("Fix_X") = {26, 53, 45, 18};
Physical Surface("Fix_Y") = {22, 49, 14, 41};
Physical Volume("3_Dofs") = {1,2};
```

The above geometry file is then meshed with gmsh to get the .msh file. In this file, the connect command is applied between physical group *Foundation_Base_Surface* and *Soil_Top_Surface* to create contact/interface elements. The corresponding connect command would be as

```
$ gmessy Example_4.gmessi
.........
[Connect{Physical_Group#Soil_Top_Surface, ↩
   Physical_Group#Foundation_Base_Surface, ↩
   Physical_Group#Foundation_Base_Surface, dv1:= 0\0\1, mag:= 0, Tolerance:= ↩
   0.001, algo:= find, noT:= 1, PhysicalGroupName:= Contact_Elements}]
.............
```

Similarly the updated [Example_4.msh] contains the new physical group and terminal shows the new physical group of 2-noded elements created. Figure 207.9 shows the new nodes found and creation of 2-noded elements.

```
$ gmessy Example_3.gmessi
New Physical Group "Contact_Elements" having id 10 consisting of 72 Nodes and ↩
   36 2-noded elements created
```



(a) Initial mesh file Example4.msh generated by gmsh. The dir vector is in Z axis {0,0,1}

(b) New physical group Contact_Element

Figure 207.9: Example 4 finding nodes problem.(b) shows the nodes and elements generated by gmESSI Translator.

```
$ cat Example_4_ESSI_Simulation\Example_4.msh
10
2 2 "Soil_Base_Surface"
2 3 "Soil_Top_Surface"
2 4 "Foundation_Base_Surface"
2 5 "Foundation_Top_Surface"
2 7 "Fix_X"
2 8 "Fix_Y"
3 1 "Soil"
3 6 "Foundation"
3 9 "3_Dofs"
1 10 "Contact_Elements"
.............
```

NOTE : Since the algo is to only find the nodes, so no new nodes are created, but only elements are created. The same message can be seen on the terminal.

207.2.5.9 Write Command

Write command takes filename as an argument and writes the content of a physical group in two separate files one containing all the nodes info and other containing all the elements info and places in the same XYZ_ESSI_Simulation folder. The command syntax is

gmESSI:: [Write_Data{PhyEntyTag,filename}]

- Creates files *XYZ_filename_Nodes.txt* and *XYZ_filename_Elements.txt*
- XYZ_filename_Nodes.txt :: Contains data for all nodes in a physical group. Each node data is represented in one line as

  Node_no x_coord y_cord z_cord

  with meanings as usual.
- XYZ_filename_Elements.txt :: Contains data for all elements in a physical group.Each element data is represented in one line as

  Element_no Element_type node1 node2 node3 ..

  with meanings as usual. Element_type refers to the same as in Gmsh Manual.

[Example_4.gmessi] shows the usage of write command.

207.2.5.10 Write DRM HDF5 Command

Domain reduction method (DRM) is a very useful method to input 3D seismic excitations into earthquake soil structure interacting system. With a defined physical group as DRM layer, a HDF5 file containing geometric information of the DRM layer, can be generated with the following commands for 1D, 2D and 3D mesh, respectively:

- gmESSI::[Generate_DRM_HDF5_1D{Physical_Group#<PhyEnty Name or Tag>, Surface_Normal:=<X | Y | Z>, Node_Coordinate_Tol:=<tolerance>, FileName:=<HDF5 file name>}]
- gmESSI::[Generate_DRM_HDF5_2D{Physical_Group#<PhyEnty Name or Tag>, Surface_Plane:=<XY|XZ|YZ>, Surface_Normal:= <X | Y | Z>, Node_Coordinate_Tol:=<tolerance>, FileName:=<HDF5 file name>}]
- gmESSI::[Generate_DRM_HDF5_3D{Physical_Group#<PhyEnty Name or Tag>, Surface_Normal:=<X | Y | Z>, Node_Coordinate_Tol:=<tolerance>, FileName:=<HDF5 file name>}]

Where:

- Physical_Group# defines the physical group name or tag for the DRM layer.
- Surface_Normal:= defines the surface normal direction of the DRM layer. It can be X or Y or Z.
- Surface_Plane:= defines the surface plane of the 2D DRM layer. It can be XY or YZ or XZ.

- Node_Coordinate_Tol:= defines the tolerance to distinguish two different DRM nodes. The tolerance should be much smaller than the FEM mesh size!
- FileName:= defines the file name of the HDF5 file to be generated.

207.2.6 Steps For Using gmESSI tool

Using gmESSI it is very easy to convert a .msh file to ESSI (.fei) file. This section guides the user through a simple [Example_1.geo], to show the steps necessary for generating Real-ESSI files directly from .msh file through gmESSI. Lets define a problem as shown in Figure 207.10. The [Example_1.geo] can be located in the gmESSI ''Examples' directory. Alternatively, it can be downloaded here.

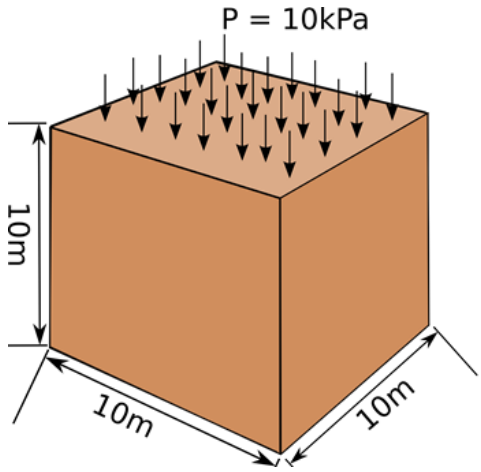


Figure 207.10: Example_1 description of a block of soil with surface load.

It is a block of dimension $10m \times 10m \times 10m$ of soil mass whose all 4 lateral faces are fixed in ux, uy dofs. The bottom face is fixed in ux, uy, uz dofs. A uniform pressure surface load of $10Pa$ is applied. The density and elastic modulus of the soil increases from $2000 * kg/m^3$ and Young's modulus is taken as $200MPa$ as shown in Figure 207.10.

207.2.6.1 Building geometry (.geo) file in Gmsh

The first step is to make the geometry file in *Gmsh*. While creating the geometry the user should also define all the physical groups on which they intend to either apply boundary condition, define elements, loads etc. In [Example_1.geo], 3 physical groups are needed : one for applying surface load, one for fixities, and one for defining the soil volume and assigning material. The content of [Example_1.geo] file is shown below

```
$cat Example_1.geo

// Size of the block
Size = 10;

//// Mesh Size of the block
Mesh_Size = 2;
```

```
8
9  // Adding Points and extruding
10 Point(1)={-Size/2,-Size/2,-Size/2};
11 Extrude{Size,0,0}{Point{1};Layers{Size/Mesh_Size};Recombine;}
12 Extrude{0,Size,0}{Line{1};Layers{Size/Mesh_Size};Recombine;}
13 Extrude{0,0,Size}{Surface{5};Layers{Size/Mesh_Size};Recombine;}
14
15 //// Create Physical Groups
16 Physical Volume ("Soil") = {1};
17 Physical Surface ("Base_Surface") = {5};
18 Physical Surface ("Lateral_Surface") = {18,22,14,26};
19 Physical Surface ("Top_Surface") ={27};
```

207.2.6.2 Generate mesh (.msh) file in Gmsh

Once .geo file is ready with all the physical groups, next step is to mesh the model. The mesh operation will generate the mesh file (.msh) that contains all the mesh information.

The model can be meshed from the terminal directly by running:

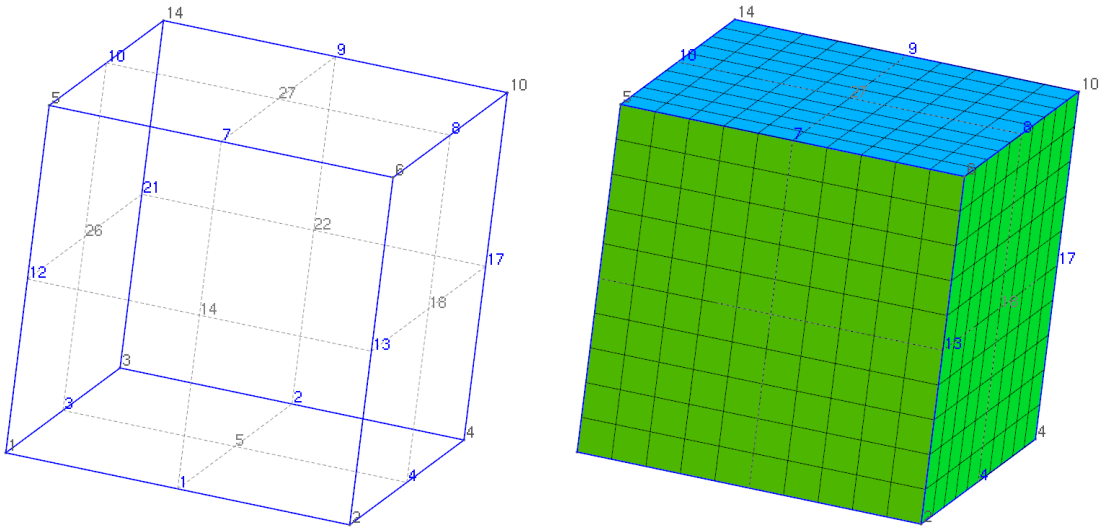
```
1 gmsh Example_1.geo -3
```

Here -3 means we are meshing a 3D object, which will automatically mesh all the 3D volumes, 2D surfaces and 1D lines object defined in the geometry model. If there are only 2D surfaces and/or 1D lines object defined in the geometry (.geo) file, use -2 instead. If there are only 1D lines object defined in the geometry (.geo) file, use -1 instead.

A quick look at the generated [Example_1.msh] file containing physical groups is shown below:

```
1  $cat Example_5.geo
2  ..............
3  $PhysicalNames
4  4
5  2 2 "Base_Surface"
6  2 3 "Lateral_Surface"
7  2 4 "Top_Surface"
8  3 1 "Soil"
9  $EndPhysicalNames
10 ................
```

Figure 207.11 shows the geometry and mesh visualization in Gmsh. It is noted that Gmsh performs meshing for linear interpolation elements by default. In other words, the above cubic block geometry object is meshed into eight-node bricks, that have linear isoparametric interpolation, `8NodeBrick`. For higher order interpolation meshing options, that is for meshing twenty-seven node brick elements mesh, `27NodeBrick` for example, additional -order int should be used. int here is the integer specifying the



(a) Geometry File (b) Mesh File

Figure 207.11: Gmsh geometry and mesh file for Example_1

order of meshing. For example, the following terminal command with -order 2, i.e., 2^{nd} order meshing, generates twenty-seven node brick meshes (`27NodeBrick`):

```
gmsh Example_1.geo -3 -order 2
```

207.2.6.3 Writing all gmESSI Commands for the model

Using gmESSI for mesh conversion is very easy. To achieve this, a [Example_1.gmessi] file is created containing all the required gmESSI commands to be executed sequentially. Let us look at each of them

Since physical group names and ids are required for referring the gmESSI commands, its always best to copy all the physical group data from the .msh file (in this case *[Example_1.msh]* file) in the header of .gmessi file, so that its easier for th user to refer to the physical groups while writing commands in .gmessi file. The contents of the .gmessi file are shown below.

```
$cat Example_1.gmessi

### Physical Groups defined in the msh file.
#2 2 "Base_Surface"
#2 3 "Lateral_Surface"
#2 4 "Top_Surface"
#3 1 "Soil"

### loading the gmsh file
```

```
gmESSI.loadGmshFile("Example_1.msh")

### Defining the Simulation Directory and node, element, load and main file
### Its important to define the directory and these files at the beginning of any gmESSI command conversion
### 1 refers as overwrite mode ( will overwrite the directory if present) --- 0 would not overwrite
gmESSI.setSimulationDir("./Example_1_ESSI_Simulation",1)
gmESSI.setMainFile(gmESSI.SimulationDir+ "main.fei")
gmESSI.setNodeFile(gmESSI.SimulationDir+ "node.fei")
gmESSI.setElementFile(gmESSI.SimulationDir+ "element.fei")
gmESSI.setLoadFile(gmESSI.SimulationDir+ "load.fei")

#### // is used to provide commands and gets translated in the main.fei file
#### Also, the commands followed by exclamation '!' get directly copied to the main.fei file
#### Usually, the user would write Real-ESSI DSL against the exclamation mark.

// My new model
! model name "Soil_Block";

[Add_All_Node{ unit:= m, nof_dofs:= 3}]

// Adding Material also assigning it to elements
! add material #1 type linear_elastic_isotropic_3d_LT mass_density = 2000*kg/m^3 elastic_modulus = 200*MPa poisson_ratio = 0.3;
[Add_8NodeBrick{Physical_Group#Soil, MaterialNo:= 1}]

! include "node.fei";
! include "element.fei";
! new loading stage "Stage1_Self_Weight";

# Applying Fixities
[Fix_Dofs{Physical_Group#Base_Surface, all}]
[Fix_Dofs{Physical_Group#Lateral_Surface, ux uy}]

#### For applying Self-Weight Load to the soil elements
! add acceleration field # 1 ax = 0*g ay = 0*g az = -1*g ;
! add load #1 to all elements type self_weight use acceleration field # 1;

 #Updating the tag inside gmESSI as user entered by himself load tag
gmESSI.setESSITag("load",2)


! include "load.fei";
! NumStep = 10;
!
! define algorithm With_no_convergence_check;
! define solver UMFPack;
! define load factor increment 1/NumStep;
! simulate NumStep steps using static algorithm;
```

```

#### updating the new load file before new loading stage
gmESSI.setLoadFile(gmESSI.SimulationDir+ "Surface_Load.fei")
! new loading stage "Stage2_Surface_Loading";

### For applying Surface load on the Top Surface of the Soil Block
[Add_8NodeBrick_SurfaceLoad{Physical_Group#Soil,Physical_Group#Top_Surface,10*Pa}]

##### For applying Nodal loads to all the nodes of the top surface
#[Add_Node_Load_Linear{Physical_Group#Top_Surface, ForceType:= Fx, Mag:= 10*kN}]


! include "Surface_Load.fei";
! NumStep = 10;
!
! define algorithm With_no_convergence_check;
! define solver UMFPack;
! define load factor increment 1/NumStep;
! simulate NumStep steps using static algorithm;

! bye;
```

NOTE: The gmESSI commands are executed and written to the file sequentially, so the user should be careful with the order of translation.

207.2.6.4 Executing gmESSI on Example_1.gmessi input file

Once .gmessi input file is ready, the next task is to run it using the 'gmessy' command in terminal. Running would carryout the translation to all and produce the log of translation, displayed on the terminal

```
$ gmessy Example_1.gmessi

Message:: newDirectory created as ./Example_1_ESSI_Simulation

Add_All_Node{ unit:= m, nof_dofs:= 3}
   Found!!
   Successfully Converted

Add_8NodeBrick{Physical_Group#Soil, MaterialNo:= 1}
   Found!!
   Successfully Converted

Fix_Dofs{Physical_Group#Base_Surface, all}
   Found!!
   Successfully Converted
```

```
16
17 Fix_Dofs{Physical_Group#Lateral_Surface, ux uy}
18    Found!!
19    Successfully Converted
20
21 Add_8NodeBrick_SurfaceLoad{Physical_Group#Soil,Physical_Group#Top_Surface,10*Pa}
22    Found!!
23    Successfully Converted
24
25
26 ********************** Updated New Tag Numbering **********************
27 damping = 1
28 displacement = 1
29 element = 126
30 field = 1
31 load = 27
32 material = 1
33 motion = 126
34 node = 217
35 nodes = 217
36 Gmsh_Elements = 276
37 Gmsh_Nodes = 217
```

It would create a folder [Example_1_ESSI_Simulation] and places load.fei, node.fei, element.fei and main.fei files. The user at this point do not need to write anything in the Example_5_analysis.fei file as every command was sequentially written down in .gmessi file and is converted. The content of the main.fei is shown below.

```
1  $cat Example_5_analysis.fei
2  // My new model
3   model name "Soil_Block";
4
5  // Adding Material also assigning it to elements
6   add material #1 type linear_elastic_isotropic_3d_LT mass_density = 2000*kg/m^3 ↩
       elastic_modulus = 200*MPa poisson_ratio = 0.3;
7
8   include "node.fei";
9   include "element.fei";
10  new loading stage "Stage1_Self_Weight";
11
12
13  add acceleration field # 1 ax = 0*g ay = 0*g az = -1*g ;
14  add load #1 to all elements type self_weight use acceleration field # 1;
15  include "load.fei";
16  NumStep = 10;
17
18   define algorithm With_no_convergence_check;
19   define solver UMFPack;
20   define load factor increment 1/NumStep;
21   simulate NumStep steps using static algorithm;
```

```
new loading stage "Stage2_Surface_Loading";

include "Surface_Load.fei";
NumStep = 10;

 define algorithm With_no_convergence_check;
 define solver UMFPack;
 define load factor increment 1/NumStep;
 simulate NumStep steps using static algorithm;
bye;
```

207.2.6.5 Running Real-ESSI and visualization in paraview

With all files ready in their place, the next step is to run the main.fei file directly in ESSI.

```
$essi -f main.fei
```

Running ESSI creates .feioutput file which can be visualized in paraview using PVESSIReader plugin. Figure 207.12 shows the visualization of hdf5 output produced in paraview.

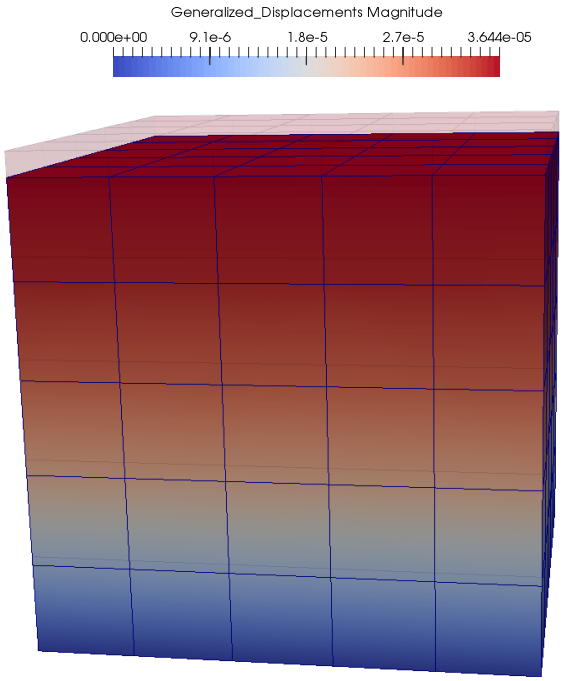


Figure 207.12: Visualizing output in Paraview.

207.2.7 Illustrative Examples

The *Examples* directory of gmESSI folder contains five examples as Example_1,Example_2.... and Example_5. They are summarized as

1. [Example_1] : Modeling of Surface load on block of Soil. The geometry (.geo), mesh (.msh) and .gmessi input files can be downloaded HERE.
2. [Example_2] : Modeling of Cantilever Beam. The geometry (.geo), mesh (.msh) and .gmessi input files can be downloaded HERE.
3. [Example_3] : Modeling of Tower (beam) located above the ground and embedded in soil using contact/interface elements. The geometry (.geo), mesh (.msh) and .gmessi input files can be downloaded HERE.
4. [Example_4] : Modeling of a concrete foundation on Soil connected by contact elements. The geometry (.geo), mesh (.msh) and .gmessi input files can be downloaded HERE.
5. [Example_5] : Modeling of a embedded shells and beam in Solids. The geometry (.geo), mesh (.msh) and .gmessi input files can be downloaded HERE.

[Example_1] was discussed in the previous section. Examples 1 to 4 are discussed and refereed in the manual at several instances. The user is encouraged to over these examples and learn to create geometry '.geo' and .gmessi input files. Here, two examples Example_2 about cantilever beam analysis and Example_5 about beams and shell is discussed.

207.2.7.1 Modeling of Cantilever Beam With Surface Load [Example_2]

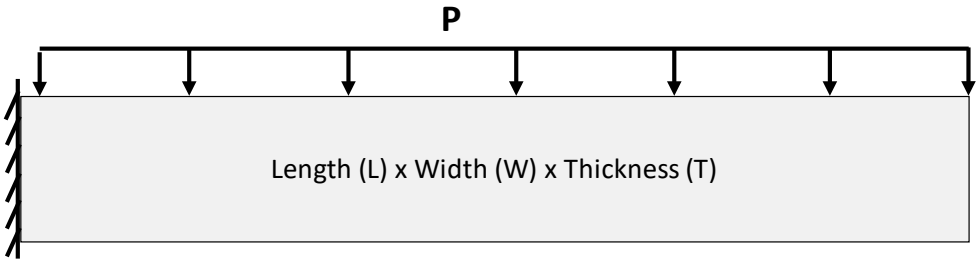


Figure 207.13: Illustration of the cantilever problem.

The problem consist of a cantilever beam with its left end fixed. A uniform surface load of P is applied. The geometry (.geo) and the gmessi input file for this problem can be downloaded here. Figure 207.14 shows visualization of output after running the model in Real-ESSI.

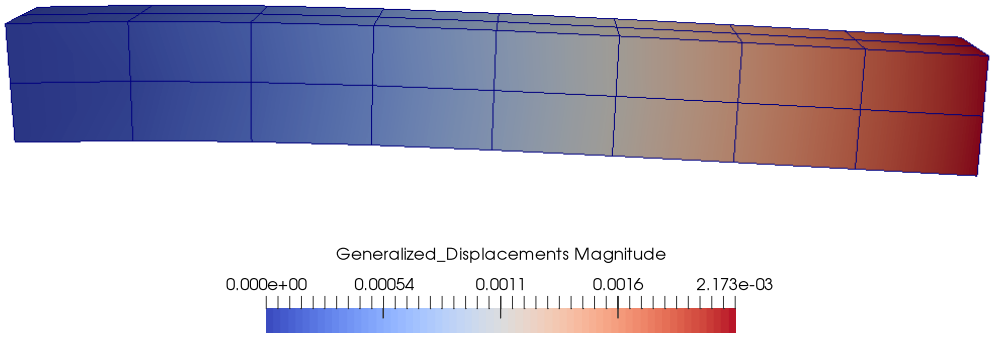


Figure 207.14: Illustration of the cantilever problem.

207.2.7.2 Modeling of a embedded shells and beam in Solids [Example_5]

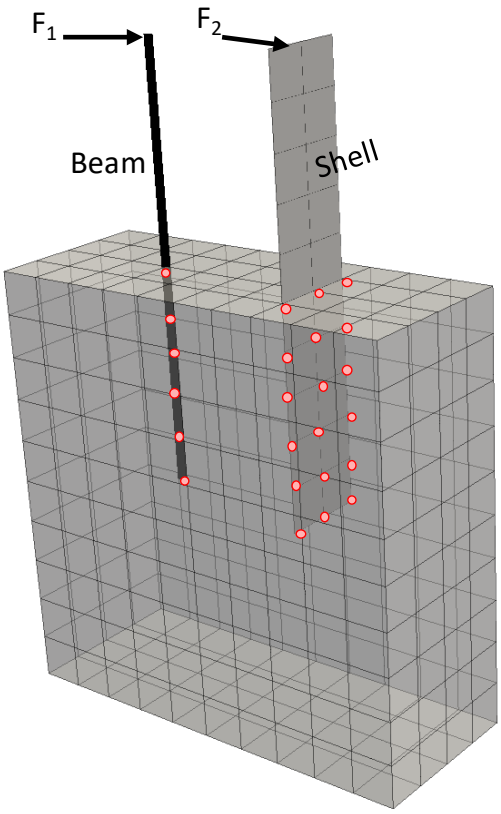


Figure 207.15: Illustration of the cantilever problem.

The problem consist of solid of 3 dofs in which beams and shells of 6dofs are embedded. The embedded beams and shell elements are connected by contact/interface elements. A nodal load to the top of the beam and shell is applied. The geometry (.geo) and the gmessi input file for this problem can be downloaded here. Figure 207.16 shows visualization of output after running the model in Real-ESSI.

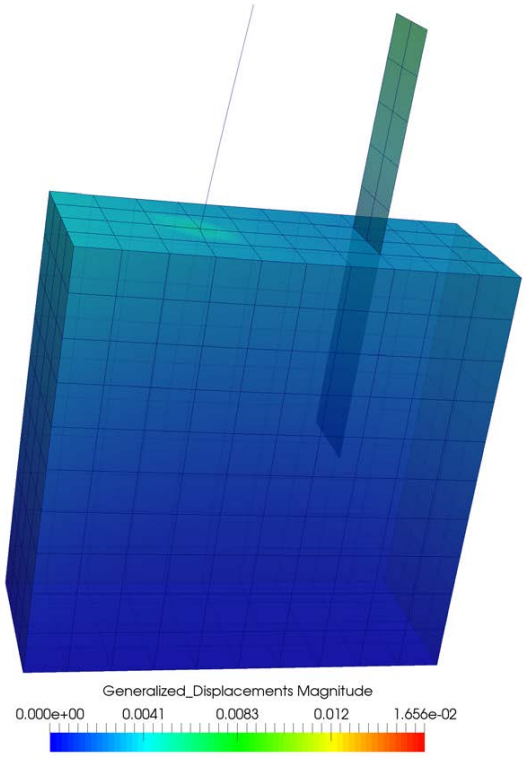


Figure 207.16: Visualizing displacement field in Paraview.

207.2.8 Realistic Models Developed Using gmESSI

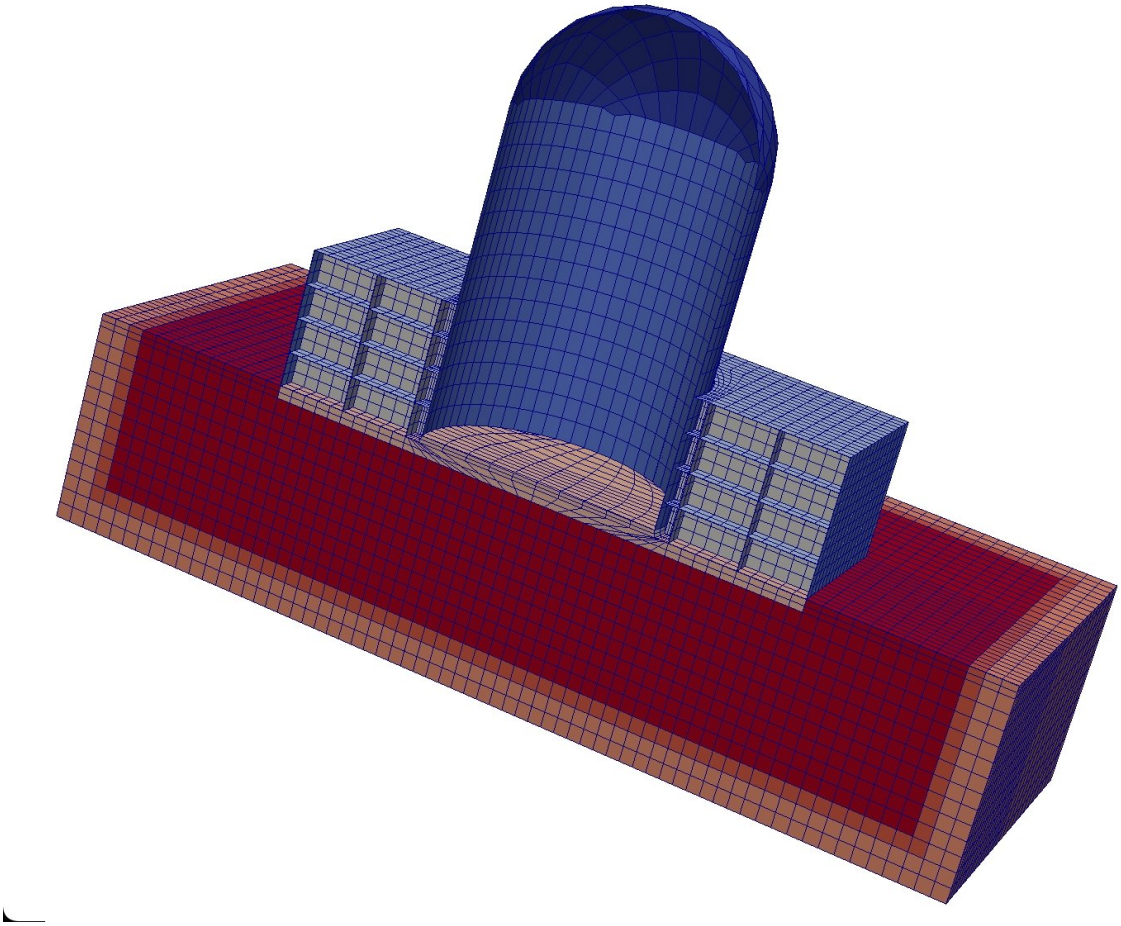


Figure 207.17: Nuclear Power Plant model 1, half model shown, with vertical plane cut.

207.3 Introduction to SASSI-ESSI Translator

This section will cover a simple mesh translator that translates mesh from SASSI format into ESSI format.

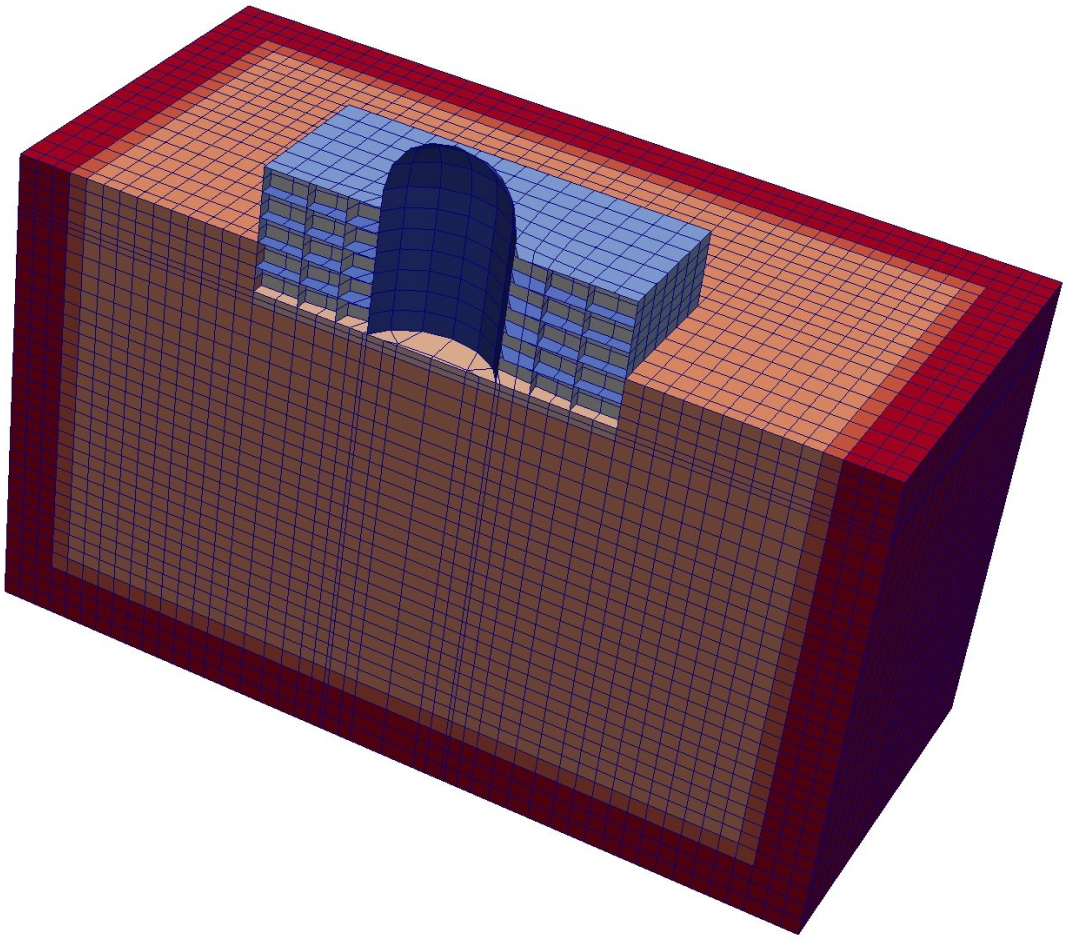


Figure 207.18: Nuclear Power Plant model 2, half model shown, with vertical plane cut.

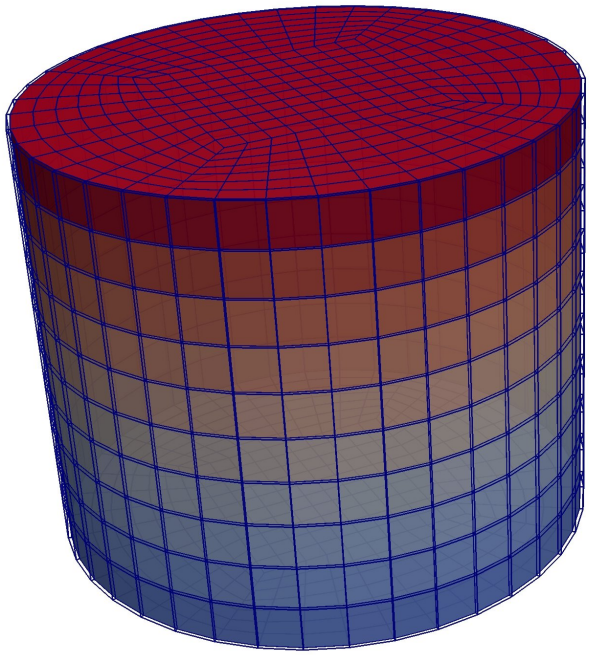


Figure 207.19: Shear Box.

Chapter 208

Real-ESSI Post Processing Methods

(2010-2014-2016-2017-2018-2019-2020-2021-)

(In collaboration with Prof. Sumeet Kumar Sinha, Dr. Yuan Feng, Prof. Han Yang, and Dr. Hexiang Wang)

208.1 Introduction

208.2 Model Results Post-Processing

This chapter describes methodology for post processing simulation results from the Real-ESSI Simulator. Two main approaches are used:

- Plotting time histories of scalar results (described in section 208.3 on page 1290), using Python and/or Matlab for:
  - components of displacements, velocities, accelerations, pore fluid pressures, for finite element nodes,
  - components of stress and/or strain at integration (Gauss) points within each finite element,
  - components of section forces for structural finite elements
  - energy input and dissipation for parts or whole of the volume/model, in incremental and/or cumulative form
- Visualization of a part or a complete model for displacements, velocities, accelerations, stress and strain components, sectional forces, energy dissipation through visualization system ParaView, as described in section 208.4 on page 1291.

208.3 Time Histories Plotting

Time histories of various scalar results (as listed above) can be extracted from output files, saved in HDF5 (Group, 2020), in a format described in chapter 206, on page 1180 in Jeremić et al. (1989-2025).

An excellent set of Python postprocessing tools was developed by Dr, Konstantinos Kanellopoulos, from ETH Zürich! You can find these tools at his github:
https://github.com/ConstantinosKanellopoulos/Real-ESSI_postprocessing_tools.

208.4 Post Processing and Visualization using ParaView

ParaView package http://www.paraview.org/ (Ayachit, 2015) is a very powerful multi-platform data analysis and visualization program available as an Open Source. Paraview can be run on supercomputers to analyze datasets of peta-scale size as well as on laptops for smaller data, and has become an integral tool in many national laboratories, universities and industry, and has won several awards related to high performance computation.

PVESSIReader is a plugin for paraview that integrates Real-ESSI Simulator output to Paraview for visualization. PVESSIReader reads Real-ESSI output file, in HDF5 format, files with extension .feioutput. The plugin works for sequential, parallel as well as remote visualization mode. It has a number of visualization features to visualize stresses, eigen modes, relative displacement, physical groups, energy dissipation, etc.

The installation of both Paraview and PVESSIReader is described in some detail in section 209.8.2 on page 1345 of the main document (Jeremić et al., 1989-2025).

208.4.1 Visualization in ParaView : Features

PVESSIReader has been consistently developed and added with lot of visualization options which are built on ParaView Visualization Toolkit (VTK) framework. This section shows all the visualization options that PVESSIReader offers other than what is available in ParaView. The features are illustrated in subsections below with help of examples in *Examples* folder of PVESSIReader source directory.
Before, looking at the features, it's important to know how PVESSIReader works. PVESSIReader takes Real-ESSI HDF5 output (.feioutput) file format as input and creates a *PVESSIReader* folder inside the HDF5 file. PVESSIReader does this to ensure the visualization to be optimized. The contents inside this folder are not important for any regular user. The contents of "PVESSIReader" folder is shown in Figure 208.1, although it is not important to regular users. User must know that, the plug-ins first creates this folder and then uses the content of this folder for visualization in ParaView. So creation/reading of this folder is the essence to visualization in ParaView.

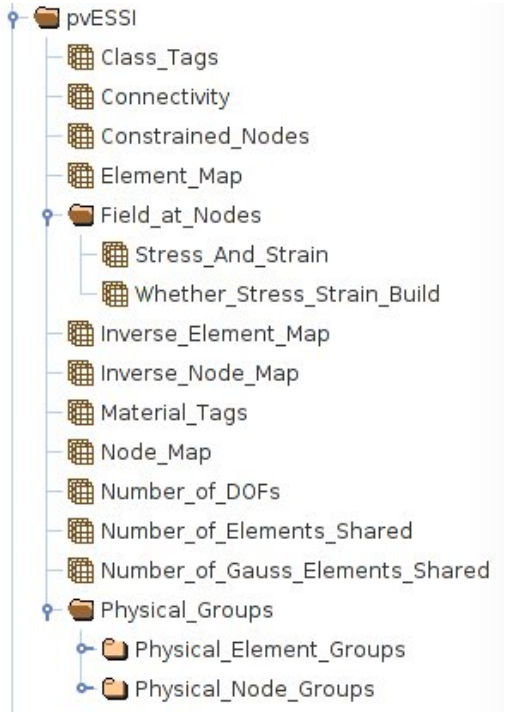


Figure 208.1: Contents of PVESSIReader folder.

208.4.1.1 PVESSIReader Visualization Options

By default PVESSIReader builds a node mesh. Figure 208.2 shows the various visualization options that is available for PVESSIReader. The Gauss to node interpolation is turned off and other options shown in Figure 208.2 is turned off. As stated in previous Section 208.4.1, the plugin creates the *PVESSIReader*

folder inside the output HDF5 file only once and uses it for rest of visualization (even after you close and reopen it). The 'Build PVESSIReader folder' button shown in Figure 208.2 on clicking rebuilds the 'PVESSIReader folder'. If in case the loading of '.feioutput' file fails in ParaView, the user should clicks the 'Build PVESSIReader Folder' before hitting 'Apply' button.

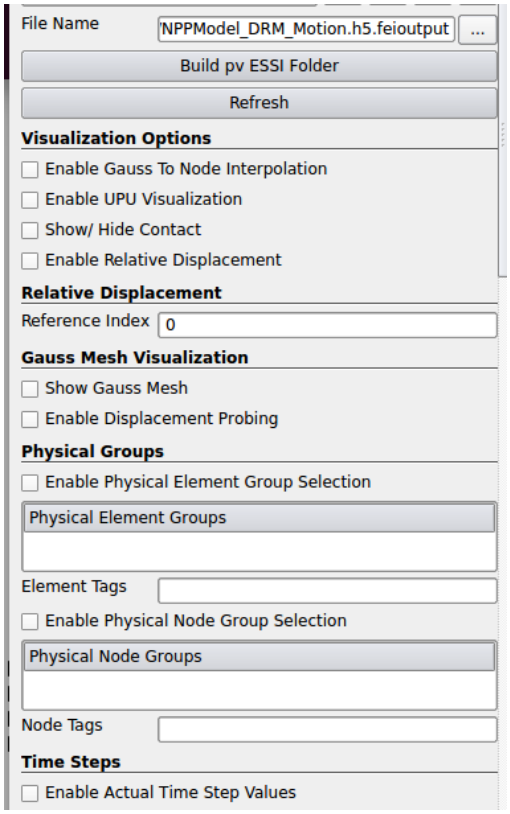


Figure 208.2: PVESSIReader build options.

Description of various PVESSIReader visualization options in the order shown in Figure 208.2 is listed below.

- Build PVESSIReader Folder - Rebuild the content of PVESSIReader folder inside output file
- Refresh - It reloads the current visualization view.
- Enable Gauss to Node Interpolation - Enables the interpolation from Gauss points to node using shape functions. It works only for 8 node and 27 node brick with 8 and 27 Gauss points respectively. See Section 208.4.1.9.
- Enable uPU Visualization - Enables the visualization of fluid displacements (U) and pore-pressure (P) at nodes. See Section 208.4.1.10.

- Enable Relative Displacement - When this is enabled, the displacement of any time step can be visualized with respect to any other reference time step. See Section 208.4.1.6.
- Reference Time Step No -This option is used to set the reference time step number about which the relative displacements would be visualized. See Section 208.4.1.6.
- Show Gauss Mesh - This option can be enabled to visualize the Gauss points (mesh) of the entire model. See Section 208.4.1.8.
- Enable Displacement Probing - This option only works if Gauss mesh option is enabled. With this option, displacements are calculated at Gauss points using ParaView interpolation functions for each elements containing those Gauss points. See Section 208.4.1.8.
- Physical Groups - This option is enabled to visualize pre-defined physical groups in Real-ESSI input or manually defined selected nodes or elements. See Section 208.4.1.12.
- Enable Actual Time Step Values - By default instead of actual simulation time (in seconds), time step number of analysis is provided to ParaView VCR. Figure 208.3 shows the result of enable/disable of this option.

(a) Actual Time Step Values Disabled

(b) Actual Time Step Values Enabled

Figure 208.3: Illustration of difference between enable and disable of `actual time step values`.

In the Figure 208.3, the enabled option gives the exact simulation time of $3.895s$. Whereas, disabling the option shows time step number of 200.

The visualization gets automatically updated on enable/disable of options. When one hit's apply, the corresponding changed result gets updated i.e. the mesh would get real time updated with enable/disable of these options.

208.4.1.2 Sequential Visualization

Sequential visualization means visualizing the Real-ESSI output on a single core of laptop/desktop. This is used for single output files that Real-ESSI produces for sequential runs. Figure 208.4 shows the visualization of output file produced by sequential Real-ESSI simulation.

```
cd pvESSI/Examples
ParaView ShearBox_Sequential.h5.feioutput
```

The parallel output files of ESSI can also be visualized sequentially. Each individual (core) file can be sequentially visualized showing only a part of the model results. Also all the parallel files at once can be opened as well in ParaView as shown in Figure 208.5. All PVESSIReader examples can be obtained at http://sokocalo.engr.ucdavis.edu/~jeremic/lecture_notes_online_material/Real-ESSI_pvESSI/Examples.

1. To open one core output in sequential

```
cd pvESSI/Examples
ParaView ShearBox_Parallel.h5.1.feioutput
```

2. To open all cores output in parallel

```
cd pvESSI/Examples
ParaView ShearBox_Parallel.h5.feioutput
```

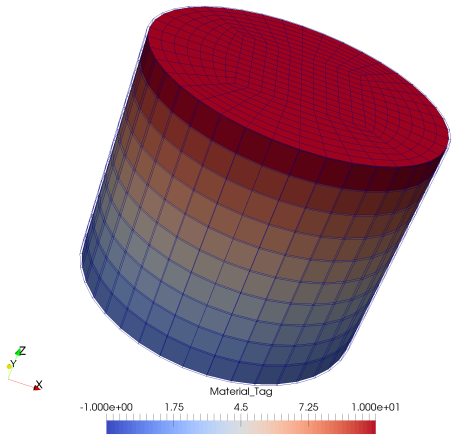
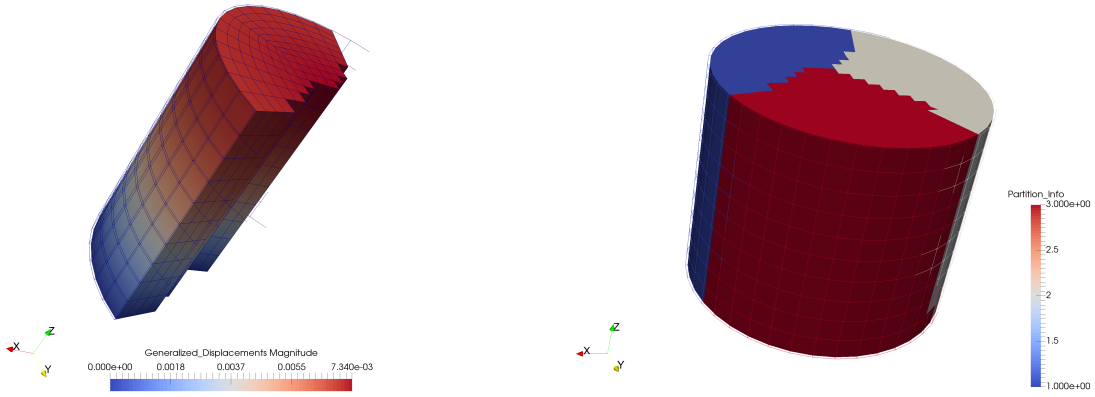


Figure 208.4: Sequential Visualization of output produced by sequential Real-ESSI simulation.



(a) Visualizing only one slave output file

(b) Visualizing all slave output files at once

Figure 208.5: Sequential Visualization of output produced by parallel Real-ESSI simulation.

208.4.1.3 Remote Visualization

Remote visualization is an important feature that ParaView offers. This is an important feature in need, when simulations are run on super computers with thousands of cores. The steps for remote visualization are shown below with an example on local desktop.

1. Run pvserver on server

```
$pvserver
Waiting for client...
%Connection URL: cs://sumeet:11111
%Accepting connection(s): sumeet:11111
Connection URL: cs://jeremic:11111
Accepting connection(s): jeremic:11111
```

2. Open ParaView on client side and click on connect button located on top left window.

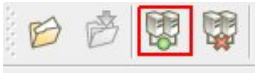
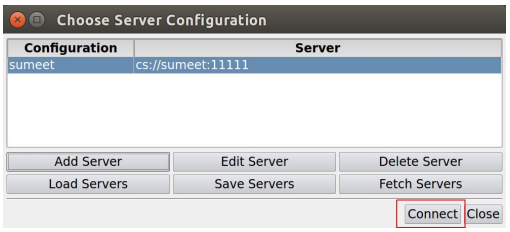
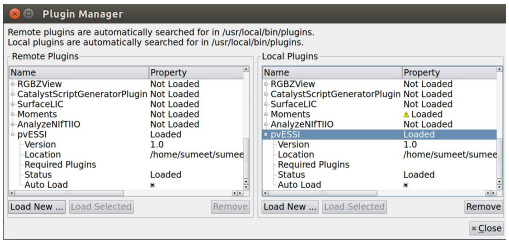


Figure 208.6: Connect Server.

3. Select and connect to the server and then load the plugins on both client and server side as shown in Figure 208.7



(a) Connect to Server



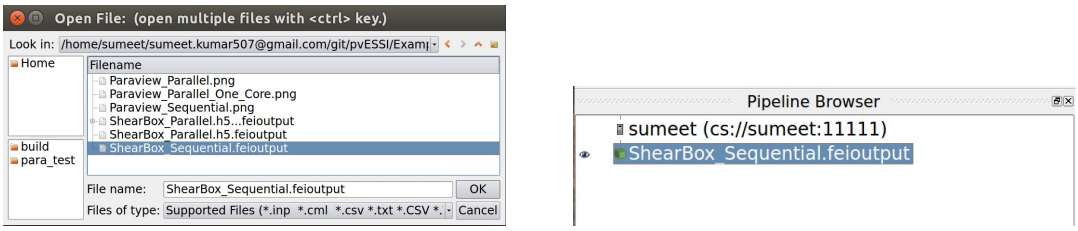
(b) All Core Outputs

Figure 208.7: Connect to the server and load plugins.

4. Navigate to *pvESSI/Examples/ShearBox_Parallel.h5.1.feioutput* and hit apply

208.4.1.4 Parallel Visualization

Parallel visualization is similar to remote visualization. The only difference is to start the pvserver in parallel on multiple cores. It is recommended to have the same number of cores that was used in Real-ESSI parallel simulation. Below shows steps on how to do parallel visualization in ParaView.



(a) Locating file on server side

(b) Client Side Pipeline

Figure 208.8: ParaView Remote Visualization.

```
$ mpirun -np $(nop) pvserver
# $(nop) is replaced by number of cores on which parallel visualization is to be run.
```

The next following steps are same to that of Remote visualization as shown in Section 208.4.1.3. Thus, parallel visualization can be performed remotely as well as locally.

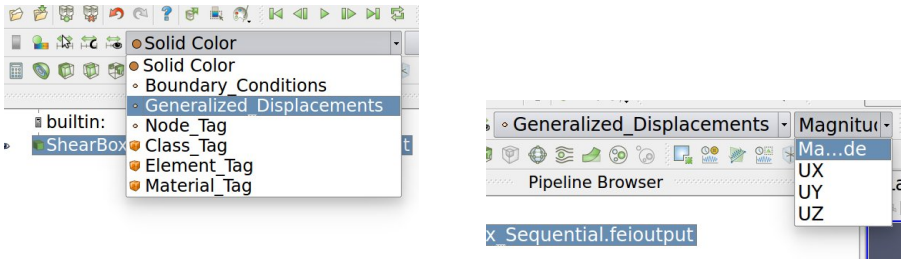
208.4.1.5 General Field Visualization

Below is the list of general visualization variables available for any model in ParaView using PVESSIReader plugin. The following subsections describes each option through an example. The examples can be found in pvESSI/Examples directory. The example file '`ShearBox_Sequential.h5.feioutput`' can be downloaded here.

```
cd pvESSI/Examples
ParaView ShearBox_Sequential.h5.feioutput
```

Displacement Field : The displacement field represents the total displacement from the start of the Real-ESSI simulation. There are two modes of displacement field visualization available in ParaView. NOTE: Please remember to change step number from 0 to any other step number, as all output, including displacements, is 0.0 at step 0.

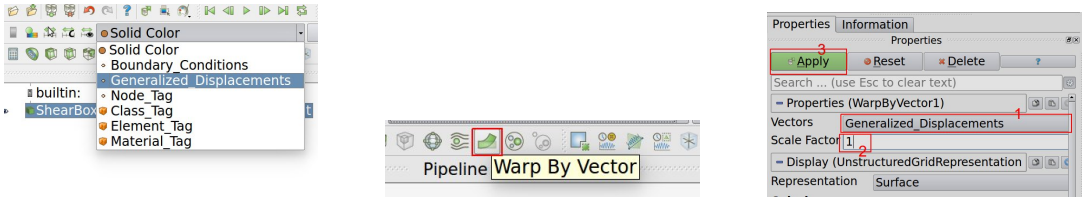
1. Scalar field visualization : The options available are each individual displacement vector components u_x, u_y and u_z in x,y,z directions respectively. It also shows the displacement magnitude $|u| = \sqrt{u_x^2 + u_y^2 + u_z^2}$. The units of displacements field in $[m]$.



(a) Select Displacement Field (b) Select scalar field

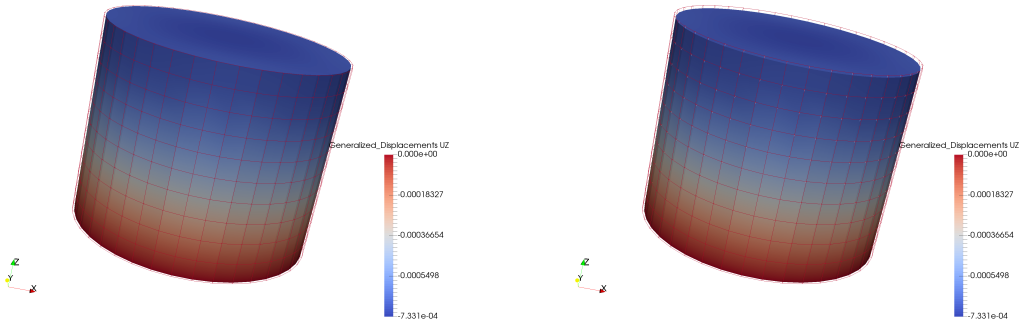
Figure 208.9: Displacement Scalar Field Visualization.

2. Vector Field visualization : This is achieved using '`Wrap by Vector`' plugin available in ParaView. Figure 208.10 and Figure 208.11 shows steps to visualize deformed mesh.



(a) Select Displacement Field (b) Select plugin (c) Plugin properties

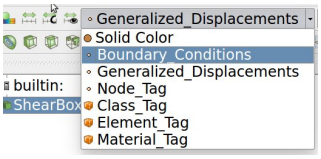
Figure 208.10: Deformation Visualization.



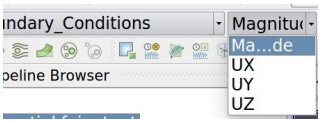
(a) Displacement Field Visualization in Uz (b) Deformation Visualization

Figure 208.11: Displacement field visualization in ParaView.

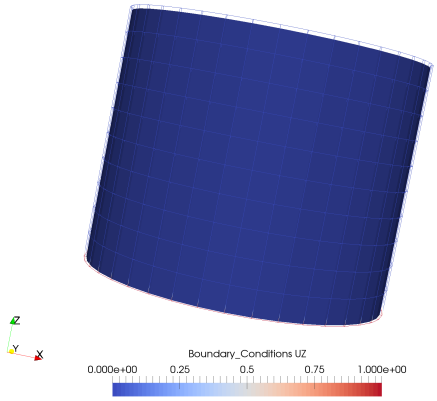
Boundary Conditions: Again this a vector field which contains information about boundary conditions i.e fixities applied in u_x, u_y and u_z directions. A value of 1 means the node is fixed while 0 means it is free. Figure 208.12 shows steps to visualize boundary conditions.



(a) Select Boundary Conditions



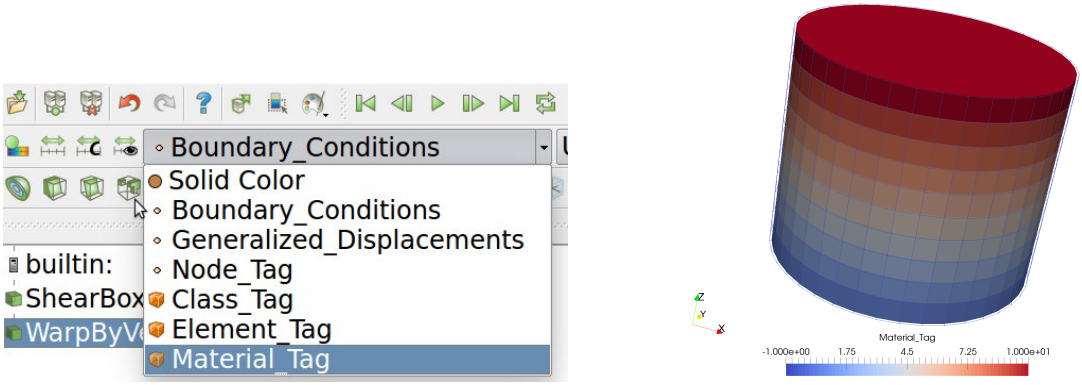
(b) Select Fixities Type



(c) Boundary Conditions in Uz

Figure 208.12: Boundary Conditions Visualization.

Material Tag Visualization : This is a scalar field visualization that shows the *material tag no* associated with the elements in Real-ESSI simulation. Figure 208.13 shows steps to visualize element's material tag.

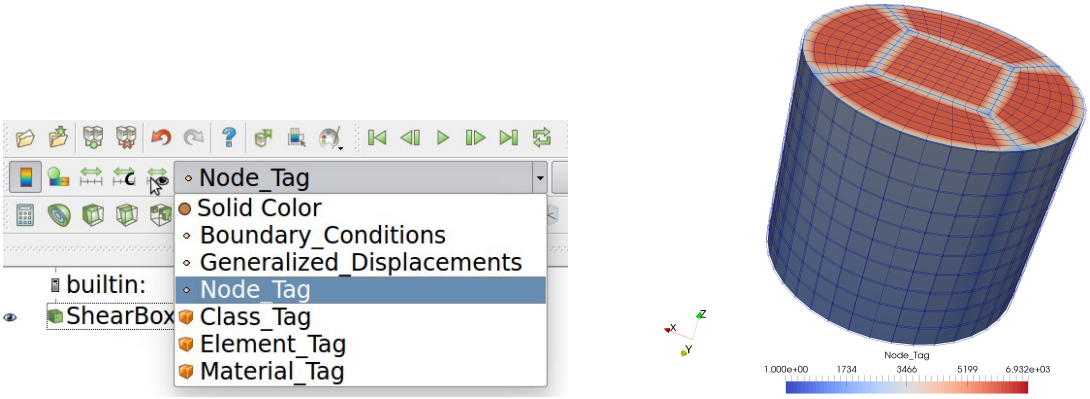


(a) Select Material Tag

(b) Material Tag Field

Figure 208.13: Material Tag Visualization.

Node Tag Visualization : This is a scalar field visualization that shows the `node no` associated with the nodes in Real-ESSI simulation. Figure 208.14 shows steps to visualize node's tag.

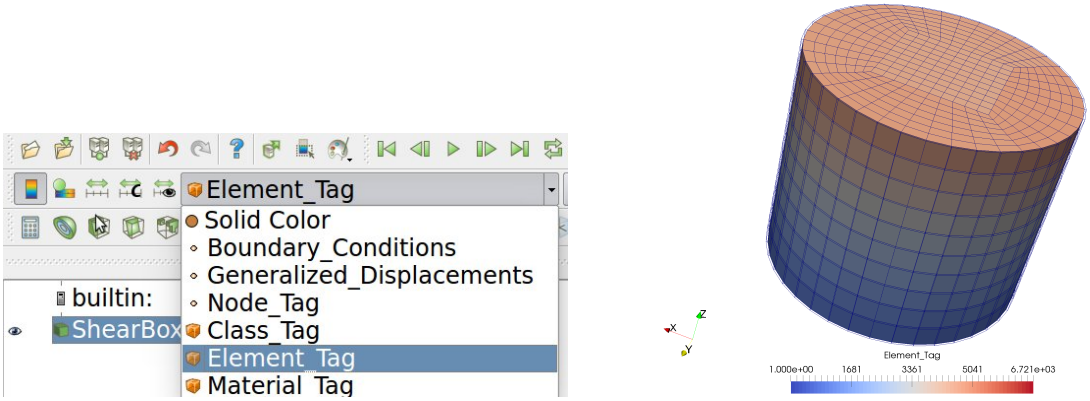


(a) Select Node Tag

(b) Node Tag Field

Figure 208.14: Node Tag Visualization.

Element Tag Visualization : This is a scalar field visualization that shows the `element no` associated with the elements in Real-ESSI simulation. Figure 208.15 shows steps to visualize element's tag.

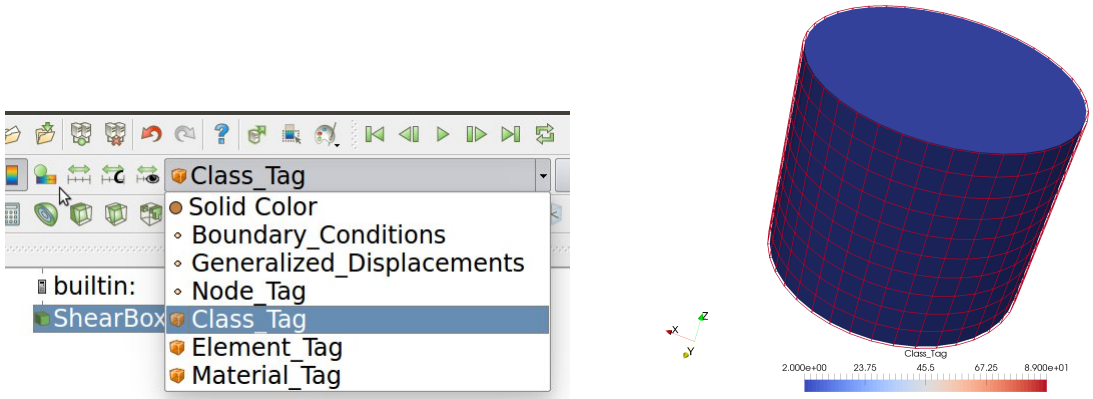


(a) Select Element Tag

(b) Element Tag Field

Figure 208.15: Element Tag Visualization.

Element Class Tag Visualization : This is a scalar field visualization that shows the Element's Class Tag number associated with the each *element type* in Real-ESSI simulation. Figure 208.15 shows steps to visualize element's tag. Section 206.5.4.4 shows the class tag for various element types available in Real-ESSI.



(a) Select Class Tag

(b) Class Tag Field

Figure 208.16: Class Tag Visualization.

208.4.1.6 Relative Displacement Visualization

When the '`Enable Relative Displacement`' is checked, the relative displacement visualization option becomes active. BY default, the relative displacement time step number is set to '0' as shown in Figure 208.17. Time step number '0' corresponds to initial conditions of the loading stage output file.

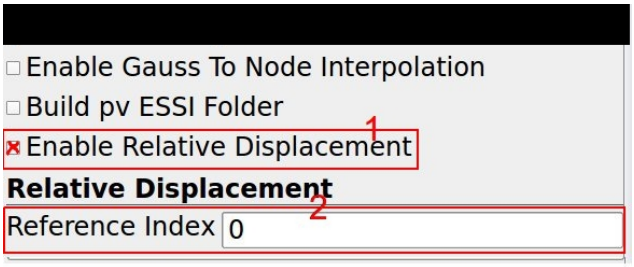


Figure 208.17: Enable Relative Displacement.

Reference Time Step Number - It defines the relative time step index number for relative displacement visualization. By default it is set to 0 i.e. to the initial conditions. Its very useful, when one wants to visualize deformation coming from the stage itself. For example:- Separating self-weight from Static Pushover Analysis in the Shear Box simulation. The steps to do the same is shown below. The example file `ShearBox_PushOver.h5.feioutput` can be downloaded here.

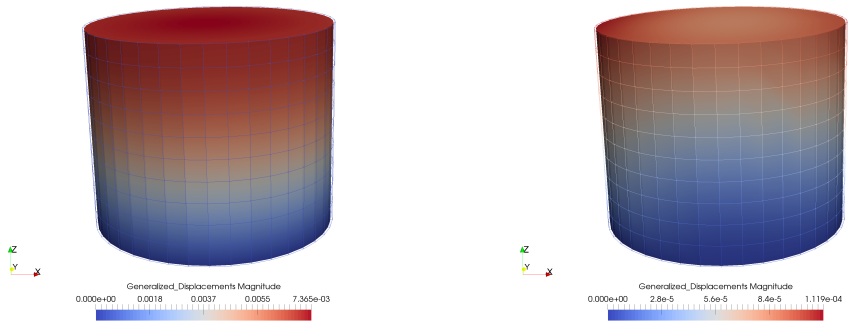
1. Open an example in ParaView

```
cd pvESSI/Examples
ParaView ShearBox_PushOver.h5.feioutput
```

2. Check on `Enable Relative Displacement` under visualization options

3. Apply `warp by vector` plugin and follow the steps as shown in Figure 208.10

Figure 208.18 shows the visualization with and without relative displacement.



(a) Without Relative Displacement (b) With Relative Displacement

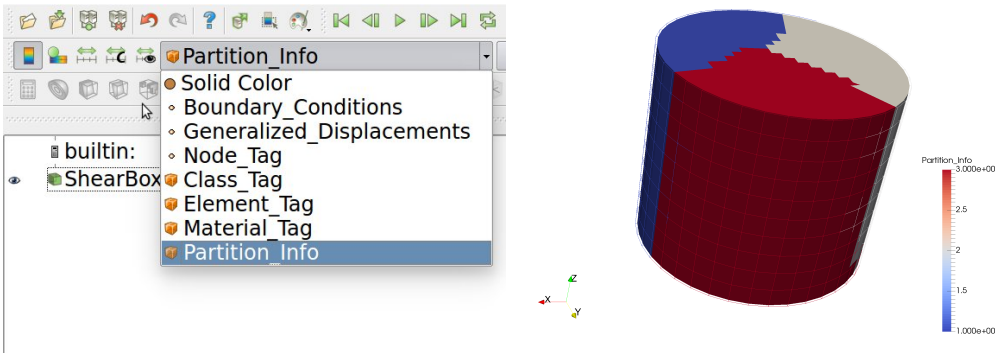
Figure 208.18: Pushover analysis of Shear Box after self-weight load application.

208.4.1.7 Visualizing Element's Partition

If the ESSI simulation was run in parallel mode, it becomes important to visualize the elements distribution between different cores. In ParaView, one can see the element distribution by selecting "`Partition Info`". Following is shown in Figure 208.19 an example to visualize mesh partitioning. All the example files can be obtained here.

```
cd pvESSI/Examples
ParaView ShearBox_Parallel.h5.feioutput
```

and then select Partition_Info as shown below in Figure 208.19



(a) Select Partition Info (b) Mesh Partitioning

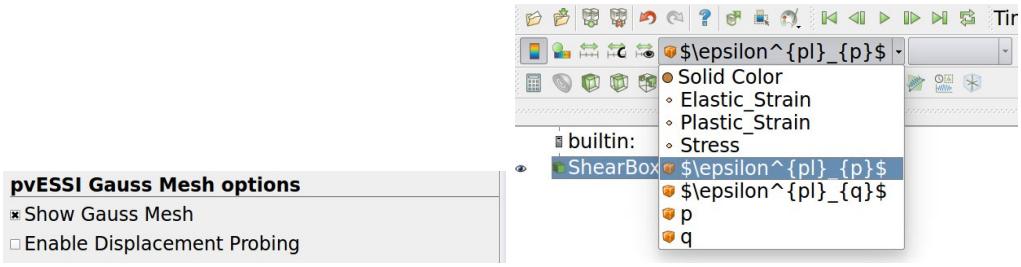
Figure 208.19: Visualizing mesh partitioning.

208.4.1.8 Gauss Mesh Visualization Options

Often, it is required to visualize stress and strain fields. Since stress or strains are evaluated at Gauss points in 3-D elements, Gauss mesh is needed to visualize them. PVESSIReader offers option to visualize Gauss mesh and it's fields.

- Show Gauss Mesh - Shows only Gauss mesh with Gauss attributes.
- Enable Displacement Probing - When this option is enabled, displacements are probed to the Gauss location. Its useful in the situation, when one wants to visualize the change in stress with deformation. With this as active, one can apply '`warp by vector`' filter.

It must be noted that the `Enable Displacement Probing` options only works when `Show Gauss Mesh` mode is enabled. Figure 208.20 shows the steps to visualize Gauss mesh.



(a) Gauss Mesh Options (b) Gauss Mesh Fields

Figure 208.20: Visualizing Gauss mesh and it's fields.

The various fields that can be visualized in Gauss mesh mode as shown in Figure 208.20 are shown below.

- Total Strain ϵ : It defines the total strain from the start of the simulation. It has six independent component $\epsilon_{xx}, \epsilon_{xy}, \epsilon_{xz}, \epsilon_{yy}, \epsilon_{yz}$ and ϵ_{zz}. The magnitude of the total stress in ParaView is defined as $\sqrt{\epsilon_{ij} : \epsilon_{ij}}$.
- Total Plastic Strain ϵ^{pl} : It defines the total plastic strain from the start of the simulation. It has six independent component $\epsilon^{pl}_{xx}, \epsilon^{pl}_{xy}, \epsilon^{pl}_{xz}, \epsilon^{pl}_{yy}, \epsilon^{pl}_{yz}$ and ϵ^{pl}_{zz}. The magnitude of the total plastic strain in ParaView is defined as $\sqrt{\epsilon^{pl}_{ij} : \epsilon^{pl}_{ij}}$.
- Total Effective Stress $\sigma\prime$: It defines the total effective stress from the start of the simulation. It has six independent component $\sigma'_{xx}, \sigma'_{xy}, \sigma'_{xz}, \sigma'_{yy}, \sigma'_{yz}$ and σ'_{zz}. The magnitude of the total effective stress in ParaView is defined as $\sqrt{\sigma'_{ij} : \sigma'_{ij}}$. The unit of visualization is in $[Pa]$.

- Total Mean Effective Stress p : It defines the total mean of the effective stress σ' from the start of the simulation. It is defined as $p = -\sigma'_{ii}/3$ as described in Equation ??. The unit of Visualization is in $[Pa]$.

- Total Deviatoric Effective Stress q : It defines the deviatoric invariant of the total effective stress σ' from the start of the simulation. It is defined as $q = \sqrt{3J_2}$ as described in Equation ??. Where, J_2 is the second invariant of the deviatoric stress tensor $s_{ij} = \sigma'_{ij} - \sigma'_{kk}/3\delta_{ij}$. The unit of visualization is in $[Pa]$.

- Total Mean Plastic Strain ϵ_p^{pl} : It defines the mean total plastic strain ϵ^{pl} invariant from the start of the simulation. It is defined as $\epsilon_p^{pl} = -\epsilon_{ii}^{pl}/3$. This visualization parameter is unit-less.

- Total Deviatoric Plastic Strain ϵ_p^{pl} : It defines the deviatoric invariant of the total plastic strain ϵ^{pl} from the start of the simulation. It is defined as $\epsilon_p^{pl} = \sqrt{3J'_2}$. Where, J'_2 is the second invariant of the deviatoric plastic strain tensor $e_{ij}^{pl} = \epsilon_{ij}^{pl} - \epsilon_{kk}^{pl}/3\delta_{ij}$. This visualization parameter is unit-less.

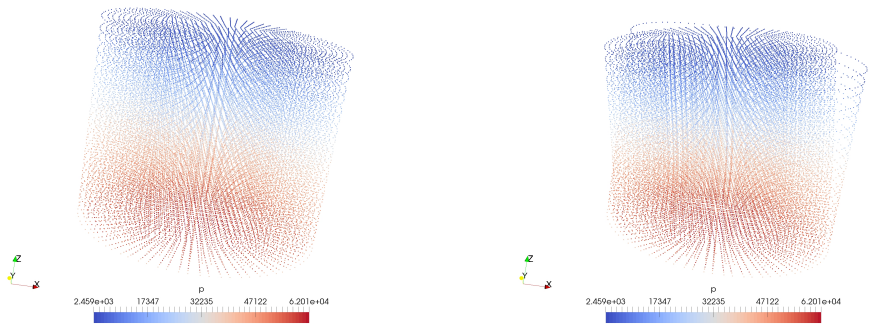
1. Open an example in ParaView. All the example files can be obtained at http://sokocalo.engr.ucdavis.edu/~jeremic/lecture_notes_online_material/Real-ESSI_pvESSI/Examples.

```
cd pvESSI/Examples
ParaView ShearBox_PushOver.h5.feioutput
```

2. Check on *Enable Relative Displacement* under PVESSIReader build options

3. Enable Gauss mesh as shown in Figure 208.20(a). Select `Mean Effective Stress` p `[Pa]`. The resulting visualization is shown in Figure 208.21(a).

4. Enable `displacement probing` as shown in Figure 208.20(a). Apply a `warp by vector` filter and select the vector `displacement` as shown in Figure 208.10. Now select again the `Mean Effective Stress` p `[Pa]` field option to visualize. The resulting visualization is shown in Figure 208.21(b).



(a) Gauss Mesh

(b) Gauss Mesh with displacement probing

Figure 208.21: Visualization of mean effective stress p invariant in Gauss mesh.

208.4.1.9 Gauss To Node Interpolation Mode Visualization

This visualization mode can be enabled by checking the 'Gauss To Node Interpolation' option as shown in Figure 208.22(a). In this mode, the total effective stress σ'_{ij}, total strain ϵ_{ij}, total plastic strain ϵ^{pl}_{ij}, total mean effective stress p, total deviatoric effective stress q, total mean plastic strain ϵ^{pl}_{p} and total deviatoric plastic strain ϵ^{pl}_{q} are interpolated from the Gauss points to the nodes of individual element. Individual shape functions of the element (with full Gauss integration) are used to obtain the stress or strain field at nodes. To smooth out the jumps in stress or strain field at the node by adjacent elements, unweighted averaging is performed. For the elements (usually structural) with no Gauss points, the stress or strain contribution at nodes are considered as zero. While taking the averaging, their contributions are not taken, as Real-ESSI does not output stress/strain for them.

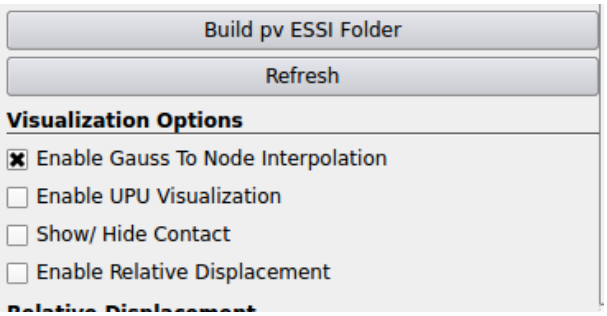
In this mode, visualization of all the parameters listed and described in Section 208.4.1.8 is available. Figure 208.22 show the steps to enable and use `Gauss to Node Interpolation` option.

1. Open an example in ParaView. All the example files can be obtained at http://sokocalo.engr.ucdavis.edu/~jeremic/lecture_notes_online_material/Real-ESSI_pvESSI/Examples.

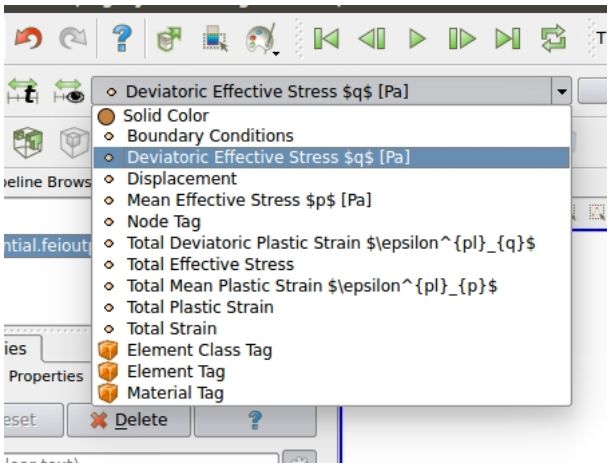
```
cd pvESSI/Examples
ParaView ShearBox_Sequential.feioutput
```

2. Follow the steps as shown in Figure 208.22

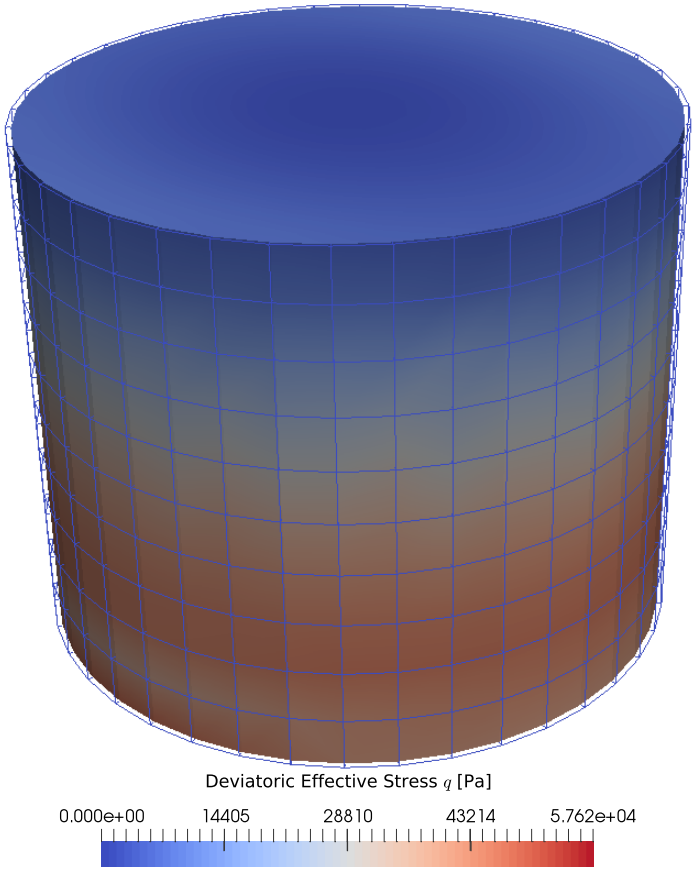
Note : The option `Gauss to node interpolation` is provides only an approximate estimate for stress and strains at nodes. The values obtained at nodes is not accurate and thus `Gauss Mesh Visualization` option described in Section 208.4.1.8 must be performed to get the accurate stress and strains at Gauss points. Also, it must be noted that this option works only for 8 node brick with 8 Gauss points and 27 node brick with 27 node points. For elements which have less number of nodes that Gauss points, the total number of equations (unknowns) is not equal to constraints (knowns). In this case, only the shape function defined at the nodes are used to get the stress or strain back to the node.



(a) Enable Gauss To Node Visualization Option



(b) Select the visualization variable



(c) Visualization of Deviatoric Stress interpolated to nodes

Figure 208.22: Steps to visualize stress and strain interpolated from Gauss points to nodes.

208.4.1.10 upU Visualization

This mode is to visualize the upU elements used in Real-ESSI simulation. Enabling this mode, produces additional outputs of '`Pore Pressure` $p[Pa]$' and '`Fluid Displacement` $U_x[m]$, $U_y[m]$ `and` $U_z[m]$' at nodes. These additional outputs are described below.

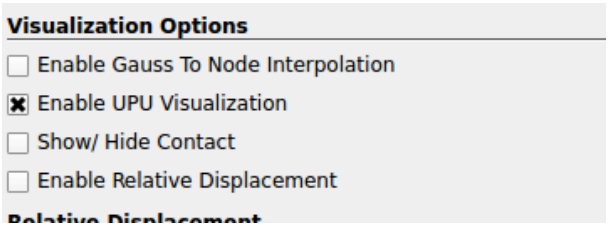
- Pore Pressure $p[Pa]$: It defines the pore-fluid pressure in the upU element at the nodes. The magnitude of the pore pressure is $[Pa]$.
- Fluid Displacement $U[m]$: It defines the displacement by the fluid particles of upU at nodes. The units is in meters $[m]$. The solid displacement is termed as u and refers to the '`Displacement` u' variable in visualization as described in Section 208.4.1.5.

Since general dry elements does not have any fluid, enabling this option would produce '`zero`' pore fluid pressure and fluid displacements at nodes. Below is shown an example that shows how to use the upU visualization feature. Figure 208.23 shows the steps.

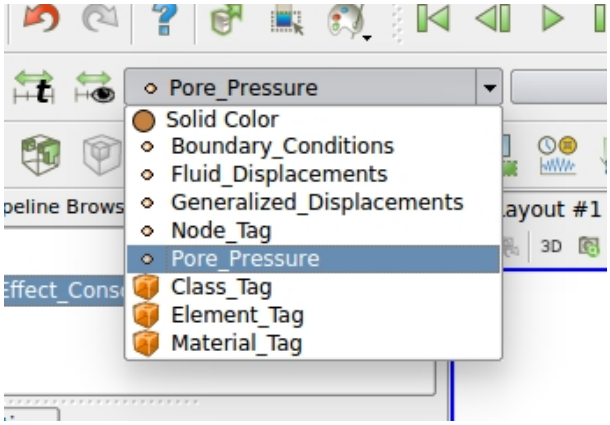
1. Open an example in ParaView. All the example files can be obtained at `http://sokocalo.engr.ucdavis.edu/~jeremic/lecture_notes_online_material/Real-ESSI_pvESSI/Examples`.

```
cd pvESSI/Examples
ParaView upU_Visualization_Example.feioutput
```

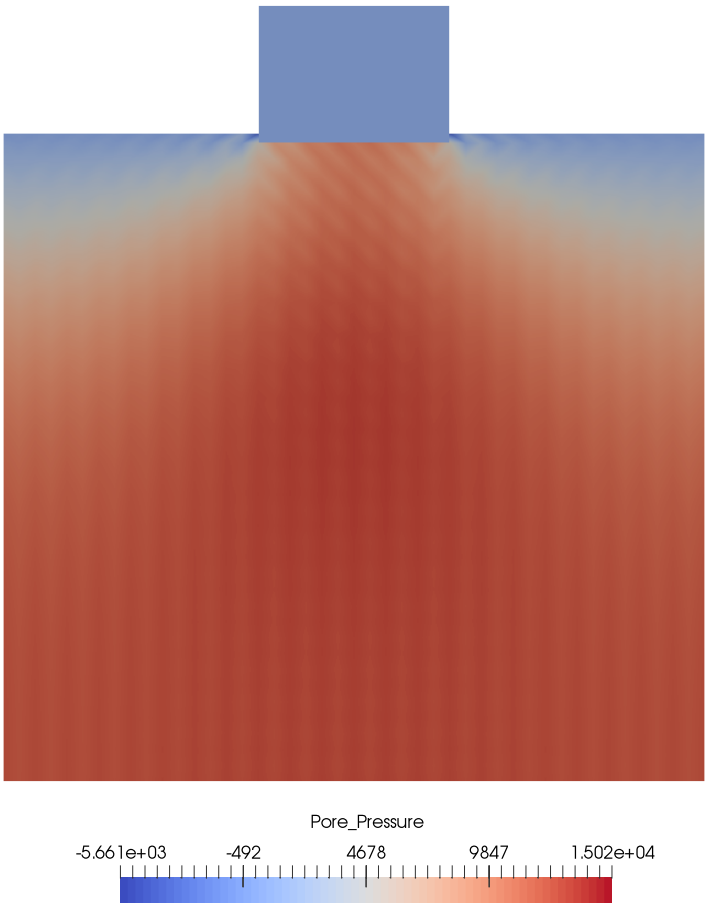
2. Follow the steps as shown in Figure 208.23



(a) Enable upU Mode Visualization Option



(b) Select the visualization variable either pore pressure or fluid displacement



(c) Visualization of Pore Pressure p at nodes

Figure 208.23: Steps to visualize pore pressure p or fluid displacements U in upU visualization mode.

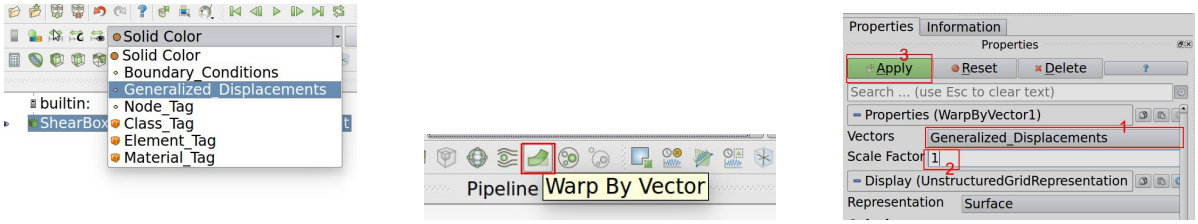
208.4.1.11 Eigen Mode Visualization

Visualization of eigen modes is that same as visualizing "displacements" and applying "warp by vector" filter on *Eigen Value Analysis* output of Real-ESSI simulation.

1. Open an *eigen value analysis* output. All the example files can be obtained at http://sokocalo.engr.ucdavis.edu/~jeremic/lecture_notes_online_material/Real-ESSI_pvESSI/Examples.

```
cd pvESSI/Examples
ParaView ShearBoxWall_Eigen_Analysis.h5.feioutput
```

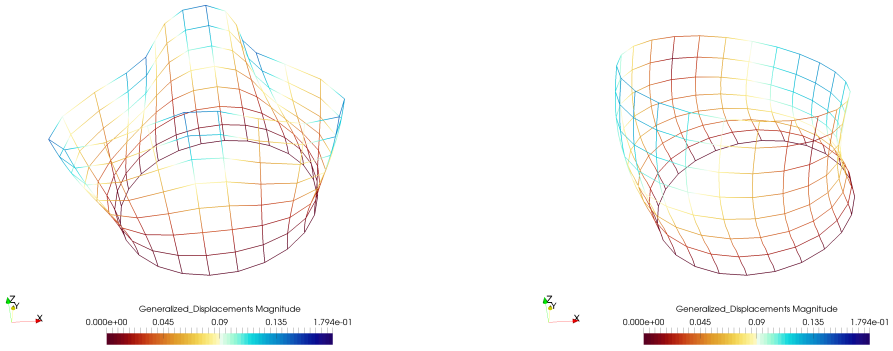
2. Select displacement field and then apply warp by vector plugin and selected its properties



(a) Select Displacement Field (b) Select plugin (c) Plugin properties

Figure 208.24: Eigen Modes visualization.

3. Now n^{th} time steps here, corresponds to the n^{th} eigen mode.



(a) Eigen Mode 5 (b) Eigen Mode 9

Figure 208.25: Few eigen modes.

208.4.1.12 Visualizing Physical Node and Element groups

In Real-ESSI it is possible to define different physical groups, for nodes and for elements. If one has defined physical groups in Real-ESSI, you can visualize the same in ParaView. There are two sections here that shows all the physical groups (nodes and elements) defined in the model as shown in Figure 208.26. Section 205.3.4.36 and Section 205.3.4.43 shows how to define and add physical group of nodes and elements respectively in Real-ESSI. All the example files can be obtained at http://sokocalo.engr.ucdavis.edu/~jeremic/lecture_notes_online_material/Real-ESSI_pvESSI/Examples.

```
cd pvESSI/Examples
ParaView Model_With_Physical_Groups.h5.feioutput
```

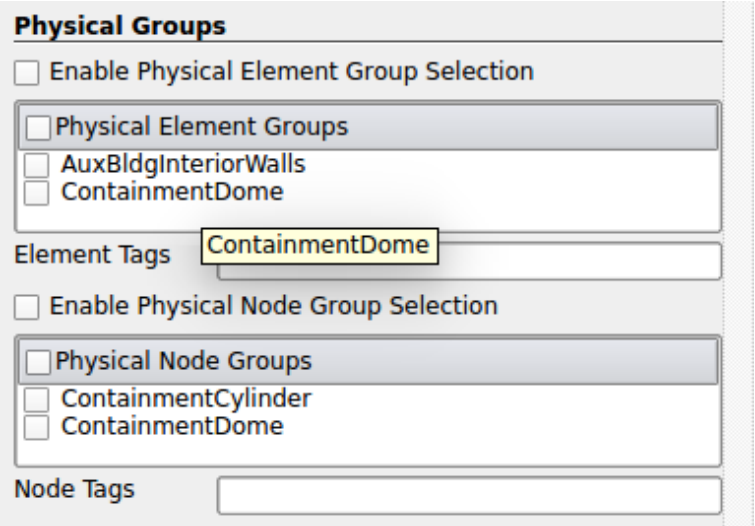


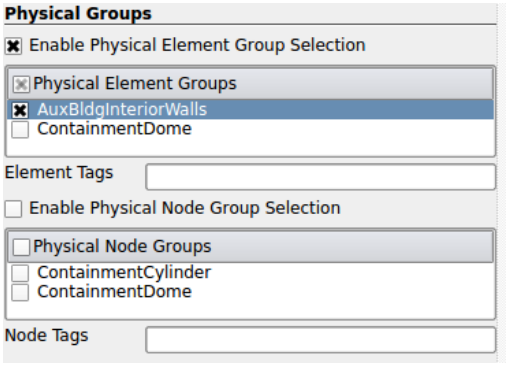
Figure 208.26: Physical Group Visualization Options.

This feature is very useful, when one is interested only in some specific regions of the model than the whole model. Also, This feature becomes very useful, for complicated "interested region/parts of the mesh", which cannot be selected by usual clip/box/..etc filters

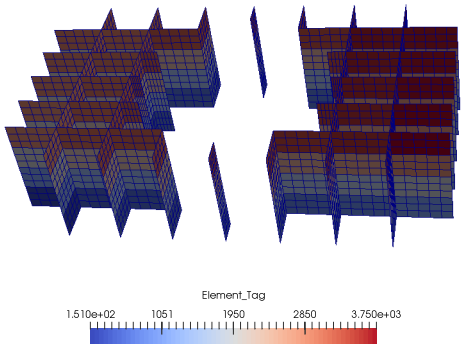
1. Enable Physical Element Group Selection - Enables the selection of Physical Element Group. By default, it is disabled and one would see the whole mesh. By enabling it, one would only see the selected 'Physical Element Groups'.

2. Physical Element Groups - It shows all the physical element groups defined in Input file of Real-ESSI. The user can select (one or more) of physical groups and hit apply to visualize them. It would show any effect only if the above options Enable Physical Element Group Selection is checked. Figure 208.27 shows steps to visualize the physical element groups defined in input files using

Real-ESSI DSL.



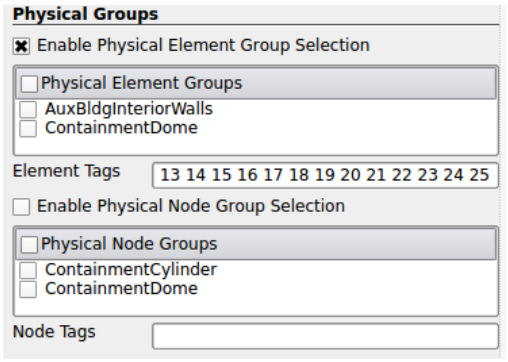
(a) Select 'Physical Element Group Selection' Option



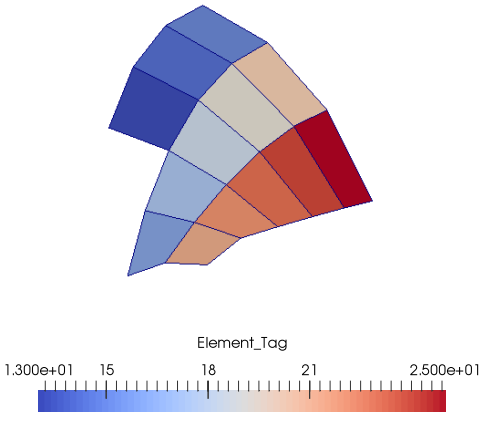
(b) ElementGroup : Auxiliary Building Interior Walls

Figure 208.27: Visualization of physical groups predefined in input file using Real-ESSI DSL.

3. Element Tags - This options provides user and interface to manually write element tags to be visualized. The user should enter the element tags against this option as a integer list separated by space. For example:- '2 10 12 13 16', where each of the number corresponds to the element tag defined in the model. Again, this option would only work if Enable Physical Element Group Selection option is checked. Figure 208.28 shows steps to visualize elements defined manually.



(a) Select 'Physical Element Group Selection' Option

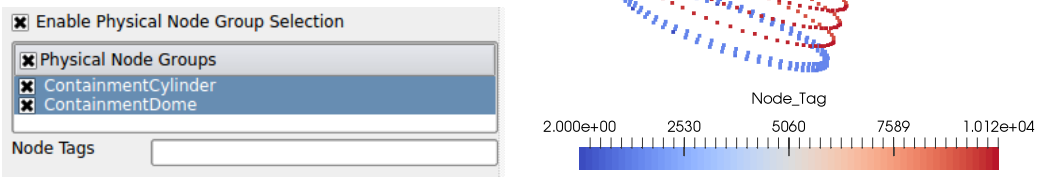


(b) ElementGroup : Manually Defined Element Tags

Figure 208.28: Visualization of physical element group manually defined using PVESSIReader option.

4. Enable Physical Node Group Selection - Enables the selection of Physical Node Group. By default, it is disabled and you would see the whole mesh. By enabling it you would only see the selected Physical Node Groups.

5. Physical Node Groups - It shows all the physical node groups defined in Input file of Real-ESSI. The user can select (one or more) of physical groups and hit apply to visualize them.It would show any effect only if the above options Enable Physical Node Group Selection is checked. Figure 208.29 shows steps to visualize the physical element groups defined in input files using Real-ESSI DSL.

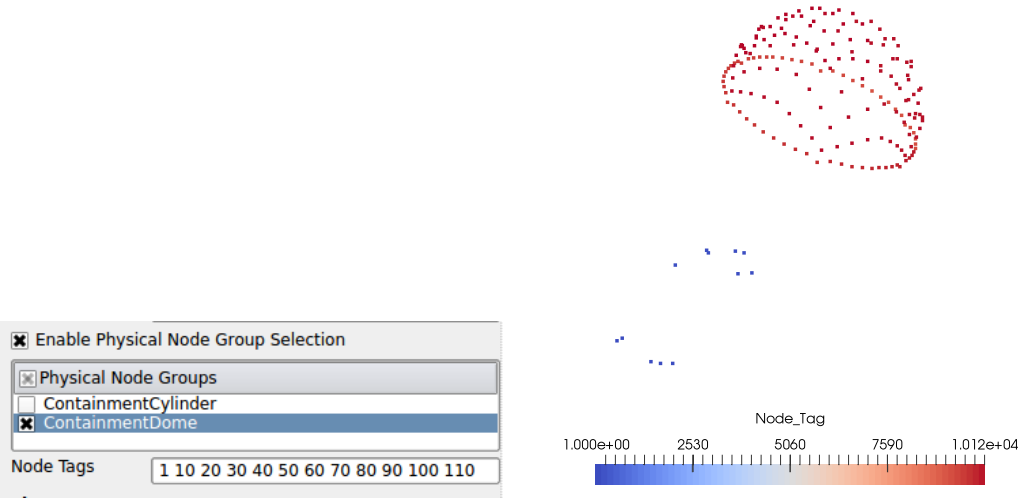


(a) Select 'Physical Node Group Selection' Option (b) NodeGroup : Auxiliary Building Interior Walls

Figure 208.29: Visualization of physical groups predefined in input file using Real-ESSI DSL.

6. Node Tags - This options provides user and interface to manually write node tags to be visualized. The user should enter the node tags against this option as a integer list separated by space. For example:- '2 10 12 13 16', where each of the number corresponds to the node tag defined in the model. Again, this option would only work if Enable Physical Node Group Selection option is checked. Figure 208.30 shows steps to visualize nodes defined manually.

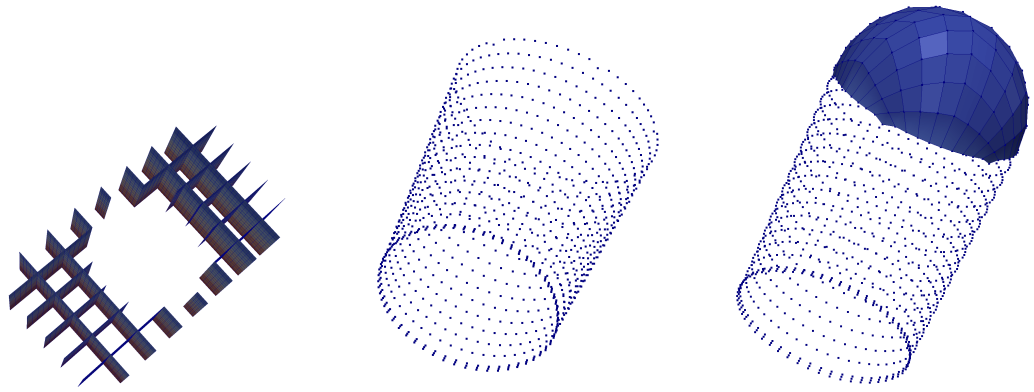
NOTE: The user can also select both at once, i.e physical element group and physical node group, from the above menu. Figure 208.31 shows mixed selection.



(a) Select 'Physical Node Group Selection' Option

(b) NodeGroup : Manually Defined Node Tags

Figure 208.30: Visualization of physical node group manually defined using PVESSIReader option.



(a) ElementGroup : Auxiliary Building Interior Walls

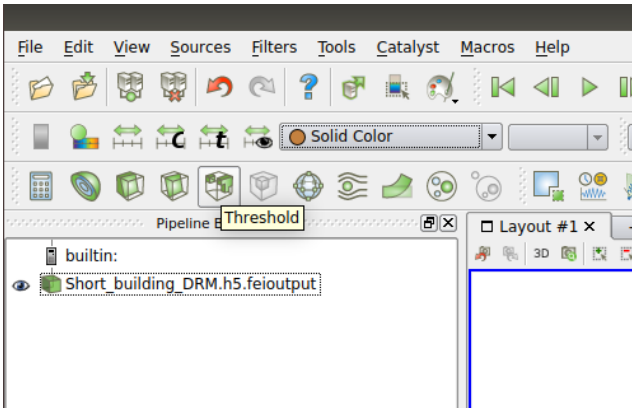
(b) Node Group : Containment Cylinder

(c) Mixed Group : Containment Cylinder with Dome

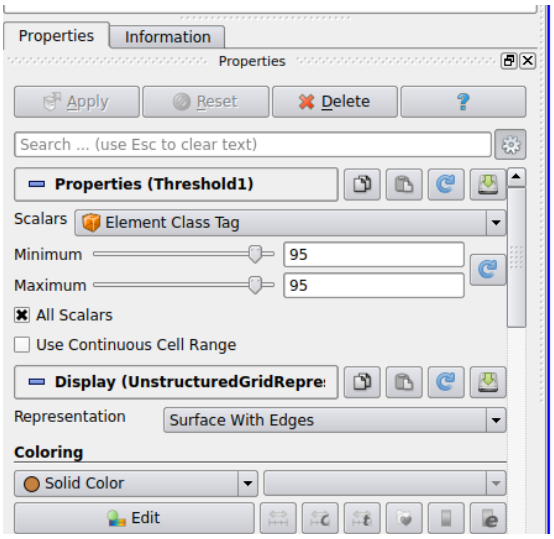
Figure 208.31: Visualization of physical groups.

208.4.1.13 Using Threshold to Visualize Certain Elements

ParaView allows user to choose specific element types and only visualize selected elements. This function is achieved using Threshold. As shown in Figure 208.32, first click on the Threshold button in toolbar. Then, choose Element Class Tag in the drop-down list of Scalars, which can be found in Properties. A certain range of Element Class Tag can be chosen by setting the minimum and maximum values. If the minimum and maximum values are the same, only one element type will be selected and visualized.



(a) Click on Threshold



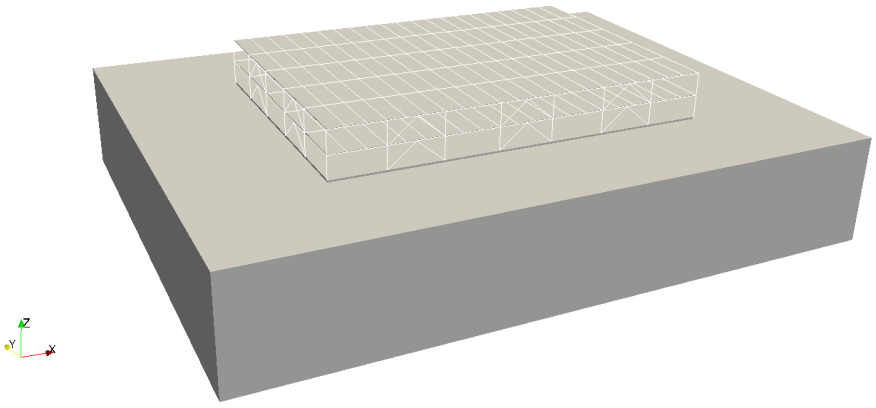
(b) Choose Element Class Tag

Figure 208.32: Using Threshold to Visualize Certain Elements.

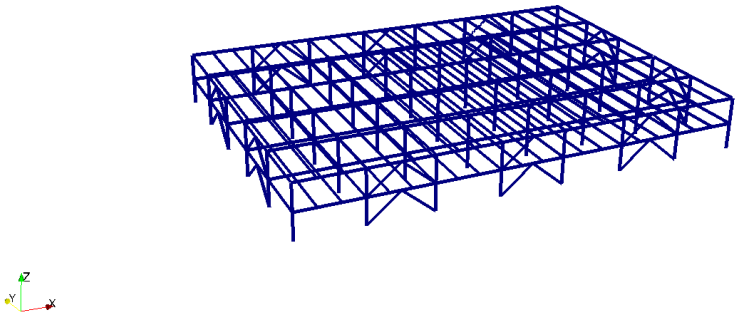
A list of available element class tags in Real-ESSI is provided in Table 208.1.

Table 208.1: Available element class tags in Real-ESSI.

Finite Element Name	Element Class Tag
Truss Element	88
Shear Beam Element	93
Elastic Beam–Column Element	89
Timoshenko Elastic Beam–Column Element	129
Elastic Beam-Column Element with Lumped Mass	90
3D Displacement Based Fiber Beam-Column Element	95
4 Node ANDES Shell with Drilling DOFs	92
3 Node ANDES Shell with Drilling DOFs	91
Super Element Linear Elastic Import	9904
8 Node Brick Element (Order One, Two, Three, Four, Five, Six)	2 (14, 26, 38, 50, 62, 74)
8 Node Brick u-p Element (Order One, Two, Three, Four, Five, Six)	3 (15, 27, 39, 51, 63, 75)
8 Node Brick u-p-U Element (Order One, Two, Three, Four, Five, Six)	4 (16, 28, 40, 52, 64, 76)
20 Node Brick Element (Order One, Two, Three, Four, Five, Six)	5 (17, 29, 41, 53, 65, 77)
20 Node Brick u-p Element (Order One, Two, Three, Four, Five, Six)	6 (18, 30, 42, 54, 66, 78)
20 Node Brick u-p-U Element (Order One, Two, Three, Four, Five, Six)	7 (19, 31, 43, 55, 67, 79)
27 Node Brick Element (Order One, Two, Three, Four, Five, Six)	8 (20, 32, 44, 56, 68, 80)
27 Node Brick u-p Element (Order One, Two, Three, Four, Five, Six)	9 (21, 33, 45, 57, 69, 81)
27 Node Brick u-p-U Element (Order One, Two, Three, Four, Five, Six)	10 (22, 34, 46, 58, 70, 82)
Variable Node Brick Element (Order One, Two, Three, Four, Five, Six)	11 (23, 35, 47, 59, 71, 83)
Variable Node Brick u-p Element (Order One, Two, Three, Four, Five, Six)	12 (24, 36, 48, 60, 72, 84)
Variable Node Brick u-p-U Element (Order One, Two, Three, Four, Five, Six)	13 (25, 37, 49, 61, 73, 85)
8 Node Cosserat Brick Element	96
Bonded Contact/Interface/Joint Element	102
Force Based Dry Hard Contact/Interface/Joint Element	86
Force Based Dry Soft Contact/Interface/Joint Element	87
Force Based Coupled Hard Contact/Interface/Joint Element	97
Force Based Coupled Soft Contact/Interface/Joint Element	98
Stress Based Dry Hard Contact/Interface/Joint Element with Elastic Perfectly Plastic Shear Behavior	99
Stress Based Dry Hard Contact/Interface/Joint Element with Nonlinear Hardening Shear Behavior	100
Stress Based Dry Hard Contact/Interface/Joint Element with Nonlinear Hardening and Softening Shear Behavior	101
Stress Based Dry Soft Contact/Interface/Joint Element with Elastic Perfectly Plastic Shear Behavior	107
Stress Based Dry Soft Contact/Interface/Joint Element with Nonlinear Hardening Shear Behavior	108
Stress Based Dry Soft Contact/Interface/Joint Element with Nonlinear Hardening and Softening Shear Behavior	109
Stress Based Coupled Hard Contact/Interface/Joint Element with Elastic Perfectly Plastic Shear Behavior	104
Stress Based Coupled Hard Contact/Interface/Joint Element with Nonlinear Hardening Shear Behavior	105
Stress Based Coupled Hard Contact/Interface/Joint Element with Nonlinear Hardening and Softening Shear Behavior	106
Stress Based Coupled Soft Contact/Interface/Joint Element with Elastic Perfectly Plastic Shear Behavior	110
Stress Based Coupled Soft Contact/Interface/Joint Element with Nonlinear Hardening Shear Behavior	111
Stress Based Coupled Soft Contact/Interface/Joint Element with Nonlinear Hardening and Softening Shear Behavior	112



(a) Full Model



(b) Only Fiber Beam-Column Elements

Figure 208.33: An example of using threshold to visualize fiber beam-column elements in a soil-structure model.

Chapter 209

Software Platform Build Process

(1993-1994-1996-1999-2003-2005-2007-2008-2009-2010-2011-2015-2017-2018-2019-2020-2021-)

(In collaboration with Prof. José Abell, Dr. Yuan Feng, Dr. Hexiang Wang, and Prof. Han Yang)

209.1 Chapter Summary and Highlights

This Chapter gives a brief description of build and installation procedures for the Real-ESSI Simulator. It is noted that current installation procedures rely on Real-ESSI Simulator program Debian package distribution, with pre and post processing modules that are installed separately. Both sequential and parallel version of the Real-ESSI Simulator program Debian packages are distributed, for Linux/Ubuntu operating system.

209.2 Introduction to the Real-ESSI Simulator Program

The Real-ESSI Simulator systems consists of the Real-ESSI Program, Real-ESSI Computer and Real-ESSI Notes. Alternative name for the Real-ESSI Simulator system is Real-ESSI Simulator system. The name Real-ESSI, is explained in section 201.2.6 on page 710.

209.3 Real-ESSI Simulator System Install

In adition to the Real-ESSI Program, Real-ESSI Simulator system consists of a pre-processing modules and post-processing modules. Installation of pre-processing modules is described in Chapter 207, on page 1221 of the main document, lecture notes (Jeremić et al., 1989-2025). Installation of post-processing modules is described in Chapter 208, on page 1287 of the main document, lecture notes (Jeremić et al., 1989-2025).

Both pre and post processing manuals are also available through the main Real-ESSI Simulator web site: http://real-essi.info/.

209.4 Build Procedures for the Real-ESSI Program and Modules

Note: This section describes build procedure for the Global Release 25.04 version of Real-ESSI. The very same procedures will apply to future version...

These build procedures are meant for users that have access to Real-ESSI Program source code. Required operating system is Ubuntu 24.04 LTS, unless otherwise specified. Building Real-ESSI on older versions of Ubuntu is no longer supported. Users that do not have source code can install Real-ESSI form Debian package, that is also available.

209.4.1 System Libraries Update/Upgrade

```
sudo apt update
sudo apt upgrade
sudo apt dist-upgrade
sudo apt autoremove
```

209.4.2 Install Build Dependencies

```
sudo apt install -y bison
sudo apt install -y build-essential
sudo apt install -y cmake
sudo apt install -y flex
sudo apt install -y git
sudo apt install -y mpich
sudo apt install -y ssh
sudo apt install -y valgrind
sudo apt install -y wget
sudo apt install -y zlib1g-dev
sudo apt install -y libboost-all-dev
sudo apt install -y libgmp3-dev
sudo apt install -y libhdf5-serial-dev
sudo apt install -y liblapack-dev
sudo apt install -y libmpfr-dev
sudo apt install -y libopenblas-dev
sudo apt install -y libopenmpi-dev
sudo apt install -y libpthread-workqueue-dev
sudo apt install -y libssl-dev
sudo apt install -y libtbb-dev
```

You may need to manually link the HDF5 libs to their proper names so that the compiler can find them. The HDF5 maybe in different versions. NOTE: what is needed is a latest version of libhdf5 for serial execution, the one "cpp" and the one without "cpp", so search for it by doing:

```
cd /usr/lib/x86_64-linux-gnu
dir `find . -name "*hdf5*serial*cpp*"`
```

and

```
dir `find . -name "*hdf5*serial*"` | grep -v cpp
```

Then the linking has to be for both versions, for example:

```
cd /usr/lib/x86_64-linux-gnu
sudo ln -s libhdf5_serial.so.103.3.0 libhdf5.so
sudo ln -s libhdf5_serial_cpp.so.103.3.0 libhdf5_cpp.so
```

If the libs `libhdf5.so` and `libhdf5_cpp.so` are already there, just move on.

209.4.3 Download Real-ESSI Source

It is important to note that Real-ESSI sources are not available for public download. This is so that we can control and guarantee Real-ESSI quality. Only developer collaborators are contributing sources, and those sources are quality checked and quality assured. In addition, there are a number of unique solutions, unique formulations, unique implementation details within Real-ESSI sources that are not available in any other research of commercial program. If you happen to obtain Real-ESSI sources from Prof. Jeremić, you can proceed with the installation procedure below.

Please make sure that you are in the main directory where your Real-ESSI global release is placed ((GLOBAL_RELEASE). Make a directory where all the sources will reside and go there:

```
cd
mkdir Real-ESSI
cd Real-ESSI
```

Obtain Real-ESSI sources from the github:

```
git clone git@github.com:BorisJeremic/Real-ESSI.git
```

Go to the Real-ESSI source directory:

```
cd Real-ESSI
```

Remember to 'git checkout' to the proper branch.

209.4.4 Download and Compile Real-ESSI Dependencies

Make directories for the dependencies:

```
mkdir -p ../RealESSI_Dependencies
mkdir -p ../RealESSI_Dependencies/include
mkdir -p ../RealESSI_Dependencies/lib
mkdir -p ../RealESSI_Dependencies/bin
mkdir -p ../RealESSI_Dependencies/SRC
```

Go to the directory, download and extract the sources of the dependencies:

```
cd ../RealESSI_Dependencies
wget http://sokocalo.engr.ucdavis.edu/~jeremic/RealESSI/Dependencies_SRC.tar.gz
tar -xzvf ./Dependencies_SRC.tar.gz -C ./SRC --strip-components 1
```

Go to the Real-ESSI directory and compile the dependencies:

```
cd ../Real-ESSI
./build_libraries suitesparse
./build_libraries arpack
```

```
./build_libraries lapack
./build_libraries parmetis
./build_libraries petsc_itself
```

209.4.5 Configure, Build, and Install the Real-ESSI Program

Configure and build the sequential version of Real-ESSI:

```
mkdir build
cd build
cmake ..
make -j 8
cd ..
```

Configure and build the parallel version of Real-ESSI:

```
mkdir pbuild
cd pbuild
cmake -DCMAKE_CXX_COMPILER=/usr/bin/mpic++ -DPROGRAMMING_MODE=PARALLEL ..
make -j 16
cd ..
```

Copy the Real-ESSI executables to system directory:

```
sudo cp build/essi /usr/local/bin/essi-sequential
sudo cp pbuild/essi /usr/local/bin/essi-parallel
```

209.4.6 Install Sublime Text and Real-ESSI Packages

Sublime Text (https://www.sublimetext.com/) is the recommended editor for Real-ESSI input files and pre-processing files. Install Sublime Text following the official installation steps, or using the following command:

```
wget -qO - https://download.sublimetext.com/sublimehq-pub.gpg | gpg --dearmor | ↩
    sudo tee /etc/apt/trusted.gpg.d/sublimehq-archive.gpg
echo "deb https://download.sublimetext.com/ apt/stable/" | sudo tee ↩
    /etc/apt/sources.list.d/sublime-text.list
sudo apt-get update
sudo apt-get install sublime-text
```

Open Sublime Text. Open the 'Tools' menu and select 'Install Package Control...'. Open the 'Preferences' menu, select 'Package Control', then select 'Package Control: Install Package'.

In the opened search bar, type the package name and click on the package to install it. Three packages should be installed:

FEI Syntax-n-Snippets, Real-ESSI syntax and auto completion plugin for [].fei files (input files for Real-ESSI program).

gmsh-Tools, syntax and autotext completion for Gmsh model development tools for Real-ESSI.

gmESSI-Tools, syntax and autotext completion for gmESSI model development tools for Real-ESSI.

209.4.7 Install HDFView

`HDFView` can be used to open Real-ESSI output files, which are in HDF5 format. Download the latest version of `HDFView` from https://support.hdfgroup.org/ftp/HDF5/releases/HDF-JAVA/. Click on the latest version, which is `hdfview-3.2.0` as of June 2022. Go to `bin/`, click `HDFView-3.2.0-ubuntu2004_64.tar.gz` and save the file in your `./Downloads/` directory. Then extract and install HDFView:

```
cd
tar -xvf ./Downloads/HDFView-3.2.0-ubuntu2004_64.tar.gz -C ./Downloads
sudo apt install -y ./Downloads/hdfview_3.2.0-1_amd64.deb
sudo ln -s /opt/hdfview/bin/HDFView /usr/local/bin/hdfview
```

Now you can use `HDFView` from a terminal. To be able to use `HDFView` when you click on a Real-ESSI output file, do the following additional steps. First open the file using the following command:

```
sudo gedit /usr/share/applications/hdfview-HDFView.desktop
```

Find the line:

```
Exec=/opt/hdfview/bin/HDFView
```

Replace it with:

```
Exec=/opt/hdfview/bin/HDFView %F
```

Save the file and close it.

Go to a Real-ESSI output file, which should have the suffix 'h5.feioutput'. Right click on the file and select 'Open with Other Application'. Click 'View All Applications' and choose `HDFView` from the list. Note that you only need to do this once. Next time when you click on a Real-ESSI output file, it will be opened automatically using `HDFView`.

209.4.8 Compile ParaView and PVESSIReader for Post-Processing

Install the build dependencies for ParaView:

```
sudo apt install -y libgl1-mesa-dev
sudo apt install -y libxt-dev
```

```
sudo apt install -y libqt5x11extras5-dev
sudo apt install -y libqt5help5
sudo apt install -y qttools5-dev
sudo apt install -y qtxmlpatterns5-dev-tools
sudo apt install -y libqt5svg5-dev
sudo apt install -y libtbb-dev
sudo apt install -y python3-dev
sudo apt install -y python3-numpy
sudo apt install -y ninja-build
```

Go to the directory where you want to install ParaView. Suggested location is the parent directory of the Real-ESSI source. If you are continuing from the previous subsection, do the following:

```
cd ..
```

Download the ParaView source from GitHub:

```
git clone --recursive https://gitlab.kitware.com/paraview/paraview.git
```

Make the build directory:

```
cd paraview
mkdir paraview_build
```

Modify the cmake file to include PVESSIReader plugin in the building process. Open the file `CMakeLists.txt` in the ParaView source directory. Find "set(paraview_default_plugins" and add "PVESSIReader" to the end of the list of plugins. Download the PVESSIReader source from GitHub:

```
cd Plugins
git clone git@github.com:BorisJeremic/Real-ESSI-pvESSI.git
mv Real-ESSI-pvESSI PVESSIReader
```

Go to the build directory and compile ParaView:

```
cmake -GNinja -DPARAVIEW_USE_PYTHON=ON -DPARAVIEW_USE_MPI=ON ↩
    -DVTK_SMP_IMPLEMENTATION_TYPE=TBB -DCMAKE_BUILD_TYPE=Release ..
ninja
```

Copy the ParaView executable to system directory:

```
sudo cp bin/paraview /usr/local/bin/paraview
```

Start `ParaView` and click 'Tools' → 'Manage Plugins...'. Click 'Load New...' and find the plugin named 'PVESSIReader.so' under directory `paraview/lib/paraview-5.10/plugins/PVESSIReader/`. Also check the box 'Auto Load' then close `ParaView`.

The procedures described in this subsection are based on the official build instruction of ParaView

(https://gitlab.kitware.com/paraview/paraview/blob/master/Documentation/dev/build.md).

209.5 Build Real-ESSI Debian Package

Note: This section describes Debian package build procedure for the Global Release 22.07 and later versions of Real-ESSI. As noted before, the very same procedures will apply to future Ubuntu versions ...

Starting from the Global Release 22.07 version, the Real-ESSI Simulator system is distributed as a Debian package for Linux users. This section documents the build procedure of a Real-ESSI Debian package. Note that the steps described here are for building a "basic" or "quick" stand-alone Debian package containing the already-compiled Real-ESSI program, pre-processor gmsh/gmESSI and results viewer Paraview/pvESSI modules. This is different from building a Debian package containing the program sources. For more information see:

- https://wiki.debian.org/Packaging
- https://www.internalpointers.com/post/build-binary-deb-package-practical-guide
- https://ubuntuforums.org/showthread.php?t=910717

209.5.1 Build the Real-ESSI Program and Modules

Before starting to build the Debian package, you should have finalized building the Real-ESSI program and modules from source. To build Real-ESSI from source, please follow the build procedure described in section 209.4.

209.5.2 Build the Debian Package

209.5.2.1 Package Name

Standard Debian notation is all lowercase in the following format:

```
<project>_<major version>.<minor version>-<package revision>
```

The current Real-ESSI Debian package has the name:

```
real-essi_25.04_amd64
```

Note that the version names will change and be consistent with the version of Real-ESSI program, as decribed at http://real-essi.info/, so that users will have to change the above name to reflect the actual Real-ESSI version and the Debian package.

209.5.2.2 Create Directory

Create a directory to make your package in. The name should be the same as the package name.

```
mkdir real-essi_25.04_amd64
```

209.5.2.3 Create Internal Structure

Good idea is to put packaging directory in the root of Real-ESSI system. So go to where all of this happening, for example:

```
cd /home/jeremic/oofep/Rad_na_Sokocalu/GLOBAL_RELEASE/Real-ESSI
```

Make space for files of your program where they would be installed on a linux system.

```
mkdir real-essi_25.04_amd64/usr
mkdir real-essi_25.04_amd64/usr/local
mkdir real-essi_25.04_amd64/usr/local/bin
mkdir real-essi_25.04_amd64/usr/lib
mkdir real-essi_25.04_amd64/usr/lib/x86_64-linux-gnu
mkdir real-essi_25.04_amd64/opt
mkdir real-essi_25.04_amd64/opt/gmESSI
mkdir real-essi_25.04_amd64/opt/paraview
```

209.5.2.4 Copy Files

Copy the files to the packaging directory. Note that you should use your own directory paths....

For example:

```
cp bin/essi.sequential real-essi_25.04_amd64/usr/local/bin/essi.sequential
cp bin/essi.parallel real-essi_25.04_amd64/usr/local/bin/essi.parallel
```

209.5.2.5 Create the `control` File

Create a special metadata file that is used by the package manager to install program. The control file lives inside the `DEBIAN` directory. Mind the uppercase: a similar directory named `debian` (lowecase) is used to store source code for the so-called source packages. This tutorial is about binary packages,so we don't need source code. Create the empty `control` file:

```
mkdir real-essi_25.04_amd64/DEBIAN
touch real-essi_25.04_amd64/DEBIAN/control
```

Open the file previously created with text editor of your choice. The `control` file is just a list of data fields, as seen in listing below:

```
Package: real-essi
Version: 25.04
Architecture: amd64
Authors: Han Yang <hhhyang@ucdavis.edu>, Boris Jeremic <jeremic@ucdavis.edu>
Maintainer: Han Yang <hhhyang@ucdavis.edu>, Boris Jeremic <jeremic@ucdavis.edu>
Depends: libboost-all-dev, libhdf5-dev, libtbb-dev, libssl-dev, libopenmpi-dev, ↩
    mpich, libgl1-mesa-dev, libxt-dev, libqt5x11extras5-dev, libqt5help5, ↩
    qttools5-dev, qtxmlpatterns5-dev-tools, libqt5svg5-dev, libtbb-dev, ↩
    python3-dev, python3-scipy, python3-numpy, python3-matplotlib, python3-pip, ↩
    python3-pygments, liboctave-dev, python2.7-dev
Section: misc
Priority: optional
Provides: real-essi
Description: The Real-ESSI Simulator.
 The Real-ESSI Simulator (Realistic Modeling and Simulation
 of Earthquakes, and/or Soils, and/or Structures and their
 Interaction) is a software, hardware and documentation
 system for high performance, sequential or parallel, time
 domain, linear or nonlinear, elastic and inelastic,
 deterministic or probabilistic, finite element modeling and
 simulation of
  - statics and dynamics of soil,
  - statics and dynamics of rock,
  - statics and dynamics of structures,
  - statics of and dynamics of soil-structure systems,
  - dynamics of earthquakes, and
  - dynamic earthquake-soil-structure interaction.
Homepage: http://real-essi.info
```

209.5.2.6 Create the Post-Installation and Post-Remove Files

```
touch real-essi_25.04_amd64/DEBIAN/postinst
touch real-essi_25.04_amd64/DEBIAN/postrm
```

Both of these files have to have the following permissions::

```
chmod u+rwx real-essi_25.04_amd64/DEBIAN/postinst
chmod u+rwx real-essi_25.04_amd64/DEBIAN/postrm

chmod og=rx real-essi_25.04_amd64/DEBIAN/postinst
chmod og=rx real-essi_25.04_amd64/DEBIAN/postrm
```

The `postinst` file/script is executed after a successful installation of the Debian package. This script looks like this:

```
#!/bin/sh
# postinst script for real-essi

set -e

case "$1" in
    configure)
# not used
# ln -s /opt/gmESSI/build/bin/gmessy /usr/local/bin/
# ln -s /opt/paraview/bin/paraview /usr/local/bin/

update-alternatives --install /usr/bin/python python /usr/bin/python2.7 1
pip install h5py
    ;;

    abort-upgrade|abort-remove|abort-deconfigure)
    ;;

    *)
        echo "postinst called with unknown argument \`$1'" >&2
        exit 1
    ;;
esac

exit 0
```

The postrm file/script is executed after a successful removal of the Debian package. This script looks like this:

```
#!/bin/sh
# postrm script for real-essi

set -e

case "$1" in
    purge|remove|upgrade|failed-upgrade|abort-install|abort-upgrade|disappear)
# not used
# rm /usr/local/bin/gmessy
# rm /usr/local/bin/paraview
    ;;

    *)
        echo "postrm called with unknown argument \`$1'" >&2
        exit 1
    ;;
esac

exit 0
```

209.5.2.7 Build the Package

Build the package:

```
dpkg-deb --build --root-owner-group real-essi_25.04_amd64
```

The `--root-owner-group` flag makes all deb package content owned by the root user, which is the standard way to go. Without such flag, all files and folders would be owned by your user, which might not exist in the system the deb package would be installed to.

The command above will generate a nice `.deb` file alongside the working directory or print an error if something is wrong or missing inside the package. If the operation is successful you have created debian package ready for distribution.

209.6 Real-ESSI and OpenFOAM, Connecting

Note: This section describes build procedure for old versions of Real-ESSI and/or its modules.

OpenFOAM is a free, open source computational fluid dynamics (CFD) software developed primarily by OpenCFD Ltd since 2004 (https://www.openfoam.com/). Real-ESSI supports numerical interface with OpenFOAM and can perform solid/structure fluid interaction analysis through Real-ESSI – OpenFOAM connection.

209.6.1 Installation of Customized OpenFOAM

We have made in-house modifications and developments to the InterFOAM application (Deshpande et al., 2012) of OpenFOAM-v1612+ for solid fluid interaction. This section presents the installation of the Customized OpenFOAM:

Install the dependencies:

```
sudo apt-get update
sudo apt-get install build-essential
sudo apt-get install flex
sudo apt-get install bison
sudo apt-get install cmake
sudo apt-get install zlib1g-dev
sudo apt-get install libboost-system-dev
sudo apt-get install libboost-thread-dev
sudo apt-get install libopenmpi-dev
sudo apt-get install openmpi-bin
sudo apt-get install gnuplot
sudo apt-get install libreadline-dev
sudo apt-get install libncurses-dev
sudo apt-get install libxt-dev
```

```
sudo apt-get install qt4-dev-tools
sudo apt-get install libqt4-dev
sudo apt-get install libqt4-opengl-dev
sudo apt-get install freeglut3-dev
sudo apt-get install libqtwebkit-dev
sudo apt-get install libscotch-dev
sudo apt-get install libcgal-dev
```

Also, make sure gcc and cmake meet the following minimum version requirements:

- gcc: version 4.8.5 or above
- cmake: version 3.3 or above

Check the version of gcc and cmake by running the following commands on terminal. If you are installing on Ubuntu 16.04 and above, the system version of gcc and cmake should already meet the requirements.

```
gcc --version
cmake --version
```

Downloaded the source code of Customized OpenFOAM:

```
wget http://sokocalo.engr.ucdavis.edu/~jeremic/lecture_notes_online_material/
_Chapter_SoftwareHardware_Build_Process/OpenFOAM/sources/OpenFOAM.tar.gz
```

Choose a directory and extract the downloaded compressed file to the target directory.

```
tar -xzvf OpenFOAM.tar.gz -C /target/directory
```

For example, hereafter we choose $HOME as target directory. Replace $HOME with your chosen directory accordingly.

```
tar -xzvf OpenFOAM.tar.gz -C $HOME
```

Go to the extracted folder and source OpenFOAM environment configurations:

```
cd $HOME/OpenFOAM
source $HOME/OpenFOAM/OpenFOAM-v1612+/etc/bashrc
```

Setup CGAL and Boost version for compilation:

```
cgal_version=CGAL-4.9.1
boost_version=boost-system
```

Check the system readiness

```
foamSystemCheck
```

Change to the main OpenFOAM directory:

```
foam
```

Note: if running *foam* cannot change to the main OpenFOAM directory, in this case the directory is *$HOME/OpenFOAM*, source the environment configuration again by running the following terminal command.

```
source $HOME/OpenFOAM/OpenFOAM-v1612+/etc/bashrc
```

Compile OpenFOAM:

```
./Allwmake
```

Since OpenFOAM is shipped with ParaView for post-processing OpenFOAM field results using developed plug-in paraFoam (https://cfd.direct/openfoam/user-guide/v6-paraview/). We also need to compile customized ParaView with paraFoam plug-in:

```
cd $WM_THIRD_PARTY_DIR
./makeParaView
```

209.6.2 Check the Customized OpenFOAM Installation

Open a new terminal and source the OpenFOAM environment:

```
source $HOME/OpenFOAM/OpenFOAM-v1612+/etc/bashrc
```

Validate the build by running:

```
foamInstallationTest
```

Create a user run directory:

```
mkdir -p $FOAM_RUN
```

go to the user run directory:

```
run
```

Copy a simulation case from OpenFOAM tutorial to the user run directory:

```
cp -r $FOAM_TUTORIALS/incompressible/simpleFoam/pitzDaily ./
```

go to the copies case directory:

```
cd pitzDaily
```

Generate the mesh:

```
blockMesh
```

Perform the analysis with the application simpleFoam:

```
simpleFoam
```

Visualize the simulation results:

```
paraFoam
```

209.6.3 Compile Real-ESSI with Link to OpenFOAM

Go to Real-ESSI source directory under directory RealESSI_ROOT and clean any previous old compilation of Real-ESSI:

```
cd RealESSI_ROOT/Real-ESSI
rm -rf bin
rm -rf lib
rm -rf build_sequential
mkdir bin
mkdir lib
mkdir build_sequential
cd build_sequential
```

Build and install the executable, using 16 CPUs in this case. Of course, if you have more CPUs available, you can use most of them. Please make sure to specify your OpenFOAM installation directory with CMake argument -DOPENFOAM_DIR. For example, in this case, we specify the installation directory as *$HOME/OpenFOAM*.

```
time cmake -DUSE_OPENFOAM=TRUE -DOPENFOAM_DIR=$HOME/OpenFOAM ..
time make -j 16
make install
```

Rename essi to essi.sequential just so to distinguish it from the parallel executable:

```
cd ../bin
cp essi essi.sequential
```

Finally, install essi.sequential in system binary directory so that others can use it:

```
sudo rm /usr/bin/essi /usr/bin/essi.sequential
sudo cp essi.sequential /usr/bin/essi.sequential
sudo chmod a+x /usr/bin/essi.sequential
```

209.7 Code Verification After the Build Process

After build process, test cases to verify that installation is successful should be run. There are four groups of verification cases. The first two groups are designed for users. The last two groups are designed for developers.

1. The first group of test cases compares the sequential essi results to the analytic solutions.
2. The second group of test cases compares the parallel essi results to the analytic solutions.
3. The third group of test cases tests the version stability between two essi executables.
4. The fourth group of test cases tests the memory management of Real-ESSI with valgrind.

209.7.1 Run all verification test cases

In order to run all test cases to verify the installation, users can run

```
cd $RealESSI_PATH/
bash run_all_verification.sh
```

Please make sure that sequential essi is available as 'essi' in the PATH, and parallel essi is available as 'essi_parallel' in the PATH before running all the verification test cases.

In addition, if users want to clean the test results, users can run

```
cd $RealESSI_PATH/
bash clean_all_verification.sh
```

Finally, users can also run a single group of test cases as follows.

209.7.2 Test Sequential Real-ESSI

In order to test whether the installation of sequential essi is successful, open the sequential example folder and run the bash script.

```
cd $RealESSI_PATH/CompGeoMechUCD_Miscellaneous/examples/analytic_solution
bash make_comparison.sh
```

This bash script will run all the examples automatically and compare the results to the analytic solutions. The comparison results are not only printed in the Terminal but also saved as a .log file in the same folder. Before you run the examples, make sure *essi* is in your PATH.

209.7.3 Test Parallel Real-ESSI

In order to test whether the installation of parallel essi is successful, open the parallel example folder and run the bash script.

```
cd $RealESSI_PATH/CompGeoMechUCD_Miscellaneous/examples/parallel
bash make_comparison.sh
```

This bash script will run all the examples automatically and compare the results to the analytic solutions. The comparison results are not only printed in the Terminal but also saved as a .log file in the same folder. Before you run the examples, make sure *essi_parallel* is in your PATH.

209.7.4 Version Stability Test

Since new features are continuously updated and improved in Real-ESSI, the version stability test helps the developers to guarantee their modification will not affect the correct operation of other code.

In order to test version stability,

```
cd $RealESSI_PATH/CompGeoMechUCD_Miscellaneous/examples/version_stability
bash generate_original.sh
```

This bash script will run all the examples automatically and save the results for reference later. This bash script above should run with the previous stable essi.

Then, to test the new essi and compare the results

```
cd $RealESSI_PATH/CompGeoMechUCD_Miscellaneous/examples/version_stability
bash make_comparison.sh
```

This bash script will run all the examples again and compare the results to the previous saved results. This bash script should run with the new essi. The comparison results are not only printed in the Terminal but also saved as a .log file in the same folder.

209.7.5 Memory Management Test

Memory management is important in C/C++ programming. This group of test cases helps the developers to track the memory leak in Real-ESSI. For the details about the code stability verification, please refer to the Section 303.2 on Page 1459.

Before you run the test cases, make sure Valgrind is installed. You can install Valgrind by this command.

```
sudo apt-get install valgrind
```

You can also download the source of Valgrind and compile it from scratch.

It is important to test memory leak in parallel simulations.

```
1 mpirun -np 3 valgrind --log-file='log_%p.valgrind' --leak-check=yes ↩
    essi-parallel-debug -f main.fei
```

A few important things to mention here:

- To test memory leak in parallel simulation, you obviously need a parallel version of Real-ESSI.
- Real-ESSI needs to be compiled in debug mode. This is important for Valgrind to capture and location the source of memory leaks.
- Running Real-ESSI in debug mode and in Valgrind means the simulation will be very slow. So it's not practical to run memory leak test using a large model. You should have a model with only a few elements/nodes (but more than 1 element so that it runs in parallel) that includes the specific functions you want to test.
- Valgrind log files will be saved in the location where you run the model. There will be multiple log files named as `log_processID.valgrind`. Each process will have its own Valgrind log file. There might be a few empty Valgrind log files generated, you can just ignore those. The number of Valgrind log files that actually contain memory leak information should be the same as the number of cores you use in your simulation.
- Valgrind is a powerful tool with many options. The command shown above is rather basic but serves as a good starting point. Memory leaks can be very tricky to track and fix. You should learn and experiment with Valgrind options for different issues you want to fix.

Valgrind log file can be very long and hard to read. At the bottom, there is a leak summary that looks like this: You should primarily focus on the 'definitely lost' result. 'Indirectly lost' and 'possibly lost'

```
==24551== LEAK SUMMARY:
==24551==    definitely lost: 1,329,224 bytes in 13,211 blocks
==24551==    indirectly lost: 674,019 bytes in 2,395 blocks
==24551==      possibly lost: 31,496 bytes in 9 blocks
==24551==    still reachable: 60,929,829 bytes in 8,999 blocks
==24551==         suppressed: 0 bytes in 0 blocks
```

Figure 209.1: Valgrind log file: Memory leak summary.

can also be problematic but should go away once you fix the source of 'definitely lost'. 'Still reachable' is usually not considered as actual memory leak but is something that can be optimized. Refer to the Valgrind User Manual for more information.

Valgrind log file contains detailed information on each memory leak. A typical leak detail looks like this. Following the trail, you should be able to locate the source of a specific leak and then fix it properly.

```
==24551== 493,960 bytes in 12,349 blocks are definitely lost in loss record 6,136 of 6,140
==24551==    at 0x4C3289F: operator new[](unsigned long) (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==24551==    by 0xB1BACD: Marray<double, 2, TinyArray_base<double, 2> >::compute_Determinant() (Marray_rank2.h:628)
==24551==    by 0x19BA8E8: ClassicElastoplasticMaterial<LinearIsotropic3D_EL, VonMises_YF<ArmstrongFrederickTensor_EV, LinearHardeningScala
==24551==    by 0x19AF12A: ClassicElastoplasticMaterial<LinearIsotropic3D_EL, VonMises_YF<ArmstrongFrederickTensor_EV, LinearHardeningScala
==24551==    by 0xC386B0: EightNodeBrick::update() (MasterEightNodeBrick.cpp:1950)
==24551==    by 0x15B3A3C: Domain::update() (Domain.cpp:3891)
==24551==    by 0x165C689: ActorSubdomain::update() (ActorSubdomain.cpp:1297)
==24551==    by 0x15766C2: AnalysisModel::updateDomain() (AnalysisModel.cpp:534)
==24551==    by 0x1572CFA: Newmark::update(Vector const&) (Newmark.cpp:570)
==24551==    by 0x157FA55: NewtonLineSearch::solveCurrentStep() (NewtonLineSearch.cpp:154)
==24551==    by 0x156926C: TransientDomainDecompositionAnalysis::analyze(double) (TransientDomainDecompositionAnalysis.cpp:242)
==24551==    by 0x1569AAC: TransientDomainDecompositionAnalysis::newStep(double) (TransientDomainDecompositionAnalysis.cpp:460)
==24551==
```

Figure 209.2: Valgrind log file: Memory leak detail.

A serious memory leak issue caused by external solvers used by PETSc was found. As shown in Figure 209.3, when the `mumps` option was used in parallel solver, a significant amount of memory leak was detected by Valgrind. More importantly, such memory leak was observed to increase with the number of time steps. This means large-scale, long-duration simulation could be interrupted due to not enough memory in the operating system. Note that this issue was also reported in other occasions where the `mumps` package is used within PETSc, as recent as June 2020.

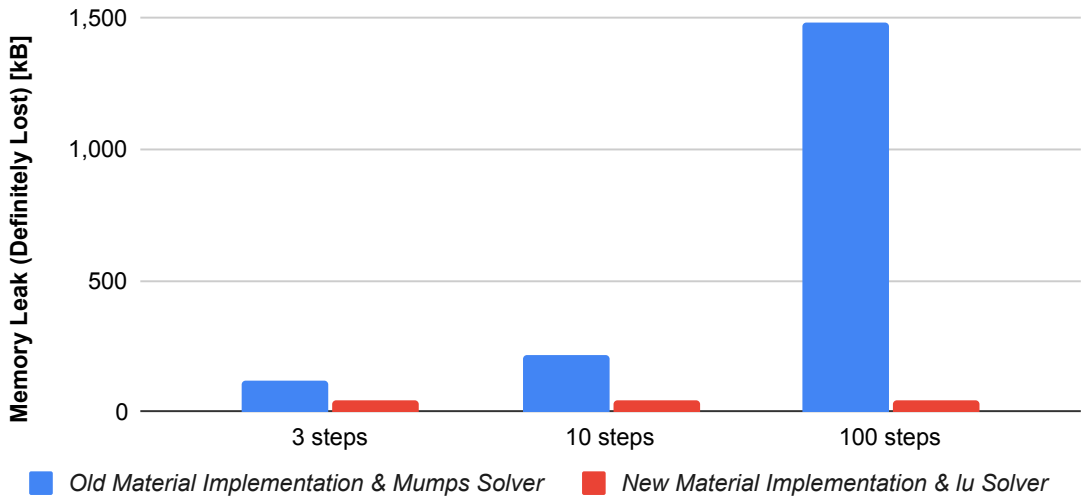


Figure 209.3: Comparison of memory leak between different PETSc solver options.

After extensive tests, it has been found out that other options/packages in PETSc don't have the memory leak issue mentioned above. Therefore it is recommended to use options other than `mumps` for large-scale, long-duration simulations. For example, the following command calls the default direct solver of PETSc:

```
define solver parallel petsc "-ksp_type preonly -pc_type lu" ;
```

209.8 Compiling Real-ESSI Utilities

Note: This section describes build procedure for old versions of Real-ESSI and/or its modules.

Real-ESSI comes with a lot of utilities to help the users speed up the simulation process. It provides mesh building, auto-input generation and visualization features which makes it quite nice.

Real-ESSI source code contains build_utilities script which can be used to build all the avialble utilities. We will go through the following subsection to introduce each utility and how to compile them.

The first step is to download all the sources of utilities that needs to be build. To do this, one has to run

```
cd Real-ESSI
./build_utilities download
```

This would download all the utilities sources in tar.gz format and would place them in "/SRC" of RealESSI_Utilities directory. The script is very powerfull and accepts targets that can be used to build a particular utility or all utilities at once. The available options to the scripts can be found by running the target help as shown below. A snippet is shown below

```
./build_utilities help

#usage: make [target]
#
#Utilities:
# gmessi Builds gmessi
# paraview Builds paraview
# pvessi Builds pvessi
# gmsh Builds gmsh
# visit Builds visit
# visitessi Builds visitessi
#
#Sequential:
# clean_utilities Cleans all utilities
#
#Default:
# all Builds all the necessary utilities for REAL-ESSI
# all_utils Builds all the necessary utilities for REAL-ESSI
# clean_all Cleans everything
# clean Cleans everything
#
#Check:
# check_utilities Checks if all utilities libraries are build
#
```

```
#Miscellaneous:
# list_utilities Lists all the available utilities version from SRC folder
# list_build_utilities Lists all the utilities library allready build in lib folder
# help Show this help.
# download Downloads the Utilities Sources
#
#Update:
# update_gmessi update gmessi utility
# update_pvessi update pvessi utility
# update_visitessi update visitessi utility
#
#Clean:
# clean_gmessi Clean gmessi utility
# clean_paraview Clean paraview utility
# clean_pvessi Clean pvessi utility
# clean_gmsh Cleans gmsh utility
# clean_visit Cleans visit
# clean_visitessi Cleans visitessi utility
```

The user can compile individual utilities by running just running

```
./build_utilities <utilitu_name>
```

Note: All the binaries of the utilities after build gets linked/copied to the RealESSI_Utilities/bin directory inside RealESSI_ROOT.

209.8.1 Installation of gmsh and gmESSI

gmsh is a 3-D finite element mesh generator for academic problems with parametric input and advanced visualization capabilities. It can be downloaded and installed from http://geuz.org/gmsh/. Additionally, the user can also install gmsh from terminal:

```
sudo apt-get install gmsh
```

gmESSI is effective pre-processor for generating Real-ESSI input files directly for the mesh file provided by gmsh. More information about gmESSI and how it works is given in Chapter 207 of the main document, lecture notes (Jeremić et al., 1989-2025). The gmESSI package is available from the main repository site: http://sokocalo.engr.ucdavis.edu/~jeremic/lecture_notes_online_material/_Chapter_SoftwareHardware_Pre_Processing/Real-ESSI_gmESSI.tgz. Before installing gmESSI, please install required libraries, as explained in Section ?? on Page ??. To install gmESSI, go to the Real-ESSI directory and then run

```
./build_utilities gmessi
```

To update the utility at the user just needs to run

```
./build_utilities update_gmessi
```

Refer to section 209.9 on page 1351 for instructions on what and how to install autocompletion and syntax coloring for gmESSI and Real-ESSI syntax on sublime text editor.

209.8.2 Installation of ParaView and PVESSIReader

ParaView package http://www.paraview.org/ is a powerfull multi-platform data analysis and visualization application avialable in Open Source. It can be run on supercomputers to analyze datasets of petascale size as well as on laptops for small datasets. ParaView can be used to visualize results of Real-ESSI simulations. A plug-in was developed for ParaView so that all the simulations results from Real-ESSI finite elements, material models and analysis types can be directly visualized, animated, etc.

209.8.2.1 Building ParaView and PVESSIReader Plugin from Source on Linux System

Note that ParaView, as well as its building procedure, has recently (during 2020) gone through some major changes. The building procedures shown in this section are mostly based on the information available at: https://gitlab.kitware.com/paraview/paraview/blob/master/Documentation/dev/build.md.

1. Install Dependencies

```
sudo apt-get install libgl1-mesa-dev libxt-dev qt5-default ↩
   libqt5x11extras5-dev libqt5help5 qttools5-dev qtxmlpatterns5-dev-tools ↩
   libqt5svg5-dev python3-dev python3-numpy ninja-build
```

2. Obtain the source of ParaView

```
git clone --recursive https://gitlab.kitware.com/paraview/paraview.git
cd paraview
git checkout v5.8.1
git submodule update --init --recursive
mkdir paraview_build
```

3. Obtain the source of PVESSIReader plugin

   - The source of PVESSIReader plugin can be downloaded from http://sokocalo.engr.ucdavis.edu/~jeremic/lecture_notes_online_material/_Chapter_SoftwareHardware_Post_Processing/_Real_ESSI_PVESSIReader_.zip

- Extract the files and move the `PVESSIReader` folder to `./Plugins/`

4. Modify the cmake file to include PVESSIReader plugin in the building process. Open the file `paraview/CMakeLists.txt` using your choice of text editor. Find "`set(paraview_default_plugins`" and add "`PVESSIReader`" to the end of the list of plugins.

5. Build

```
cd paraview_build
cmake -GNinja -DPARAVIEW_USE_PYTHON=ON -DPARAVIEW_USE_MPI=ON ↩
    -DVTK_SMP_IMPLEMENTATION_TYPE=TBB -DCMAKE_BUILD_TYPE=Release ..
ninja
```

6. Load PVESSIReader plugin into ParaView

   - Run the ParaView executable and click on Tools → Manage Plugins → Load New ...
   - Find `PVESSIReader.so` at `paraview/paraview_build/lib/paraview-5.9/plugins/PVESSIReader` and click OK to load it.
   - Now you should see PVESSIReader loaded in the list of plugins. Double click on it to expand advanced options and check Auto Load.
   - Close the ParaView application and reopen it. Now the PVESSIReader plugin should be automatically loaded and ready to use.

209.8.2.2 Building ParaView and PVESSIReader Plugin from Source on Windows System

Note that ParaView, as well as its building procedure, has recently (during 2020) gone through some major changes. The building procedures shown in this section are mostly based on the information available at: https://gitlab.kitware.com/paraview/paraview/blob/master/Documentation/dev/build.md. It is noted that user should be prepared to spend some time (perhaps hours) on installing procedure...

1. Install Dependencies

   - Download and install git bash for windows. Use the latest release version.
   - Download and install cmake. Use the lastest release version.
   - Download and install Visual Studio 2017 Community Edition. Please make sure that you tick the packages related to "Desktop Development with C++" and "Universal Windows Platform development".

- Download ninja-build and drop `ninja.exe` in `C:\Windows\`. Use the latest release version.
- Download and install both `msmpisetup.exe` and `msmpisdk.msi` from Microsoft MPI. Use the latest release version from Microsoft.
- Download and install Python for Windows. Latest release version should work fine. To avoid potential compatibility issues, install the same Python version that is used for the latest release of ParaView. (Currently, 01May2023, use Python 3.8.10 for Windows, as this version is the same as version used by Paraview 5.9.1.)
- Download and install Qt 5.12.3 for Windows, make sure to check the MSVC 2015 64-bit component during installation, make sure to add `C:\Qt\Qt5.12.3\5.12.3\msvc2017_64\bin` to your `PATH` environnement variable. Note that Qt for Windows is x86 but it works for x64 machine as well.

2. Obtain the source of ParaView, (Currently, 01May2023, version is Paraview 5.9.1.)

- Open your preferred Windows command prompt. Windows PowerShell is a nice tool for people usually work with Linux system. Git Bash application also works nice.
- To build ParaView developement version 5.9.1 (usually refered as "master"), run the following commands:

```
cd C:
mkdir pv
cd pv
git clone --recursive https://gitlab.kitware.com/paraview/paraview.git
mv paraview pv
mkdir pvb
cd pv
git checkout v5.9.1
git submodule update --init --recursive
```

- To build a specific ParaView version, please refer to https://gitlab.kitware.com/paraview/paraview/blob/master/Documentation/dev/build.md.

3. Obtain the source of PVESSIReader plugin

- The source of PVESSIReader plugin can be downloaded from http://sokocalo.engr.ucdavis.edu/~jeremic/lecture_notes_online_material/_Chapter_SoftwareHardware_Post_Processing/_Real_ESSI_PVESSIReader_.zip
- Extract the files and move the `PVESSIReader` folder to `C:\pv\Plugins\`

4. Modify the cmake file to include PVESSIReader plugin in the building process. Open the file `C:\pv\CMakeLists.txt` using your choice of text editor. Find "`set(paraview_default_plugins`" and add "`PVESSIReader`" to the end of the list of plugins.

5. Build

   - Open VS2017 x64 Native Tools Command Prompt and run the following commands

```
cd C:\pv\pvb
cmake -GNinja -DPARAVIEW_USE_PYTHON=ON -DPARAVIEW_USE_MPI=ON ↩
    -DCMAKE_BUILD_TYPE=Release ..\pv
ninja
```

   - This step could be take a few hours. If no configuration or compilation error is encountered, you should have the ParaView executable at `C:\pv\pvb\bin\`.
   - Download and install Python 3.9.11 for Windows, as needed to run ParaView executable.

6. Load PVESSIReader plugin into ParaView

   - Run the ParaView executable and click on Tools → Manage Plugins → Load New ...
   - Find `PVESSIReader.dll` at `C:\pv\pvb\bin\paraview-5.9\plugins\PVESSIReader\` and click OK to load it.
   - Now you should see PVESSIReader loaded in the list of plugins. Double click on it to expand advanced options and check Auto Load.
   - Close the ParaView application and reopen it. Now the PVESSIReader plugin should be automatically loaded and ready to use.

209.8.2.3 Building ParaView and PVESSIReader Plugin from Source on AWS

Currently, the AWS image has Ubuntu 18.04. This may change in the future. Because AWS is a remote server, properly running ParaView needs more steps in compilation. Note that most information here are based on this discussion.

1. Install Dependencies

```
sudo apt-get install libgl1-mesa-dev libxt-dev qt5-default ↩
    libqt5x11extras5-dev libqt5help5 qttools5-dev qtxmlpatterns5-dev-tools ↩
    libqt5svg5-dev python3-dev python3-numpy ninja-build gettext python-mako
```

2. Download, Build and Install LLVM

```
wget http://releases.llvm.org/7.0.1/llvm-7.0.1.src.tar.xz
mkdir llvm
cd llvm
tar -xvf /path/to/llvm-7.0.1.src.tar.xz
mkdir llvm_build
mkdir llvm_install
cd llvm_build
cmake \
  -DCMAKE_BUILD_TYPE=Release \
  -DBUILD_SHARED_LIBS=ON \
  -DCMAKE_INSTALL_PREFIX=/home/ubuntu/RealESSI_ROOT/RealESSI_Utilities/llvm/llvm_install \
      \
  -DLLVM_ENABLE_RTTI=ON \
  -DLLVM_INSTALL_UTILS=ON \
  -DLLVM_TARGETS_TO_BUILD:STRING=X86 \
  ../llvm-7.0.1.src
make -j8 install
```

3. Download, Build and Install Mesa

```
wget ↩
    https://gitlab.freedesktop.org/mesa/mesa/-/archive/mesa-18.3.3/mesa-mesa-18.3.3.tar.bz2
mkdir mesa
cd mesa
tar -xvf /path/to/mesa-mesa-18.3.3.tar.bz2
cd mesa-mesa-18.3.3
autoreconf --force --verbose --install
cd ../
mkdir mesa_build
mkdir mesa_install
cd mesa_build
../mesa-mesa-18.3.3/configure ↩
    --prefix=/home/ubuntu/RealESSI_ROOT/RealESSI_Utilities/mesa/mesa_install ↩
    \
--enable-opengl --disable-osmesa --disable-gallium-osmesa \
--enable-glx --with-platforms=x11 --disable-gles1 --disable-gles2 \
--disable-va --disable-gbm --disable-xvmc --disable-vdpau \
--disable-shared-glapi --disable-dri --with-dri-drivers= \
--enable-llvm ↩
    --with-llvm-prefix=/home/ubuntu/RealESSI_ROOT/RealESSI_Utilities/llvm/llvm_install ↩
    \
--with-gallium-drivers=swrast,swr --with-swr-archs=avx,avx2 --disable-egl
make -j8 install
```

4. Obtain the source of ParaView

```
git clone --recursive https://gitlab.kitware.com/paraview/paraview.git
cd paraview
git checkout v5.8.1
```

```
4 git submodule update --init --recursive
```

5. Obtain the source of PVESSIReader plugin

   - The source of PVESSIReader plugin can be downloaded from http://sokocalo.engr.ucdavis.edu/~jeremic/lecture_notes_online_material/_Chapter_SoftwareHardware_Post_Processing/_Real_ESSI_PVESSIReader_.zip
   - Extract the files and move the `PVESSIReader` folder to `./Plugins/`

6. Modify the cmake file to include PVESSIReader plugin in the building process. Open the file `paraview/CMakeLists.txt` using your choice of text editor. Find "`set(paraview_default_plugins`" and add "`PVESSIReader`" to the end of the list of plugins.

7. Build ParaView with Mesa

```
1  mkdir paraview_build
2  cd paraview_build
3  cmake -GNinja ↩
       -DOPENGL_gl_LIBRARY=/home/ubuntu/RealESSI_ROOT/RealESSI_Utilities/mesa/mesa_install/lib
       \
4  -DOPENGL_INCLUDE_DIR=/home/ubuntu/RealESSI_ROOT/RealESSI_Utilities/mesa/mesa_install/inclu
       \
5  -DOPENGL_EGL_INCLUDE_DIR= -DOPENGL_GLES2_INCLUDE_DIR= \
6  -DOPENGL_GLES3_INCLUDE_DIR= -DOPENGL_GLX_INCLUDE_DIR= \
7  -DOPENGL_egl_LIBRARY= -DOPENGL_gles2_LIBRARY= \
8  -DOPENGL_gles3_LIBRARY= -DOPENGL_glu_LIBRARY= \
9  -DOPENGL_glx_LIBRARY= -DOPENGL_opengl_LIBRARY= ../paraview \
10 -DPARAVIEW_USE_PYTHON=ON -DPARAVIEW_USE_MPI=ON \
11 -DVTK_SMP_IMPLEMENTATION_TYPE=TBB -DCMAKE_BUILD_TYPE=Release ../paraview
12 LD_LIBRARY_PATH=/home/ubuntu/RealESSI_ROOT/RealESSI_Utilities/llvm/llvm_install/lib ↩
       ninja
```

   Note: There are many warnings showed up when I was doing this step, but it doesn't seem to terminate the build process.

8. Run ParaView with Mesa

```
1 LD_LIBRARY_PATH=/home/ubuntu/RealESSI_ROOT/RealESSI_Utilities/llvm/llvm_install/lib:/home/
      paraview
```

9. Load PVESSIReader plugin into ParaView

   - Run the ParaView executable and click on Tools → Manage Plugins → Load New ...

- Find `PVESSIReader.so` at `paraview/paraview_build/lib/paraview-5.9/plugins/PVESSIReader` and click OK to load it.
- Now you should see PVESSIReader loaded in the list of plugins. Double click on it to expand advanced options and check Auto Load.
- Close the ParaView application and reopen it. Now the PVESSIReader plugin should be automatically loaded and ready to use.

209.9 Sublime Text Editor

Note: This section describes build procedure for old versions of Real-ESSI and/or its modules.

Install `sublime` text editor from http://www.sublimetext.com/. Then install package control to sublime in order to install plugins. (go to preferences, package control, install package.) Then install two packages:

FEI Syntax-n-Snippets, Real-ESSI syntax and auto completion plugin for [].fei files (input files for Real-ESSI program).

gmsh-Tools, syntax and autotext completion for gmsh model development tools for Real-ESSI.

gmESSI-Tools, syntax and autotext completion for gmESSI model development tools for Real-ESSI.

209.10 Model Conversion/Translation using `FeConv`

FeConv allows conversion/translation of input files (models) between Real-ESSI and SASSI, Sofistik, Ansys, OpenSees and Strudyn. FeConv was developed and is maintained by Mr. Viktor Vlaski.

209.11 Build Procedures on Amazon Web Service

This section shows the steps to install a new Real-ESSI image on Amazon Web Service (AWS). This document is only intended for Real-ESSI developers, not for general users. For using Real-ESSI on AWS, please refer to Chapter 211, on page 1373 in Jeremić et al. (1989-2025).

Noted that when creating a new image, the instance type should be consistent with future usage. For example, if the user intend to launch a Real-ESSI instance using the instance type "General Purpose", such as the T2 series, the image should also be created with the same instance type. If the image is created with a different instance type, Real-ESSI will not be able to run, and the following error message will be observed:

```
Illegal instruction (core dumped)
```

209.11.1 Sign In to AWS

Here is the link to the AWS sign in page. Click "Sign In to the Console" button on the upper right corner of the page. No need to register a new account. You should already have the account ID, IAM user name, and password for AWS sign in. If not, please contact an administrator to add you to the developers' group.

After sign in, go to the "EC2" tab under "Service". Here you can view all your instances and AMIs. This is where you can start new simulations or install new images.

Note that you probably also need to choose the correct region. On the upper right corner of the page, you can see your current region and switch to another one if necessary.

209.11.2 Copy an Existing Image

Since we already have a few images for Real-ESSI, the most efficient way to create a new image is to simply copy an existing one. To do this, go to the "AMIs" tab under "IMAGES" on the left part of the page. Now you should be able to view all existing images.

Select the image that you want to copy. Click the "Actions" button and choose "Copy AMI". On the pop-up window, enter the informations of this new image that you want to create. Then just click the "Copy AMI" button.

Now you should have a new image that has been installed with all the Real-ESSI components. To make any change inside this image, you need to launch it as a new instance and access it using X2GO. Procedures to install and use X2GO can be found in Chapter 211. For cloud server, on AWS or similar, the build procedures are the same as those for local installation, which can be found in previous sections of this chapter.

209.11.3 Create a New Image

If you need to create a new Real-ESSI image from scratch, this section shows the steps to do so. First sign in to AWS and go to "EC2". Choose the correct region. Click the "Instance" tab under "INSTANCES" on the left part of the page. Choose "Launch Instance" to start a new instance that later will be saved as your new image.

Then, follow these steps:

1. Choose AMI: Ubuntu Server 16.04 LTS (HVM), SSD Volume Type.

2. Choose Instance Type: Family = Compute optimized, Type = c5.4xlarge, vCPUs = 16, Memory (GiB) = 32.

3. Keep other options as default, and click "Review and Launch".

4. Review the information of the new instance, and click "Launch".

Next, you are asked to choose a key pair for your instance. It's recommended to create a new key pair for the first time, then use it in the future. First, choose "Create a new key pair", and enter a name. Click the "Download Key Pair" button. Save the key in a secure directory in your local computer for future use, for example in .ssh directory.

Now, you can select "Choose an existing key pair", and select your key pair that should be visible. Check the box for acknowledging the use of a private key. Finally, your new instance is launched. Note that this new instance is a brand new Ubuntu server, which means that you need to install everything.

At this point, the new Ubuntu server on AWS does not have X2GO for remote access or a GUI desktop to operate. We will now install these necessary softwares. First, run the following command to access the remote Ubuntu server on AWS using ssh. Note that you need to change the name of your ssh key to the one you just created. The public IP address can be found on the AWS webpage where you launched your new instance. Go the description of your instance to find the "IPv4 Public IP".

```
chmod 400 your_ssh_key.pem
ssh -i your_ssh_key.pem ubuntu@your_AWS_public_IP_address
```

Run the following command to install X2GO server on Ubuntu Linux.

```
sudo apt-get install software-properties-common
sudo add-apt-repository ppa:x2go/stable
sudo apt-get update
sudo apt-get install x2goserver x2goserver-xsession
```

Xfce is a lightweight desktop and ideal for usage on a remote server. Run the following command to install xfce on Ubuntu.

```
sudo apt-get install xfce4 xfce4-goodies
```

Now you can access your new instance (the remote Ubuntu server) using X2GO. Steps to do this can be found in Chapter 211. After you established remote control of the Ubuntu server on AWS, the build procedures are the same as those for local installation, which can be found in previous sections of this chapter.

The last step is to create a new image from this instance so that you can launch it in the future. Go the "Instances", and choose the correct instance. Click "Actions", and select "Create Image" under

"Image". You can change the size of the instance volume, but it's not necessary at this moment. Give your image a name and a description, and click "Create Image". Now you have sucessfully created a new image for Real-ESSI. If you go to "AMIs", you should be able to see this new image you just created.

209.11.4 Build AWS ESSI Image from Scratch

This section is a developer guide, which presents the procedures to build AWS ESSI Image from scratch. ESSI AWS users do not need to know the technical details in this section.

1. Launch EC2 instance from an AWS blank image: Ubuntu 16.04 Server.

   EC2 Dashboard → Instances → Instance → Launch Instance.

   Choose

   ```
   Ubuntu Server 16.04 (HVM), SSD Volume Type.
   ```

   Since there is no Desktop version available, so we have to launch the server version and install desktop by ourself.

   You will need to download a .pem key to launch the instance.

2. Login to the Remote Instance using Terminal.

   Copy the external IP address of the remote instance from the Browser.

   Use the downloaded .pem key to login to the remote instance.

   ```
   chmod 400 your_key.pem
   ssh -i your_key.pem ubuntu@your_remote_instance_IP
   ```

3. Install Desktop and git on AWS Remote Instance

   ```
   sudo apt update
   sudo apt install -y ubuntu-desktop git
   ```

4. Install remote-desktop-server (x2goserver) on AWS Remote Instance

   ```
   sudo add-apt-repository ppa:x2go/stable
   sudo apt update
   sudo apt install -y x2goserver x2goserver-xsession xfce4
   ```

5. Set up the automatic launch of remote desktop server

   ```
   sudo systemctl enable x2goserver.service
   sudo systemctl start x2goserver.service
   ```

6. Install ESSI

```
# Install prerequisite
sudo apt install -y cmake
sudo apt install -y build-essential
sudo apt install -y zlib1g-dev
sudo apt install -y libtbb-dev
sudo apt install -y bison flex
sudo apt install -y libboost-dev
sudo apt install -y python
sudo apt install -y gfortran
sudo apt install -y libopenblas-dev
sudo apt install -y liblapack-dev
sudo apt install -y python-scipy
sudo apt install -y libhdf5-dev libhdf5-cpp-11
sudo apt install -y python-h5py
sudo apt install -y python-matplotlib
sudo apt install -y libssl-dev

# Download ESSI
#
# using curly brackets to help in checking scripts, that rely on these
# brackets being available around URL
#
git clone {https://github.com/BorisJeremic/Real-ESSI.git} # Need ↩
    permission from Boris Jeremic for Real-ESSI on github
cd Real-ESSI

# Build ESSI Dependencies
./build_libraries download
./build_libraries sequential
./build_libraries hdf5_sequential
./build_libraries suitesparse
./build_libraries arpack
./build_libraries parmetis
./build_libraries petsc

# Build Sequential ESSI
mkdir build
cd build
cmake ..
make -j $(nproc)
cd ..


# Build Parallel ESSI
mkdir build_parallel
cd build_parallel
cmake -DCMAKE_CXX_COMPILER=/usr/bin/mpic++ -DPETSC_HAS_MUMPS=TRUE ↩
    -DPROGRAMMING_MODE=PARALLEL ..
make -j $(nproc)
cd ..
```

7. Install gmsh

```
sudo apt install -y gmsh
```

8. Install gmESSI

```
# Install the prerequisite
sudo apt install -y libboost-all-dev
sudo apt install -y build-essential
sudo apt install -y python-dev
sudo apt install -y liboctave-dev

# Install gmESSI
## download the package from the main Real-ESSI repository
#
# using curly brackets to help in checking scripts, that rely on these
# brackets being available around URL
#
wget ↩
    {http://sokocalo.engr.ucdavis.edu/~jeremic/Real_ESSI_Simulator/gmESSI/_all_files_gmESSI
mkdir Real-ESSI-gmESSI
mv _all_files_gmESSI_.tgz Real-ESSI-gmESSI
cd Real-ESSI-gmESSI

make -j $(nproc)

# Add binary PATH to ~/.bashrc
cd ./build/bin/
part1="export PATH=\""
part2=$PWD
part3=":\$PATH\""
newline=$part1$part2$part3
echo $newline >> ~/.bashrc
```

9. Install ParaView with PVESSIReader plugin

```
# Install the prerequisite
sudo apt install -y libavformat-dev
sudo apt install -y libswscale-dev
sudo apt install -y ffmpeg
sudo apt install -y libphonon-dev libphonon4 qt4-dev-tools
sudo apt install -y libqt4-core libqt4-gui qt4-qmake libxt-dev
sudo apt install -y g++ gcc cmake-curses-gui libqt4-opengl-dev
sudo apt install -y mesa-common-dev python-dev
sudo apt install -y libvtk6.2
sudo apt install -y mpich libopenmpi-dev
sudo apt install -y libxmu-dev libxi-dev

# Download the ParaView
#
```

```
# using curly brackets to help in checking scripts, that rely on these
# brackets being available around URL
#
git clone {https://github.com/Kitware/ParaView.git}
cd ParaView
git checkout v5.1.2
git submodule update --init

# Download the Plugin
cd Plugins
#
# using curly brackets to help in checking scripts, that rely on these
# brackets being available around URL
#
wget ↩
    {http://sokocalo.engr.ucdavis.edu/~jeremic/Real_ESSI_Simulator/pvESSI/_pvESSI_all_files
tar -xvzf _pvESSI_all_files_.tgz
cd ..

# Compile ParaView along with PVESSIReader
mkdir build && cd build
cmake -DPARAVIEW_USE_MPI=true -DPARAVIEW_ENABLE_PYTHON=true ↩
    -DPARAVIEW_ENABLE_FFMPEG=true ..
make -j $(nproc) # require Internet during ParaView compilation.

# Add binary PATH to ~/.bashrc
cd bin
part1="export PATH=\""
part2=$PWD
part3=":\$PATH\""
newline=$part1$part2$part3
echo $newline >> ~/.bashrc
```

10. Install Sublime Text 3 and ESSI plugin. Following this link.

```
#
# using curly brackets to help in checking scripts, that rely on these
# brackets being available around URL
#
wget -qO - {https://download.sublimetext.com/sublimehq-pub.gpg} | sudo ↩
    apt-key add -
sudo apt-get install apt-transport-https
echo "deb {https://download.sublimetext.com/ apt/stable/}" | sudo tee ↩
    /etc/apt/sources.list.d/sublime-text.list
sudo apt-get update
sudo apt-get install sublime-text
```

11. Install Sublime Text Plugin:

```
# Inside Sublime Text Window
```

```
# Ctrl+Shift+P, then Type
install package control

# Ctrl+Shift+P, then Type
install package # press ENTER, then type
fei syntax-n-snippets

# Ctrl+Shift+P, then Type
install package # press ENTER, then type
gmESSI-Tools

# Ctrl+Shift+P, then Type
install package # press ENTER, then type
gmsh-Tools
```

12. Create Image inside Browser.

    Select the launched Image with the above software installed.

    Choose Actions → Image → Create Image.

    Type your Image Name and descriptions.

    You will then see your image in EC2 Dashboard → Images → AMIs

209.11.5 Update an Existing Image

For updating an existing image, for example for a new version or release follow instruction below. First sign in to AWS and go to "EC2". Choose the correct region. Click the "Instance" tab under "INSTANCES" on the left part of the page. Choose "Launch Instance" to start a new instance that later will be saved as your new image.

Then, follow these steps:

1. Choose an existing AMI, for example GlobalRelease...
2. Choose Instance Type, for example: Family = Compute optimized, Type = c5.4xlarge, vCPUs = 16, Memory (GiB) = 32.
3. Keep other options as default, and click "Review and Launch".
4. Review the information of the new instance, and click "Launch".

Next, you are asked to choose a key pair for your instance. It's recommended to create a new key pair for the first time, then use it in the future. That is the keypair that is saved, for example in .ssh.

Now, you can select "Choose an existing key pair", and select your key pair that should be visible. Check the box for acknowledging the use of a private key. Finally, your new instance is launched. Note that this new instance is an already existing Ubuntu server/image. This image is the one we will update.

Now you can access your new instance (the remote Ubuntu server) using X2GO. Steps to do this can be found in Chapter 211. After you established remote control of the Ubuntu server on AWS, the build procedures are the same as those for local installation, which can be found in previous sections of this chapter.

The last step is to create a new image from this instance so that you can launch it in the future. Go the "Instances", and choose the correct instance. Click "Actions", and select "Create Image" under "Image". You can change the size of the instance volume, but it's not necessary at this moment. Give your image a (new) name and a description, and click "Create Image". Now you have sucessfully created a new image for Real-ESSI. If you go to "AMIs", you should be able to see this new image you just created.

Now you can go to `Software` directory and follow install procedures from section ?? on page ??.

After compiling and linking both sequential and parallel Real-ESSI, and install them on /usr/bin (follow procedures for build), and delete source code (!), one can make this instance into a new image. Create new image inside AWS EC2 Management Console Browser window. Select the launched Image with the above software installed. Choose Actions → Image → Create Image. Type your Image Name and descriptions. Click Create Image. This might take some time. You will then see your image in EC2 Dashboard → Images → AMIs (on the left side bar).

Make sure that you terminate all the running instances so that you do not get charged. Find: Action, Instance State, Terminate.

209.11.6 Upload an Existing Real-ESSI Simulator Image to AWS MarketPlace

- Copy to private image for region North Virginia
- Go to the AWS market place `https://aws.amazon.com/marketplace`,
- Choose sell in AWS marketplace,
- Choose AMIs selection the new private in Region North Virginia to publish.
- Proceed until finalizing the AWS Marketplace Image.

Chapter 210

Software Platform Procurement, Distribution

(2019-2020-2021-)

(In collaboration with Prof. Han Yang and Dr. Hexiang Wang)

210.1 Chapter Summary and Highlights

210.2 Introduction

The Real-ESSI Simulator program (Jeremić et al., 1988-2025) can be installed on user's computers in a number of different ways:

- The most efficient executables are created when Real-ESSI sources are compiled on user computer. Compilation is performed using batch scripts that execute all the necessary operations. This process takes approximately 40 minutes, for both sequential and parallel versions of the Real-ESSI. It is assumed that all the necessary libraries are installed prior to this. More details about this mode of installation are given in section 209, on page 1324 of the main document (Jeremić et al., 1989-2025). For this mode of installation, sources for the Real-ESSI need to be made available. Sources for the Real-ESSI program are usually not distributed, except to collaborators and in some other special circumstances.

- The Real-ESSI program can also be downloaded and installed as a Debian package, starting from version 22.07, built for Ubuntu 22.04 LTS. The Debian package contains the sequential and parallel Real-ESSI executables, The gmESSI tool for pre-processing using Gmsh, and pvESSI tool for post-processing using ParaView and other useful external programs, like Gmsh and ParaView, will NOT be automatically installed when installing the Real-ESSI Debian package. This change was made since those other packages should be installed using their own installation procedures, that have gone though some recent changes. Therefor installation of those packages is best done directly using downloaded version from their own web site, and then connecting them to the Real-ESSI Simulator systems using Gmsh and pvESSI tools. Installation of Gmsh and pvESSI tools is described in:

  - Installation of pre-processing modules is described in Chapter 207, on page 1221 in Jeremić et al. (1989-2025).
  - Installation of post-processing modules is described in Chapter 208, on page 1287 in Jeremić et al. (1989-2025).

  It is noted that old installations of Real-ESSI main program and gmsh and ParaView should be removed before the Real-ESSI Simulator systems is installed from Debian package.

- The Real-ESSI program can also be installed through direct download of program executables, as noted in section 210.4. These executable were build without use of any special optimization

options, so they are not very efficient, and do not use special, high performance features of most modern CPUs. On the other hand these generic executables will run, execute on most computers.

- Docker support is discontinued since Windows users can now use Windows Subsystem for Linux (WSL), and install Real-ESSI Simulator Debian package. The Real-ESSI program can also be installed through a docker container, as described in section 210.5. Similar to the previous case, these executable were developed without special optimization options, so they are not very efficient. However, Real-ESSI program will these generic executables will run, within docker container, on all computers.

210.3 Real-ESSI Program Debian Package Download and Install

210.3.1 System Libraries Update/Upgrade

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get dist-upgrade
sudo apt-get autoremove
```

sudo apt-get update

sudo apt-get upgrade

sudo apt-get dist-upgrade

sudo apt-get autoremove

210.3.2 Real-ESSI Debian Package Download

The Real-ESSI program Debian package can be downloaded from Real-ESSI Simulator website: http://real-essi.info/. Alternatively, contact Prof. Jeremić to arrange for customized Real-ESSI Debian package.

210.3.3 Real-ESSI Debian Package Install

Start the Real-ESSI Simulator Debian package install by removing the old installations of Real-ESSI program, pre-processor gmsh/gmESSI and post-processor ParaView/pvESSI. Then, go to the directory where you have downloaded the Real-ESSI Debian package. Install the Debian package, for example use the following command:

```
sudo apt install ./real-essi_22.07-1_amd64.deb
```

Note that some warning messages might appear but they don't affect the installation.

After a successful installation, the sequential and parallel Real-ESSI executables. are installed and ready to use.

210.3.4 Load pvESSI Plugin in ParaView

Install ParaView system using installation procedure described on their web site. Then install pvESSI plugin. Start `ParaView` and click 'Tools' → 'Manage Plugins...'. Click 'Load New...' and find the plugin named 'PVESSIReader.so' under directory `/opt/paraview/lib/paraview-5.10/plugins/PVESSIReader/`. Also check the box 'Auto Load' then close `ParaView`. Next time when ParaView is started, Real-ESSI output files can be visualized and post-processed.

210.3.5 Install Other Useful Programs

210.3.5.1 HDFView

`HDFView` can be used to open Real-ESSI output files, which are in HDF5 format. Download the latest version of `HDFView` from https://support.hdfgroup.org/ftp/HDF5/releases/HDF-JAVA/. Click on the latest version, which is `hdfview-3.2.0` as of June 2022. Go to `bin/`, click `HDFView-3.2.0-ubuntu2004_64.tar.gz` and save the file in your `./Downloads/` directory. Then extract and install HDFView:

```
cd
tar -xvf ./Downloads/HDFView-3.2.0-ubuntu2004_64.tar.gz -C ./Downloads
sudo apt install -y ./Downloads/hdfview_3.2.0-1_amd64.deb
sudo ln -s /opt/hdfview/bin/HDFView /usr/local/bin/hdfview
```

Now you can use `HDFView` from a terminal. To be able to use `HDFView` when you click on a Real-ESSI output file, do the following additional steps. First open the file using the following command:

```
sudo gedit /usr/share/applications/hdfview-HDFView.desktop
```

Find the line:

```
Exec=/opt/hdfview/bin/HDFView
```

Replace it with:

```
Exec=/opt/hdfview/bin/HDFView %F
```

Save the file and close it.

Go to a Real-ESSI output file, which should have the suffix 'h5.feioutput'. Right click on the file and select 'Open with Other Application'. Click 'View All Applications' and choose `HDFView` from the list.

Note that you only need to do this once. Next time when you click on a Real-ESSI output file, it will be opened automatically using `HDFView`.

210.3.5.2 Sublime Text

Sublime Text (https://www.sublimetext.com/) is the recommended editor for Real-ESSI input files and pre-processing files. Install Sublime Text using the following command:

```
wget -qO - https://download.sublimetext.com/sublimehq-pub.gpg | gpg --dearmor | ↩
    sudo tee /etc/apt/trusted.gpg.d/sublimehq-archive.gpg
echo "deb https://download.sublimetext.com/ apt/stable/" | sudo tee ↩
    /etc/apt/sources.list.d/sublime-text.list
sudo apt-get update
sudo apt-get install sublime-text
```

Open Sublime Text. Open the 'Tools' menu and select 'Install Package Control...'. Open the 'Preferences' menu, select 'Package Control', then select 'Package Control: Install Package'.

In the opened search bar, type the package name and click on the package to install it. Three packages should be installed: `FEI Syntax-n-Snippets`, `gmsh-Tools`, and `gmESSI-Tools`.

210.4 Real-ESSI Program Executables Download and Install

Executables for the Real-ESSI Simulator program (Jeremić et al., 1988-2025) are available online. Prebuilt executables are available for Linux, Ubuntu 18.04, and can be downloaded and installed by analyst.

In order for prebuild executables to be able to run on a user/analyst computer, system libraries have to be brought up to date and additional libraries installed. System libraries update/upgrade:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get dist-upgrade
sudo apt-get autoremove
```

For sequential and/or parallel version of Real-ESSI, additional libraries are needed, as described below.

210.4.1 Sequential Version of Real-ESSI Program.

Libraries required to be installed for using sequential version of the Real ESSI program:

```
sudo apt-get install libboost-all-dev
sudo apt-get install libhdf5-dev
sudo apt-get install libtbb-dev
sudo apt-get install libssl1.0.0
```

210.4.2 Parallel Version of Real-ESSI Program.

Libraries required to be installed for executing parallel version of the Real ESSI program:

```
sudo apt-get install libboost-all-dev
sudo apt-get install libhdf5-dev
sudo apt-get install libtbb-dev
sudo apt-get install mpich
sudo apt-get install libopenmpi-dev
sudo apt-get install libssl1.0.0
```

210.4.3 Real-ESSI Executable Downloads.

The Real-ESSI program executables can be downloaded from Real-ESSI Simulator website: http://real-essi.info/. Alternatively, contact Prof. Jeremić to arrange for customized Real-ESSI executables.

210.5 DISCONTINUED, use WSL! Real-ESSI Simulator Install as Container through Docker

Docker support is discontinued since Windows users can now use Windows Subsystem for Linux (WSL), and install Real-ESSI Simulator Debian package.

Recent developments in virtualization of operating systems (OS) has created an opportunity to deploy programs and software systems as container images. Container images are used by the host OS (Linux, Windows, MacOS) to create a container. A container is a running instance of a container image, and is represented by a Linux/Windows/MacOS process that can be used to run programs that are installed within container. Programs that are installed within a container have all the necessary libraries available within container and are fully self sufficient, irrespective of what container host OS is used, be it Linux or Windows or MacOS.

More information used virtualization, containers, docker, etc. can be found at:

- https://en.wikipedia.org/wiki/OS-level_virtualization
- https://en.wikipedia.org/wiki/Docker_(software)
- https://developers.redhat.com/blog/2018/02/22/container-terminology-practical-introduction

Starting from Real-ESSI version 20.07, Real-ESSI Simulator is now available as a Docker Container Image, and can be installed and used on Linux, Windows and MacOS.

210.5.1 DISCONTINUED, use WSL! Real-ESSI Docker Image Development

This section is intended for Real-ESSI developers, users can skip this section. The development of Real-ESSI Docker image follows typical steps to 'dockerize' any application. Here are some very helpful sources:

- Official documentation: https://docs.docker.com/
- A Docker Tutorial for Beginners: https://docker-curriculum.com/#our-first-image
- How to dockerize any application: https://hackernoon.com/how-to-dockerize-any-application-b60ad00e76da
- Slimming Down Your Docker Images: https://towardsdatascience.com/slimming-down-your-docker-images-275f0ca9337e

It should be mentioned that there are many different ways and styles that can be employed to create Docker image. Here, multistage build is used to save build/debug time and, more importantly, reduce size of the final image.

Provided below are steps used to create the Real-ESSI Docker image.

- Obtain the source code of Real-ESSI.
- The following 'Dockerfile' is created to build the Real-ESSI Docker image.

```
FROM ubuntu:18.04 AS basesystem

MAINTAINER Han Yang <hhhyang@ucdavis.edu>

WORKDIR /usr/src

COPY . .

RUN useradd -m ubuntu && \
        apt-get update && apt-get install -y \
        bison \
        build-essential \
        cmake \
        flex \
        libboost-all-dev \
        libhdf5-serial-dev \
        liblapack-dev \
        libopenblas-dev \
        libopenmpi-dev \
        libpthread-workqueue-dev \
```

```
        libssl-dev \
        libtbb-dev \
        mpich \
        ssh \
        valgrind \
        wget \
        zlib1g-dev



FROM basesystem AS dependencies

RUN cd Real-ESSI && \
        mkdir -p ../RealESSI_Dependencies && \
        mkdir -p ../RealESSI_Dependencies/include && \
        mkdir -p ../RealESSI_Dependencies/lib && \
        mkdir -p ../RealESSI_Dependencies/bin && \
        mkdir -p ../RealESSI_Dependencies/SRC && \
        cd ../RealESSI_Dependencies && \
        wget ↩
            http://sokocalo.engr.ucdavis.edu/~jeremic/RealESSI/Dependencies_SRC.tar.gz ↩
            && \
        tar -xzvf ./Dependencies_SRC.tar.gz -C ./SRC --strip-components 1 ↩
            && \
        cd ../Real-ESSI && \
        ./build_libraries suitesparse && \
        ./build_libraries arpack && \
        ./build_libraries hdf5_sequential && \
        ./build_libraries tbb && \
        ./build_libraries lapack && \
        ./build_libraries parmetis && \
        ./build_libraries petsc_itself



FROM dependencies AS builder

RUN cd Real-ESSI && \
        mkdir build && \
        cd build && \
        cmake .. && \
        make -j 16 && \
        cp essi essi_sequential && \
        cd .. && \
        mkdir pbuild && \
        cd pbuild && \
        cmake -DCMAKE_CXX_COMPILER=/usr/bin/mpic++ ↩
            -DPROGRAMMING_MODE=PARALLEL .. && \
        make -j 16 && \
        cp essi essi_parallel
```

```
FROM ubuntu:18.04

MAINTAINER Han Yang <hhhyang@ucdavis.edu>

RUN useradd -m ubuntu && \
       apt-get update && apt-get install -y \
       libboost-all-dev \
       libhdf5-dev \
       libopenmpi-dev \
       libtbb-dev \
       mpich \
       ssh

COPY --from=builder /usr/src/Real-ESSI/build/essi_sequential ↩
   /usr/src/Real-ESSI/pbuild/essi_parallel /usr/bin/

USER ubuntu

WORKDIR /workspace

VOLUME ["/workspace"]
```

- Put the 'Dockerfile' in the same directory with the source code of Real-ESSI.
- Build the Real-ESSI Docker image. This step usually takes a long time, especially for the first time.

```
docker build -t realessilocal:test .
```

- Correctly tag your image. This is not only necessary for later push but also just a good practice to organize your Docker images.

```
docker tag realessilocal:test realessi/real-essi-repo:<tag>
```

  Replace <tag> with the tag you want to use. It's usually a version name.

- Push your build to Docker Hub. Make sure you have the proper permission to do so.

```
docker push realessi/real-essi-repo:<tag>
```

210.5.2 DISCONTINUED, use WSL! Running Real-ESSI Container through Docker

Provided below are steps needed to install and run Real-ESSI within a Docker Container. The following steps work for both Linux and Windows systems. In a Linux system, run the following commands in a

terminal. In a Windows system, run these commands in PowerShell. It should also work for Mac OS but hasn't been tested yet.

- Install Docker on the local computer, desktop, laptop. Documentation on how to install Docker on user OS can be found here:
  - Linux: https://docs.docker.com/engine/install/#server
  - Windows: https://docs.docker.com/docker-for-windows/install/
  - MacOS: https://docs.docker.com/docker-for-mac/install/
- Manage Docker as a non-root user on Linux hosts

  If you are using a Linux host, by default you need to run Docker using `sudo`. If you don't want to preface the docker command with `sudo`, create a group called `docker` and add users to it.

  To create the `docker` group and add your user:

  1. Create the `docker` group.

     ```
     sudo groupadd docker
     ```

     Sometimes the `docker` group might already exist after the installation of Docker. This is okay, just move on to the next step.

  2. Add your user to the `docker` group.

     ```
     sudo usermod -aG docker $USER
     ```

     Replace `$USER` with your user name.

  3. Log out and log back in so that your group membership is re-evaluated. On Linux, you can also run the following command to activate the changes to groups:

     ```
     newgrp docker
     ```

  4. Verify that you can run `docker` commands without `sudo`.

     ```
     docker run hello-world
     ```

     This command downloads a test image and runs it in a container. When the container runs, it prints an informational message and exits.

  More information on managing Docker as a non-root user can be found here: https://docs.docker.com/engine/install/linux-postinstall/

- Pull the Real-ESSI image

```
docker pull realessi/real-essi-repo:tag
```

Replace `tag` with the latest version of Real-ESSI. For example, if the latest version is 23.01, then the pull command is `docker pull realessi/real-essi-repo:23.01`.

Current Real-ESSI Simulator version is kept up to date at the Real-ESSI web site HERE. In adition, you can find tags of Real-ESSI at
https://hub.docker.com/repository/docker/realessi/real-essi-repo/tags.

- Run the Real-ESSI image:

```
docker run -it --rm -v your_working_directory:/workspace ↩
   realessi/real-essi-repo:tag
```

Again, replace `tag` with the version of Real-ESSI you pulled. Once you start running the Real-ESSI Docker image, you are working inside the container. The container is Ubuntu 18.04 with Real-ESSI installed. Note that you should replace `your_working_directory` with the absolute path of your working directory.

- Run Real-ESSI:

```
essi_sequential -f main.fei
```

Note that the current directory on your local machine is shared with the container, so it can work with any files there. The files need to have the correct permissions to be run by a non-administrator user. You can move files after the container started and they will be recognized by the container.

After the simulation is finished, simply exit the container. You will see the output files and log file in your current directory. They will not be erased when you exit the container.

210.5.3 DISCONTINUED, use WSL! Performance of Real-ESSI Container

To test the performance of Real-ESSI container, a series of sequential and parallel simulations are conducted. The results and comparison are summarized in Figure 210.1.

210.6 Real-ESSI Simulator System Install

In adition to the Real-ESSI Program, Real-ESSI Simulator system consists of a pre-processing modules and post-processing modules. Installation of pre-processing modules is described in Chapter 207, on

Steel Frame - Sequential			
Local Run	Container Run		
cml04	cml04	cml05	Windows laptop
6.051s	7.159s	7.207s	1m2.156s

Steel Frame - Parallel (3)			
Local Run	Container Run		
cml04	cml04	cml05	Windows laptop
4.981s	5.446s	5.228s	1m4.104s

2D RC Frame SSI (5 steps) - Sequential			
Local Run	Container Run		
cml04	cml04	cml05	Windows laptop
2m1.976s	2m9.312s	2m3.802s	2m25.113s

2D RC Frame SSI (5 steps) - Parallel (4)			
Local Run	Container Run		
cml04	cml04	cml05	Windows laptop
40.262s	46.318s	44.406s	1m3.608s

2D RC Frame SSI (20 steps) - Sequential			
Local Run	Container Run		
cml04	cml04	cml05	Windows laptop
4m0.460s	4m11.787s	4m7.283s	4m26.416s

2D RC Frame SSI (20 steps) - Parallel (4)			
Local Run	Container Run		
cml04	cml04	cml05	Windows laptop
1m21.863s	1m31.035s	1m29.528s	2m5.348s

Figure 210.1: Comparison of Real-ESSI performance on local Linux machine and Linux/Windows containers.

page 1221 in Jeremić et al. (1989-2025). Installation of post-processing modules is described in Chapter 208, on page 1287 in Jeremić et al. (1989-2025).

Both pre and post processing manuals are also available through the main Real-ESSI Simulator web site: http://real-essi.info/.

210.6.1 Student Manual for Real-ESSI Simulator System Install

Students at ETH, Mr. Max Sieber and Mr. Antonio Felipe Salazar created a manual for installation of the Real-ESSI Simulator system on virtual machine computers. The manual is available HERE.

Chapter 211

Cloud Computing

(2017-2018-2019-2021-2023)

(In collaboration with Dr. Yuan Feng, Prof. Han Yang, and Dr. Hexiang Wang)

211.1 Chapter Summary and Highlights

Described in this chapter are details of accessing and using Real-ESSI Simulator using remote computers, the so called "cloud" computational resources. Current focus is on using Amazon Web Services (AWS) computers.

211.2 Real-ESSI Cloud Computing Overview

Cloud computing refers to the accessing and computing over the Internet rather than on local computers. Cloud computing is a model for enabling on-demand access to a shared pool of configurable computing resources, which can be setup and released rapidly.[1]

Using Real-ESSI Cloud Service, users can get computing instances on demand without requiring a lot of maintenance and financial resources a common, local parallel computer, cluster would require. In addition, users do not need to go through the installation of the dependent libraries, source-code compilation and the installation of other related software, for example preprocessing and post-processing environment. The complete Real-ESSI Simulator system is pre-configured and built within the image such that Real-ESSI Simulator system is portable over the cloud. A stable, release version of Real-ESSI is built and can be used anywhere and anytime.

There are two ways to obtain a Real-ESSI image on Amazon Web Services (AWS):

- Obtain a Real-ESSI private image from Prof. Boris Jeremić, see Section 211.3.1 on page 1376.
- Use a public image of Real-ESSI on AWS marketplace, as described in Section 211.3.2 on page 1388.

After a Real-ESSI image is launched, a Real-ESSI EC2 instance is generated on AWS. The instance can be accessed through a X2GO client. The procedures are written in Section 211.4 on page 1388.

When the simulation on the Real-ESSi instance is finished and all the output result files are fetched, remember to terminate the running instance so that AWS would not keep charging you. Section 211.5 on page 1391 describes how to terminate a running Real-ESSI instance. See Section 211.8 on page 1394 for more information about the cost of AWS cloud computing services.

211.2.1 Real-ESSI Cloud Service Content

One image is built for a single-machine setup, which contains

[1] This is an excerpt from Jeremić et al. (1989-2025)

- Ubuntu 16.04 LTS Desktop and X2GO Server
- Real-ESSI sequential program
- Real-ESSI parallel program
- Real-ESSI 3C seismic motion developments (SW4)
- Real-ESSI pre-processing (gmESSI)
- Real-ESSI post-processing (PVESSIReader)
- Real-ESSI Editor, Sublime plug-ins
- Real-ESSI Documentation
- Real-ESSI Examples

211.3 Launch Real-ESSI Instance on AWS

A Real-ESSI instance can be launched either from the private image with authorization of Prof. Boris Jeremić or from the public image on AWS market place.

211.3.1 Launch Real-ESSI Instance from AWS Private Images

Follow the steps below to launch instances from Real-ESSI Private Image.

1. Create an AWS account.

   AWS is the most widely used cloud service provider. If you do not have one, creating an AWS account is easy. You can create an AWS account through their website https://aws.amazon.com/. After you login, you can see the services on AWS `Console Home` as follows.

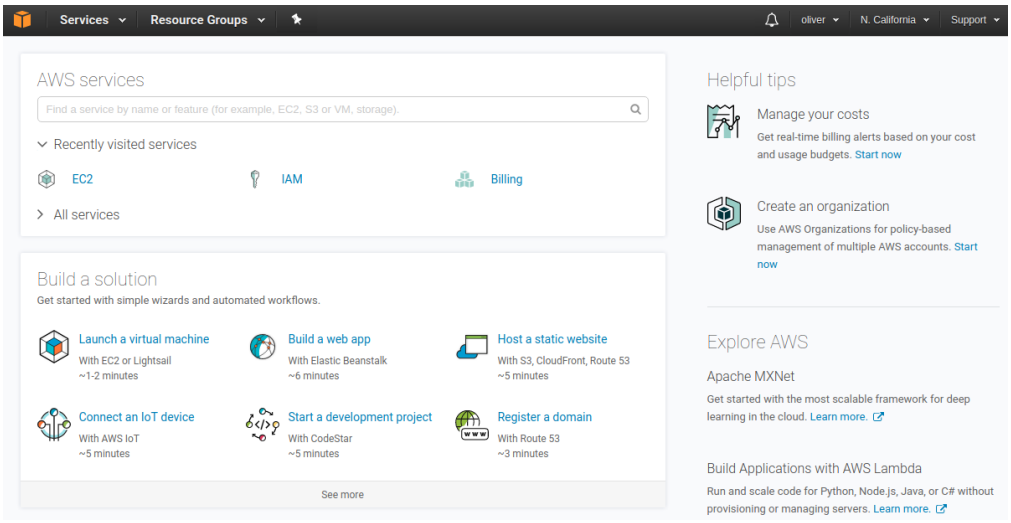


Figure 211.1: AWS Console Home.

2. Request the Real-ESSI image.

   Real-ESSI image is currently a private Amazon Machine Images (AMI). After you get the 12-digit AWS account ID, email the AWS account ID to Prof. Boris Jeremić to obtain the Real-ESSI image. From AWS Console Home, go to `Services` → `EC2`

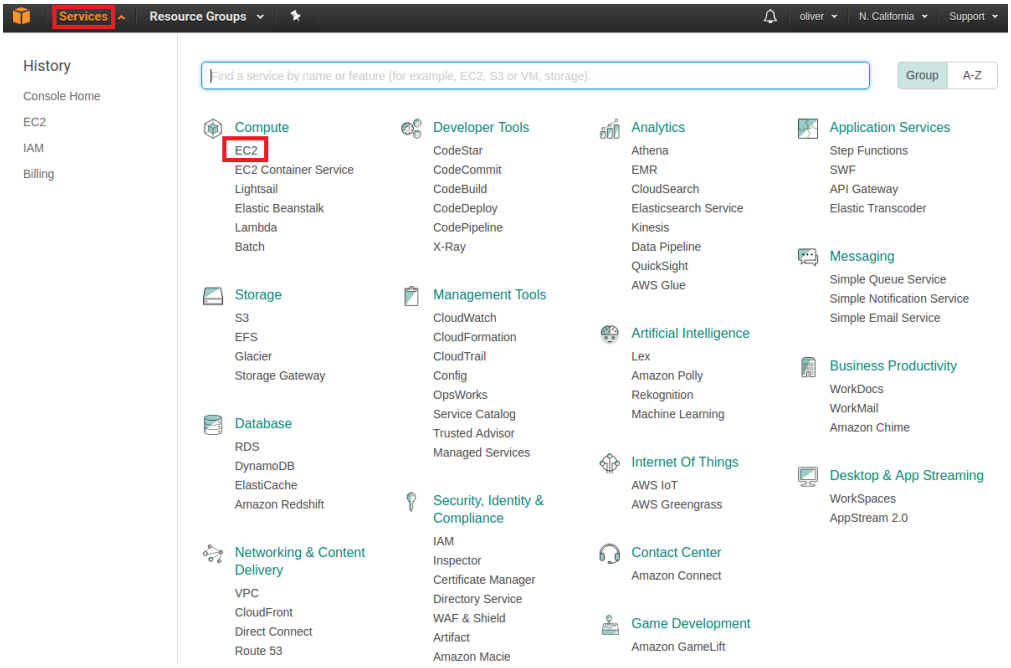


Figure 211.2: AWS Services.

From EC2 Dashboard, go to AMIs to check the Real-ESSI image.

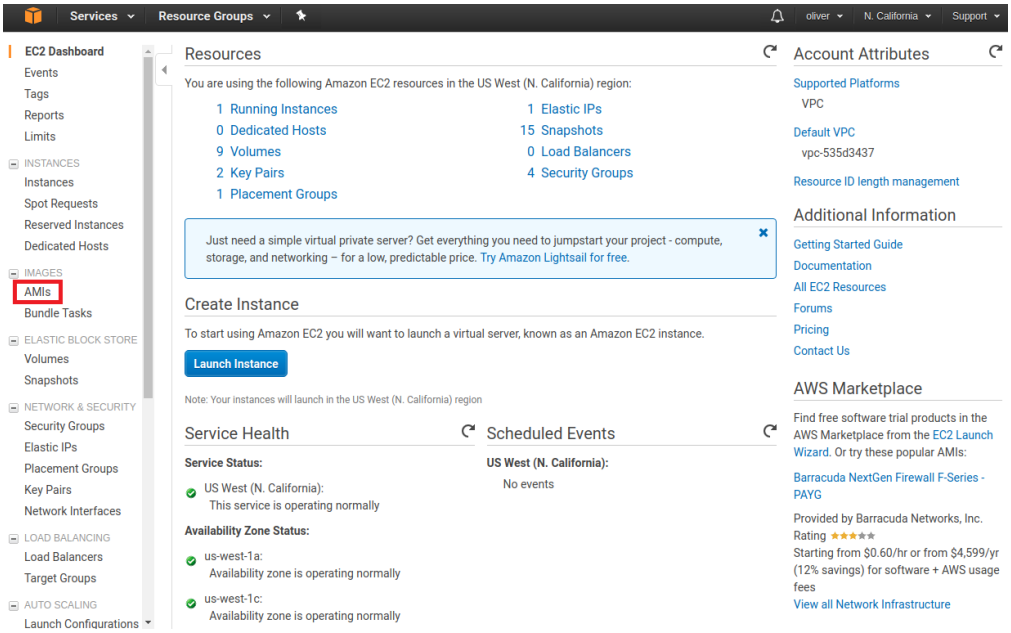


Figure 211.3: AWS EC2 Dashboard AMIs.

If users cannot find the Real-ESSI image, please make sure you are in the same AWS region with Prof. Boris Jeremić, the `region` is shown in the top-right corner on EC2 dashboard. The current Real-ESSI AMIs region are in both `North California` and `Oregon`.

3. Launch the Real-ESSI image.

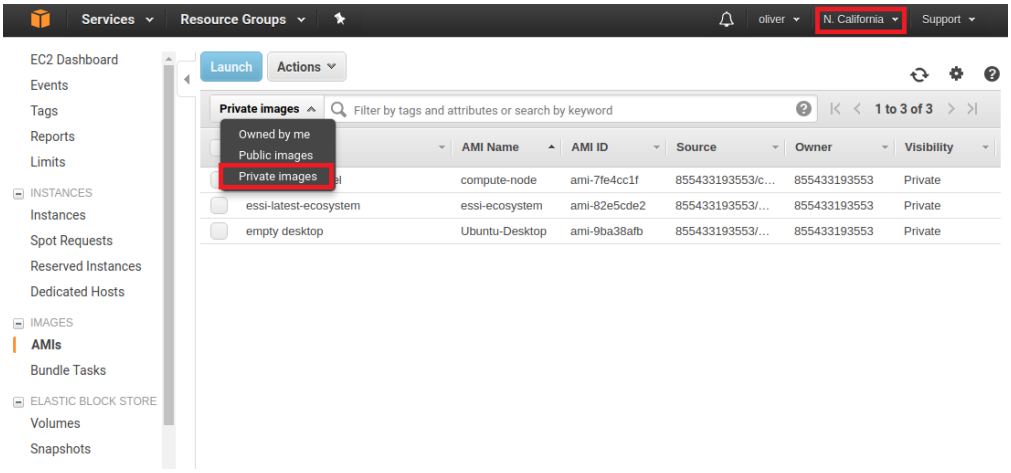


Figure 211.4: AWS EC2 Private AMIs.

Follow the steps below to launch instances from the Real-ESSI image.

(a) Choose AMI.

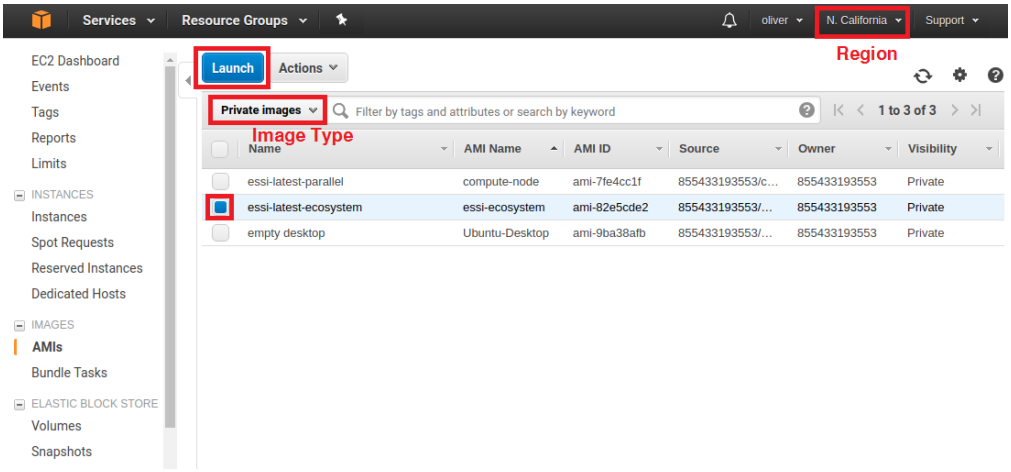


Figure 211.5: EC2 Launch Steps: Choose AMI.

(b) Choose Instance Type

From AMIs, users can launch any number and type of instances and choose the desired EC2 configurations. In order to have the best experiences, the compute-optimized instances (C4, C5 as the latest one, as of early 2019) are recommended.

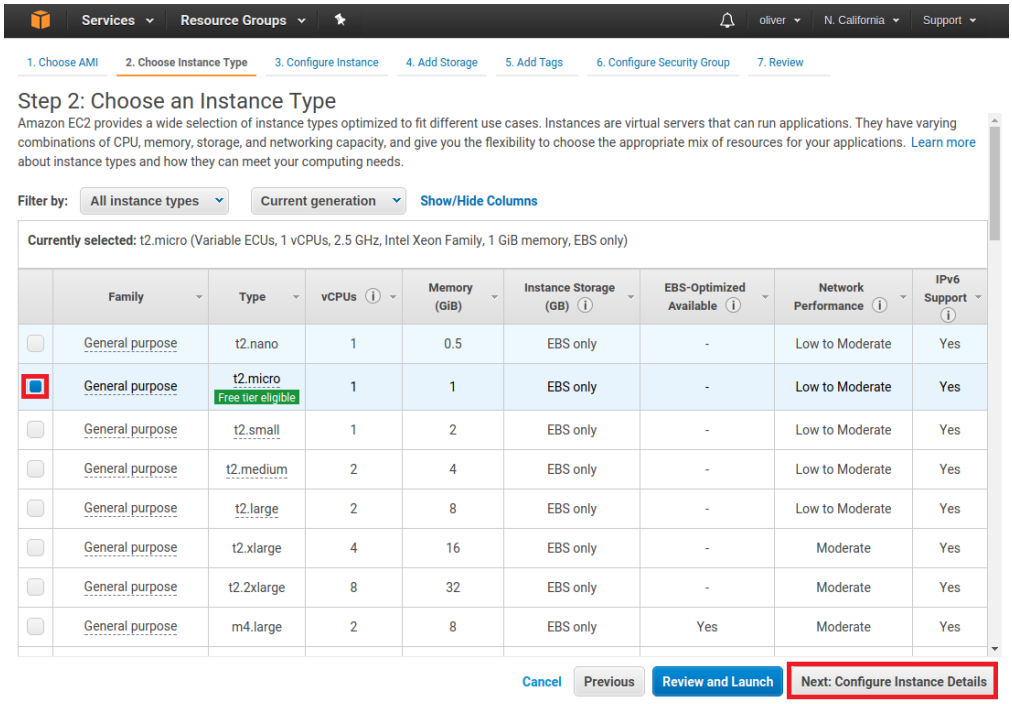


Figure 211.6: EC2 Launch Steps: Choose Instance Type.

(c) Configure Instance

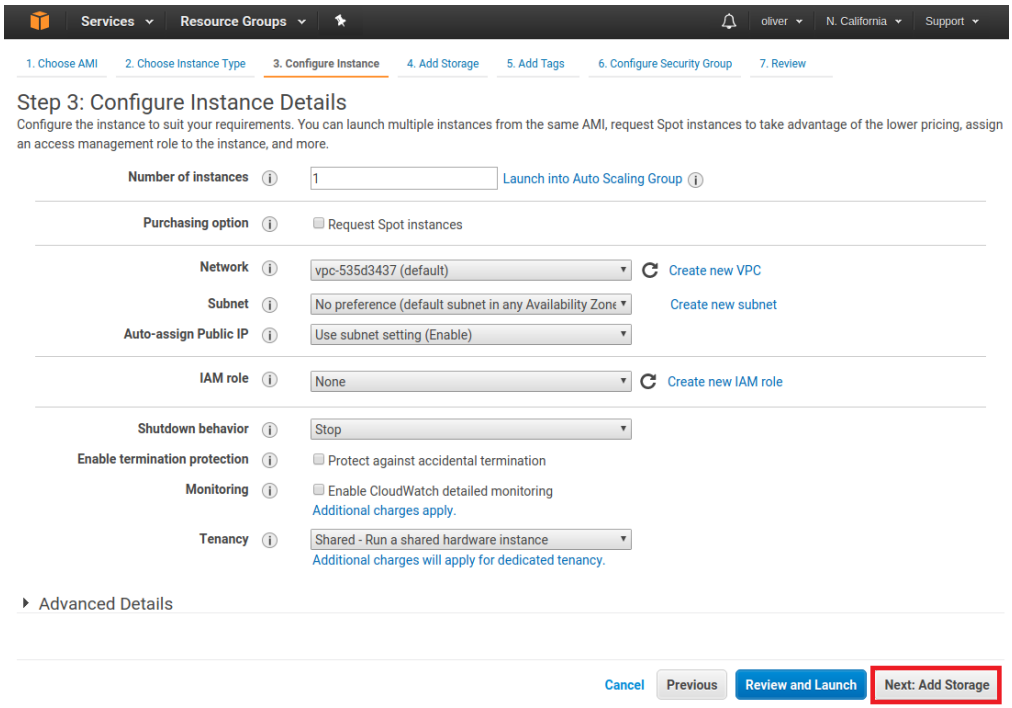


Figure 211.7: EC2 Launch Steps: Configure Instance.

(d) Add Storage

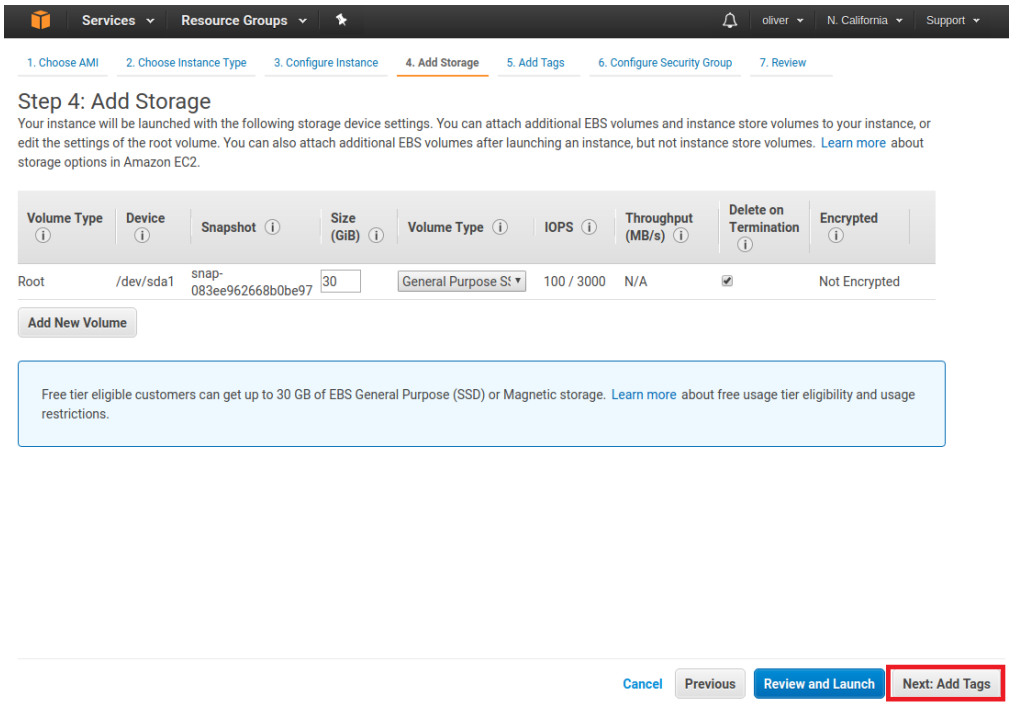


Figure 211.8: EC2 Launch Steps: Add Storage.

(e) Add Tags

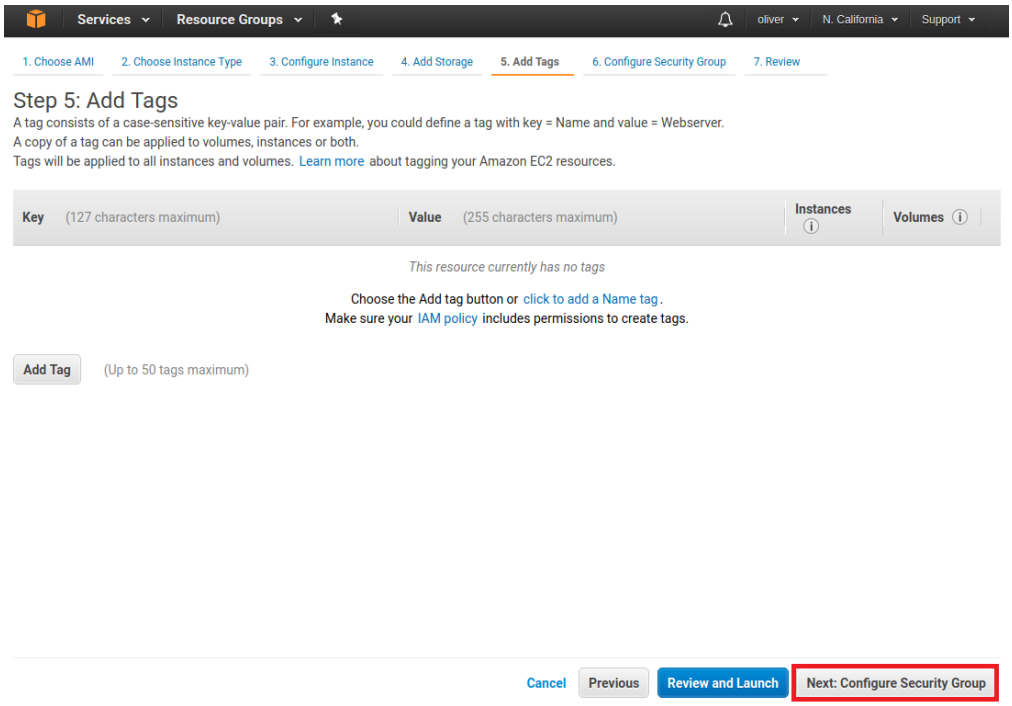


Figure 211.9: EC2 Launch Steps: Add Tags.

(f) Configure Security Group.

Please keep the default Security Group setting.

(g) Review

You may be asked to create a key-pair for later access of the instance you created. The key-pair can be reused later if you created other instances. Besides, the key-pair is portable across other machines. Last but not least, the key-pair cannot be recreated after you launch the instance, so please make sure you save the key-pair in a safe place.

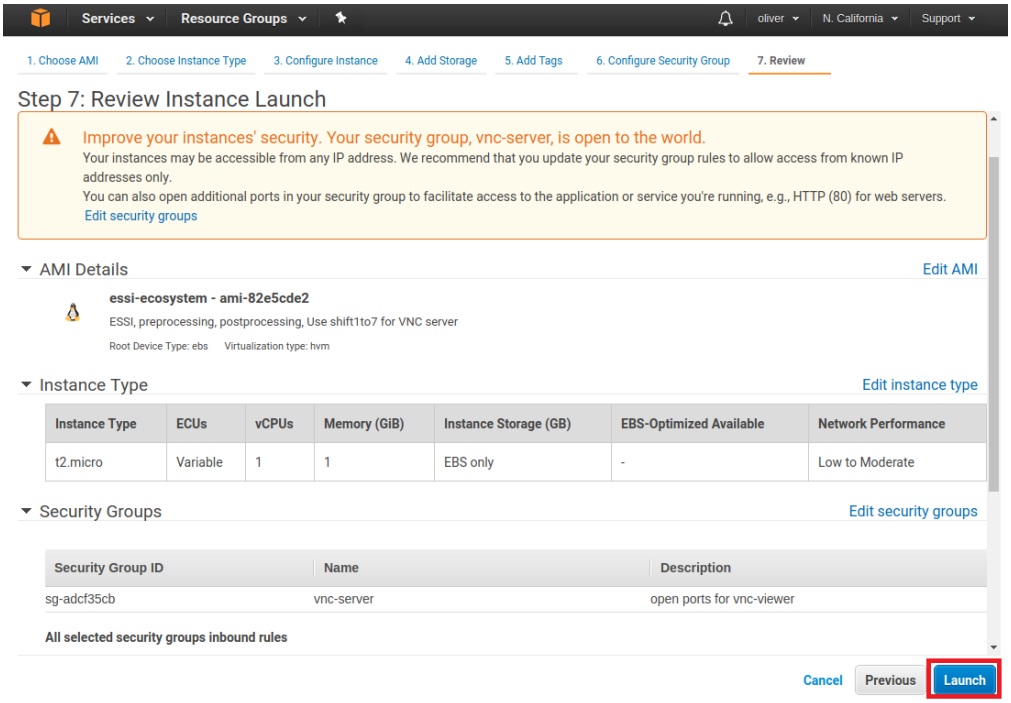


Figure 211.10: EC2 Launch Steps: Review.

4. Check the launched instances

   After the launch, you can view the running instance through `EC2 Dashboard` → `Instances`

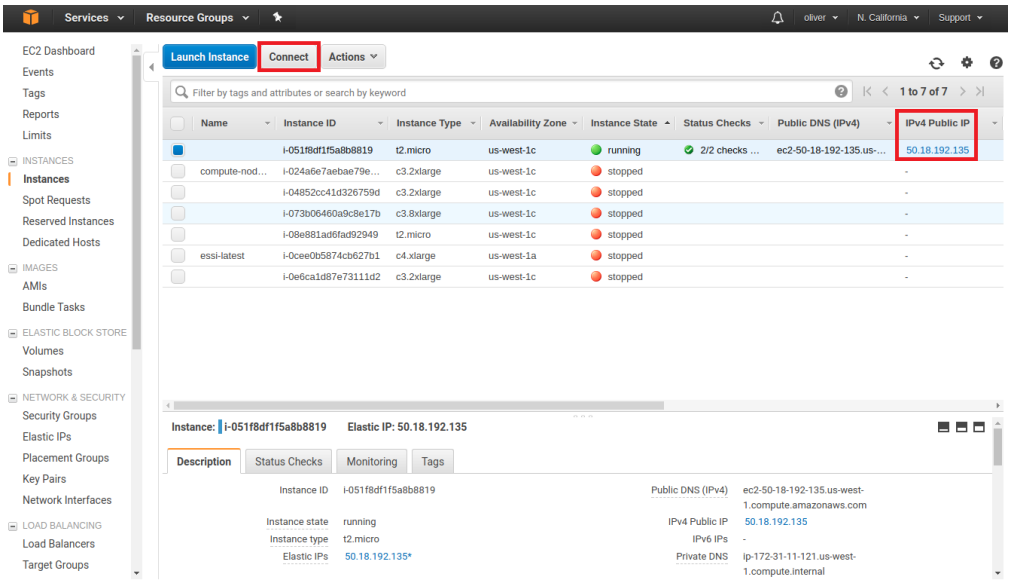


Figure 211.11: EC2 Running Instances.

You can login to your instances either by `ssh` or by using X2GO client 211.4. Please note that every time when you restart the instances, the public IP address will change.

5. Fix Public IP Address (Optional)

   The public IP address of Real-ESSI instances change for each reboot. If users want to have a fixed public IP address for every login, users can allocate one elastic IP address and associate the IP address to a Real-ESSI instance such that users can have a fixed public IP address for each login.

6. Attach more Storage (Optional)

   The Real-ESSI Image holds 30GB Hard disk and already uses 15GB. In the case of a real large simulations, this size hard drive might not be enough for the full output. Users can attach more storage through elastic block store.

211.3.2 Launch Real-ESSI Instance from AWS Market Place

This section gives a quick start guide for using Real-ESSI on AWS market place.

Real-ESSI Simulator system (pre processing, main Real-ESSI program, post processing) is available on Amazon Web Services MarketPlace. Point your web browser to the Amazon Web Services Market Place, and search for "Real ESSI", "Real-ESSI" or "MS ESSI".

In summary, a quick guide to launching an instance from AWS Market Place is:

- Go to the ESSI Cloud Product Page.
- Click Continue to go to Launch ESSI from the Cloud.
- Click Manual Launch (use 1-Click Launch, if comfortable with settings).
- Click Launch from the EC2 Console for your preferred region.
- Select your preferred instance from the table, e.g. t2.micro.
- Click Review and Launch.

211.4 Connect to Real-ESSI Instance on AWS

211.4.1 Install X2GO Client

Before connecting to the Real-ESSI cloud, users should install the client-side of X2GO. X2Go is a remote desktop software that can visualize the launched Real-ESSI instance. Installation of X2GO for different operating systems is fairly straightforward, and users can find installation instructions on their own or follow installation instructions below.

211.4.1.1 Installing X2GO client on Ubuntu Linux

User can directly install X2GO client by using debian install utility, to install x2goclient.

211.4.1.2 Installing X2GO client on Apple Mac

Users can download the package through this link: http://code.x2go.org/releases/X2GoClient_latest_macosx_10_9.dmg.

211.4.1.3 Installing X2GO client on Windows

Users can download the package through this link: http://code.x2go.org/releases/X2GoClient_latest_mswin32-setup.exe.

211.4.1.4 Installing X2GO client on other operating systems

If you are using a different operating system, please refer to X2GO website for the installation. The X2GO website for client installation is https://wiki.x2go.org/doku.php/download:start

211.4.2 Configure the Client-Side of X2GO

For all operating systems, users will see the same session when they open the x2goclient new-session, as shown in Fig. 211.12.

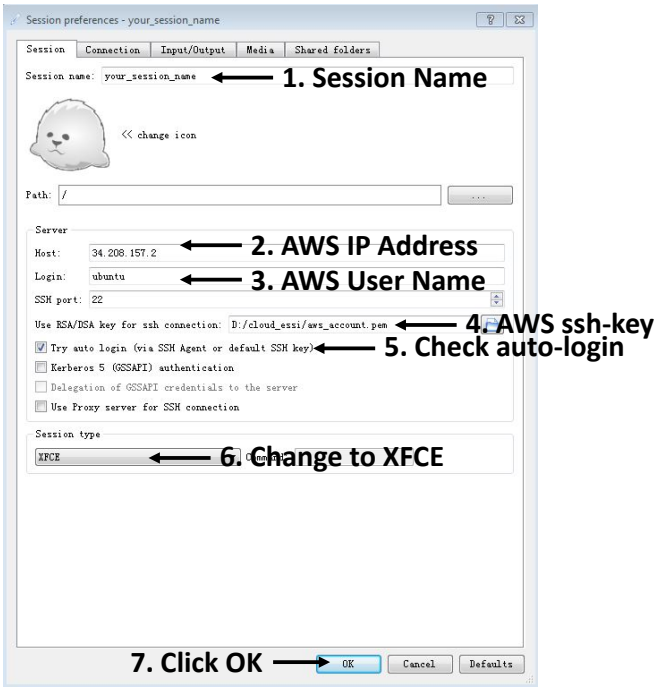


Figure 211.12: Configuration of X2GO client.

1. Users can name their own session.
2. AWS IP address is to be copied from EC2 management console, from the description TAB of launched instance, at the bottom of the page. This is IPv4 Public IP... it goes into Host: ...
3. AWS User Name is "ubuntu".
4. AWS ssh-key is the one saved from before, in .ssh directory
5. Please check the auto-login.
6. Please change the session type to XFCE.
7. Click OK to finish the configuration.

In addition to the Desktop login, users can also use `ssh` to login the Real-ESSI Terminal.

```
chmod 400 your_ssh_key.pem
ssh -i your_ssh_key.pem ubuntu@your_AWS_public_IP_address
```

211.4.3 Connect to the Launched Instance

Click the configured session to connect to the ESSI instance. You should see a virtual desktop pop up on your local machine, as shown in Fig. 211.13. Now you have successfully connected to the Real-ESSI Simulator instance on AWS. You can now use Real-ESSI Simulator within the virtual desktop.

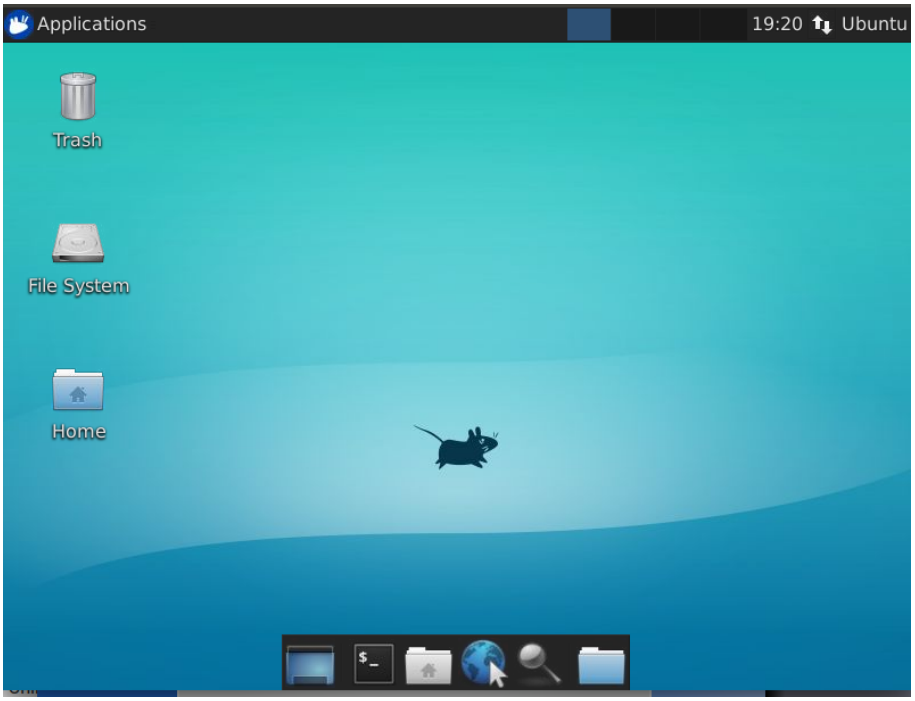


Figure 211.13: Connected to the already launched Real-ESSI instance.

211.5 Terminate Real-ESSI Instance on AWS

Once the Real-ESSI simulation on AWS is finished, the user can transfer output files to the local local computer, or leave them on AWS, preferably on cheap S3 storage Section 211.8 on page 1394 provides detailed description of storage and transfer options and costs. NOTE: Users need to terminate the running Real -ESSI instance on AWS to avoid additional charges. The terminate operation is done on AWS console that is the same place where you launch the Real-ESSI instance. As shown Fig. 211.14, following steps are required:

1. Click 'Instances' from the sidebar to see all your running instances on AWS.
2. Choose the instance you want to terminate.
3. Click 'Actions'.

4. Click 'Instance State'

5. Click 'Terminate'

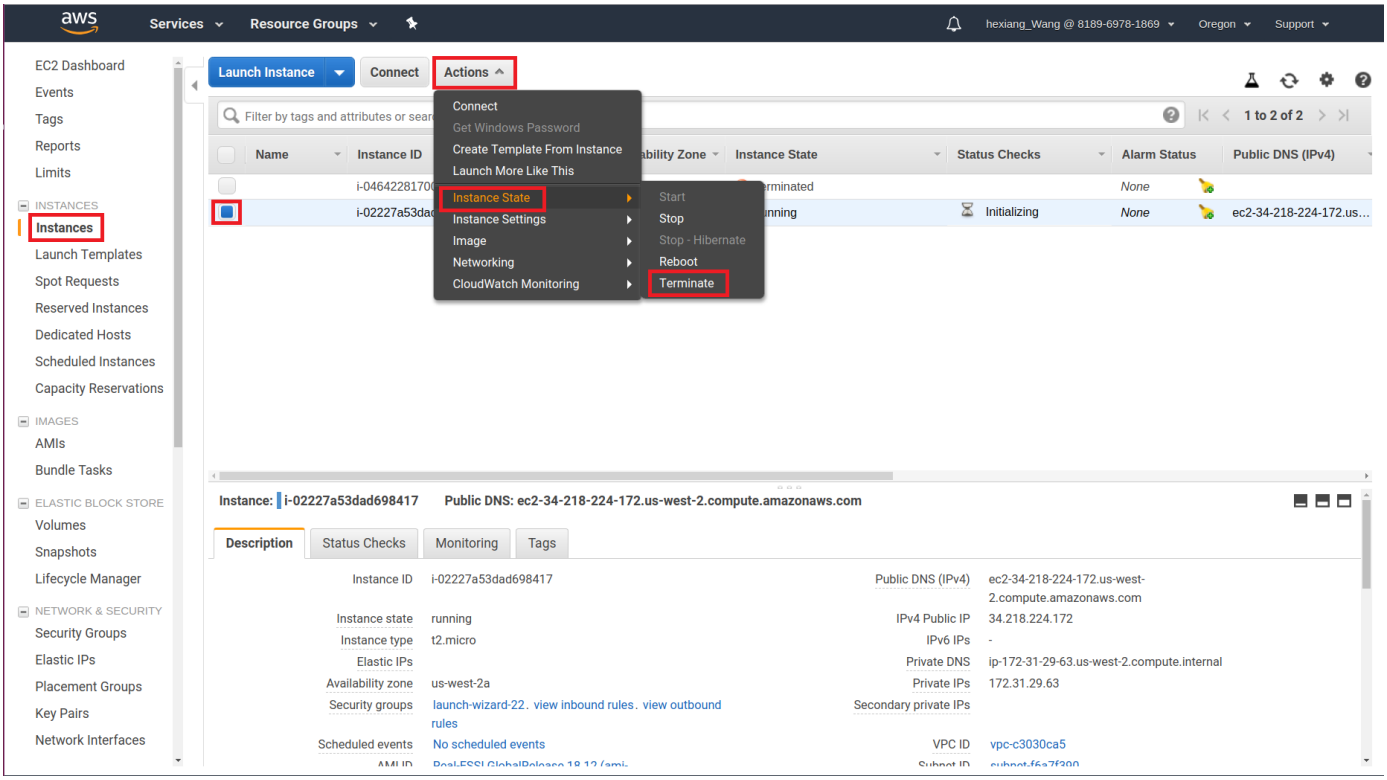


Figure 211.14: Terminate a Real-ESSI Simulator instance.

211.6 Adding Permission for Private Real-ESSI Image to User AWS Accounts

login to AWS

sign in to console

go to image in a region, say N, California

then go to EC2

go to AMIs on left side

select image to be shared

go to Actions

go to Modify Image Permissons and put user account number then click Add Permission and then Save...

211.7 Real-ESSI Instructional Videos Cloud Computing

This section presents few short instructional videos about how to use Real ESSI on Amazon Web Services (AWS) computers.

211.7.1 Installing X2GO for Windows

Youtube instructional video.

211.7.2 Installing X2GO for Macintosh

Youtube instructional video.

211.7.3 Installing X2GO for Linux

Youtube instructional video.

211.7.4 Launch AWS Marketplace

Youtube instructional video.

211.7.5 Access Running Instance on AWS

Youtube instructional video.

211.7.6 Start Real-ESSI Program on AWS

Youtube instructional video.

211.7.7 Run Real-ESSI Example Model on AWS

Youtube instructional video.

211.7.8 Visualize Real-ESSI Example Model on AWS

Youtube instructional video.

211.7.9 Post-Process, Visualize Real-ESSI Results on AWS

Youtube instructional video.

211.8 Cost of AWS EC2

The cost breakdown for using Real-ESSI on AWS (EC2) is:

- AWS computer cost

  There are 3 ways to pay for AWS computer cost (EC2 instances)

  - On-Demand instance, offers a real, instant pay-per-use model. On-Demand instance is sold at a fixed price, and AWS computer availability is guaranteed (within the limits of the service-level agreement). Running Real-ESSI On-Demand Instance: User prepares simulation runs, and then can simulate problems at hand immediately.
  - Spot instance, uses spare AWS computers that users can bid for. Prices for those spot instances fluctuate based on the supply and demand of available AWS computers. When a user makes a bid for a Spot instance, a spot instance is launched when the bid exceeds the current Spot market price, and continues until terminated by the user. The user is charged the Spot market price, not the bid price while the instance runs. Spot instances can offer substantial savings over On-Demand instances, as shown in the AWS Spot Bid Advisor. Running Real-ESSI using Spot instance: User can prepare simulation runs, and then bid on computer hardware and run simulations at later time, when cost is acceptable.
  - Reserved instance, uses spare AWS computers during scheduled, later time as determined by AWS and reserved by the user. Running Real-ESSI using Reserved Instance: User prepares simulation runs, and then reserves AWS computer to simulate problem at hand at predetermined/reserved time.

- AWS data storage cost

  Input data/files and output data/files are stored using:

  - Amazon Elastic Block Store (EBS), attached to a AWS computer (EC2 instance) during simulation run. Storage cost is charged by the size of storage in GB per month, pro-rated to the hour, until the storage is released. The cost of EBS is typically $\$0.10$ per GB per month. When running Real-ESSI program on AWS computer, the storage is used during simulation, while the data (input and output) is transferred out of the AWS computer, to other type of storage that is less expensive (the so called S3 storage, see below), or to user's desktop computer, before AWS computer/instance is terminated and storage released.

  - Amazon Simple Storage Service (S3), offers better value for longer term data storage. S3 pricing varies by region and frequency of access. Cost of S3 storage is typically between $\$0.0125$ are $\$0.03$ per GB per month.
  - Amazon Glacier, provides storage at an even lower cost of $\$0.007$ per GB per month for data archiving.

- AWS data transfer cost

  Data transfer charges are listed as part of the On-Demand EC2 pricing. Transfer is typically charged at $\$0.09$ per GB beyond the first 1GB of data and up to the first 1TB of a given month. After the first TB, price drops down.

- Real-ESSI program cost

  Use of Real-ESSI for educational purposes is free. For commercial use of Real-ESSI, please contact Prof. Jeremić or one of the commercial companies that offer access to Real-ESSI on AWS.

211.8.1 Cost of Running Real-ESSI on AWS

211.8.1.1 Small Size Real-ESSI Example

Imposed Motion Real-ESSI modeling and simulation on AWS summary:

- DOFs in the Model: 5,000
- Number of Time Step: 210
- Running Time: 30 Second
- Disk Space: 25 MB.
- Recommended Machine: Free Instance Amazon EC2 t2.micro

The Real-ESSI input files for this example are available HERE. The compressed package of input files is HERE.

The Modeling parameters are listed below

- Elastic Material Properties
  - Mass density, ρ, 2000 kg/m^3
  - Shear wave velocity, V_s, 500 m/s
  - Young's modulus, E, 1.1 GPa
  - Poisson's ratio, ν, 0.1

The thickness of the shell structure is 2 meters. The simulation model is shown below.

The simulation results:

The time series of simulation results is shown in Fig. 410.22.

The response spectrum of motion is shown in Fig. 410.23.

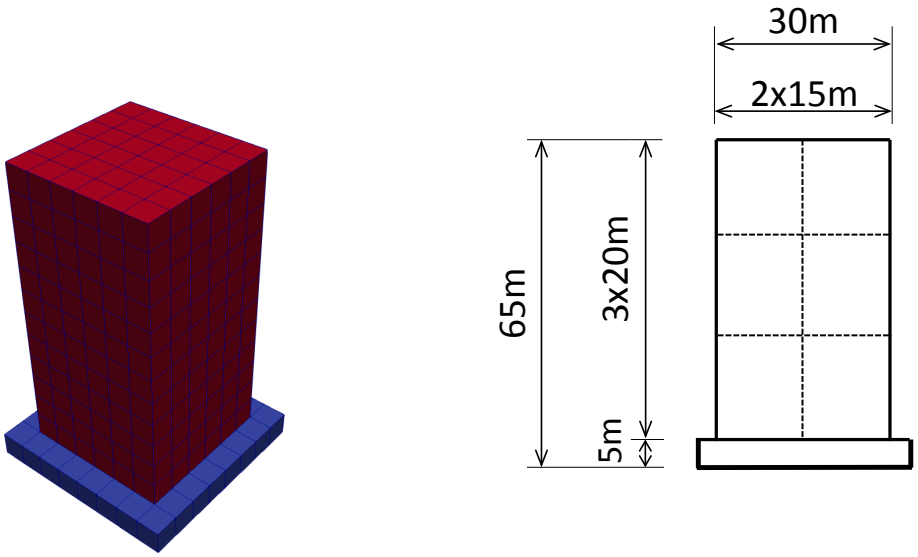


Figure 211.15: Simulation Model.

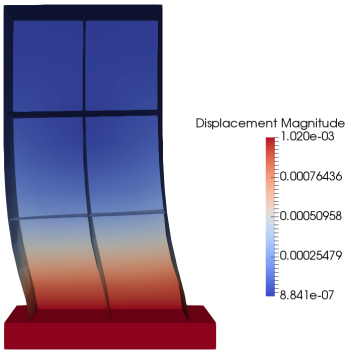


Figure 211.16: Simulation Results.

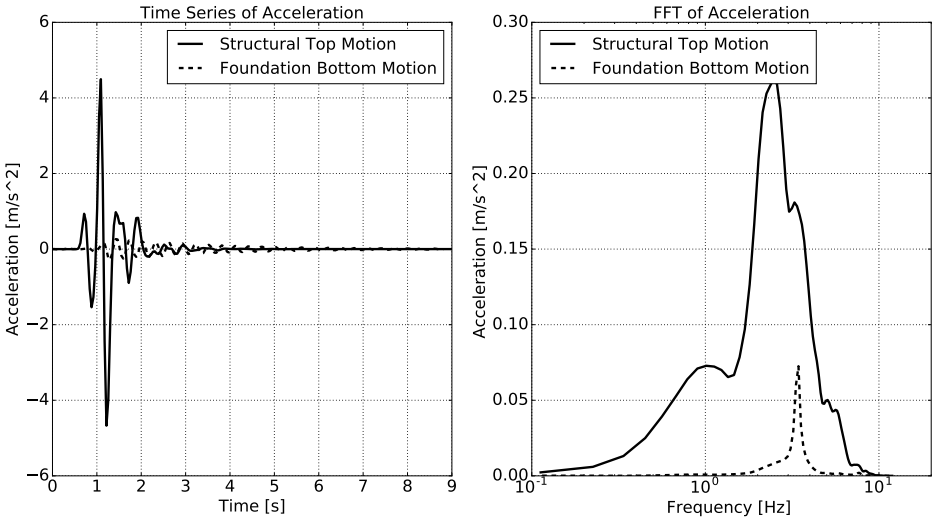


Figure 211.17: Simulation Results: Acceleration Time Series with 1C imposed motion.

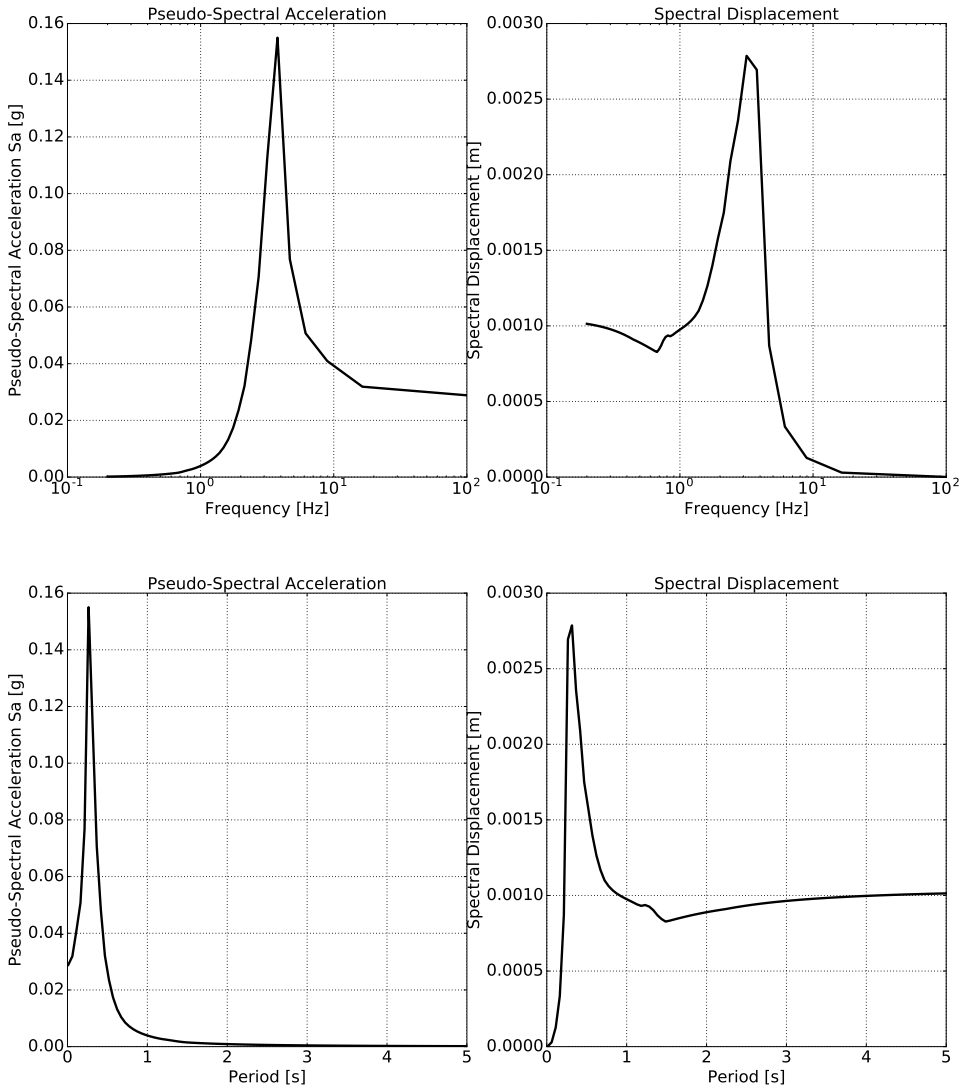


Figure 211.18: Simulation Results: Response Spectrum of Structure Top with 1C imposed motion.

Eigen Analysis Real-ESSI modeling and simulation on AWS summary:

- DOFs in the Model: 5,000
- Number of Eigenmodes: 10
- Running Time: 3 Second
- Disk Space: 25 MB.
- Recommended Machine: Free Instance Amazon EC2 t2.micro

The Real-ESSI input files for this example are available HERE. The compressed package of input files is HERE.

The thickness of the shell structure is 2 meters. The simulation model is shown below.

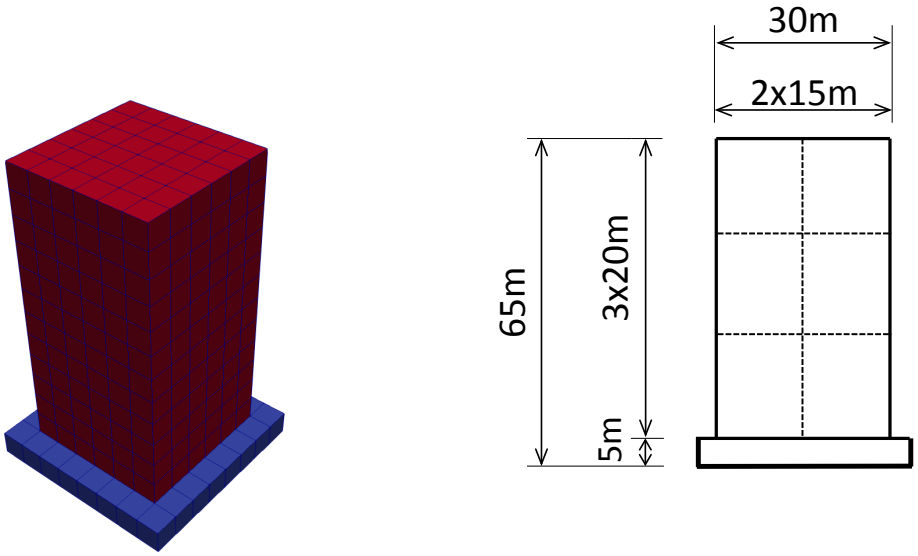


Figure 211.19: Simulation Model.

The eigen results:

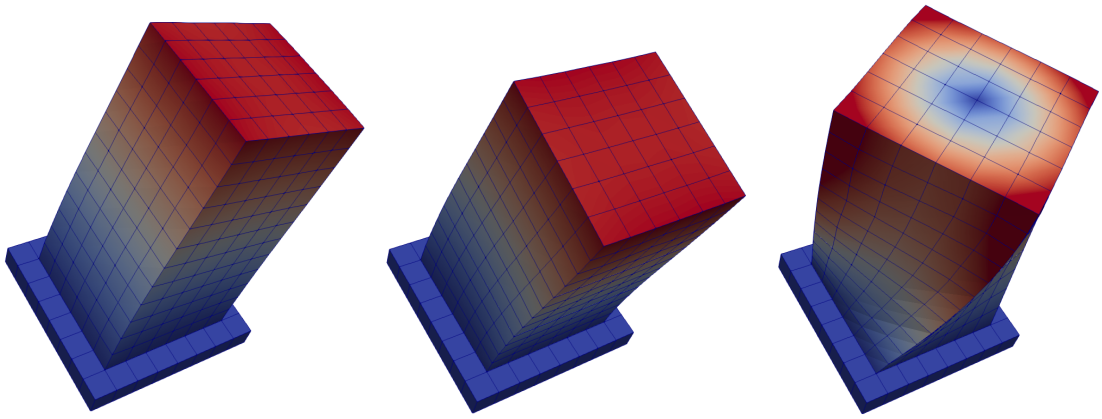


Figure 211.20: Eigen Results (Eigen Mode 1 to 3 from left to right).

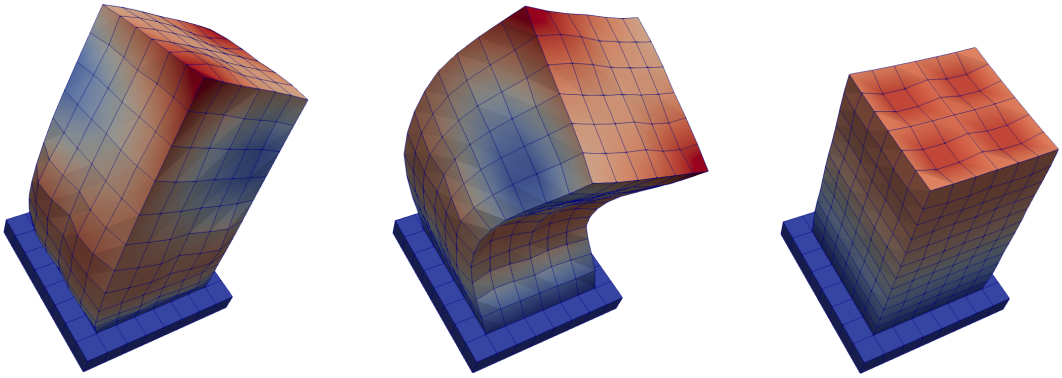


Figure 211.21: Eigen Results (Eigen Mode 4 to 6 from left to right).

211.8.1.2 Medium Size Real-ESSI Example

Elastic Material The compressed package of input files is available HERE.

Real-ESSI modeling and simulation on AWS summary:

- DOFs in the Model: 132,000
- Number of Time Steps: 210
- Running Time: 10 minutes
- Disk Space: 3GB
- Recommended Machine: Amazon EC2 c4.2xlarge instance 8 cores.
- Estimated Bill in AWS Region Oregon/Ohio/Northern Virginia:
  - For simulation time: $\$0.398 * 10/60 = \$0.07$
  - For General Purpose (SSD) Storage: $\$0.1 * 3 = \$0.3$ (monthly)
  - For S3 Storage: $\$0.023 * 3 = \$0.069$ (monthly)

The Modeling parameters are listed below

- Elastic Material Properties
  - Mass density, ρ, 2000 kg/m^3
  - Shear wave velocity, V_s, 500 m/s
  - Young's modulus, E, 1.1 GPa
  - Poisson's ratio, ν, 0.1

The illustration results of the simulation is shown in Fig. 410.12. It is noted that outside the DRM layer, there are no outgoing waves.

von-Mises Armstrong-Frederick Material The compressed package of input files is available HERE.

Real-ESSI modeling and simulation on AWS summary:

- DOFs in the Model: 132,000
- Number of Time Steps: 210
- Running Time: 46 minutes

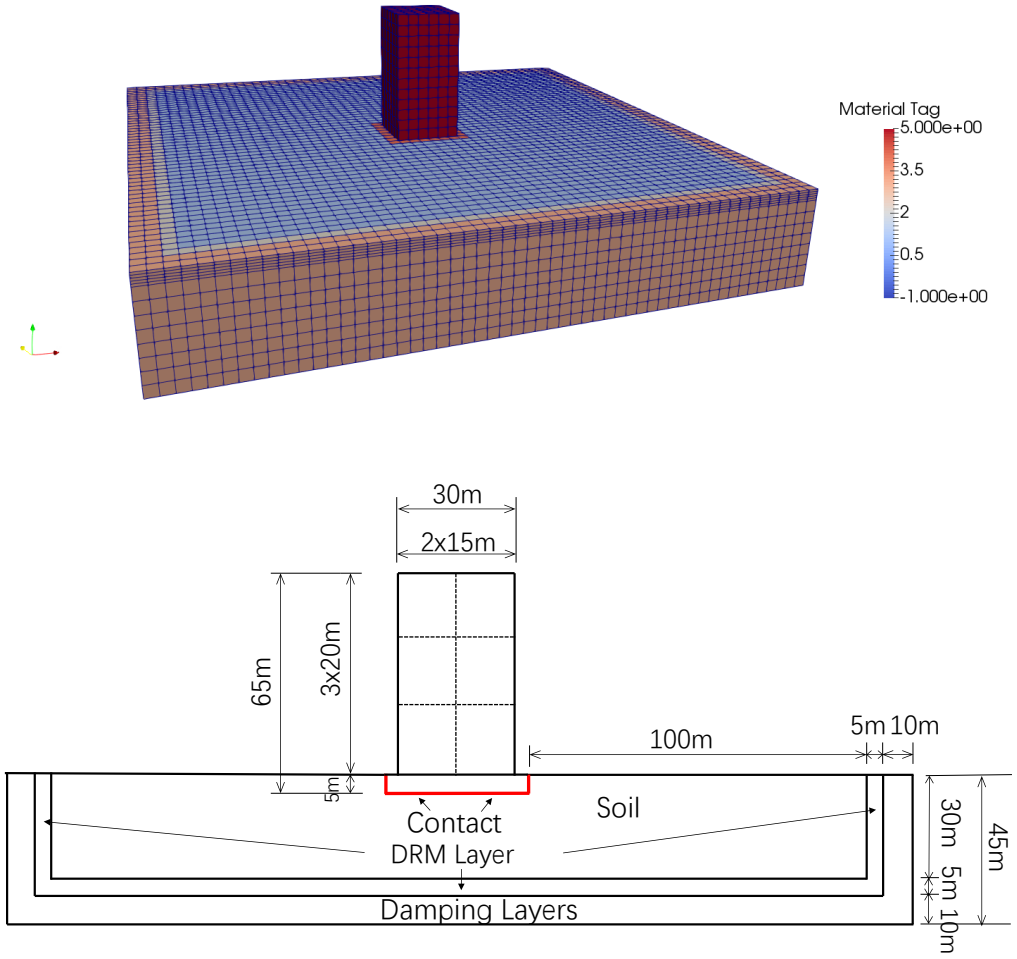


Figure 211.22: Simulation Model.

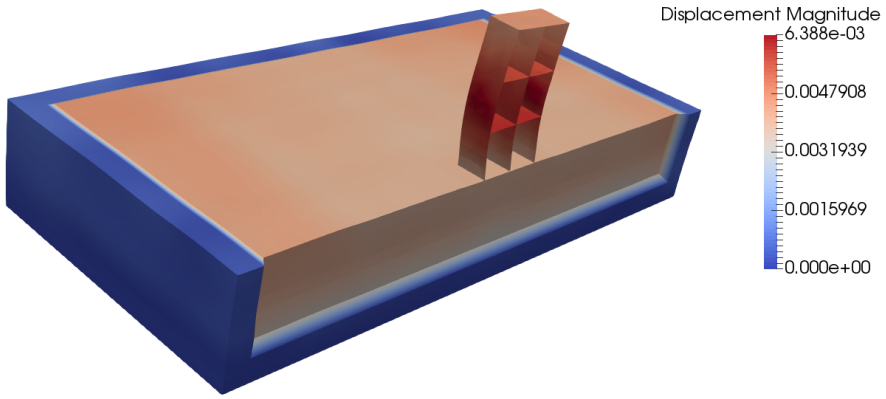


Figure 211.23: Simulation Model.

- Disk Space: 3GB
- Recommended Machine: Amazon EC2 c4.2xlarge instance 8 cores.
- Estimated Bill in AWS Region Oregon/Ohio/Northern Virginia:
  - For simulation time: $\$0.398 * 46/60 = \$0.31$
  - For General Purpose (SSD) Storage: $\$0.1 * 3 = \$0.3$ (monthly)
  - For S3 Storage: $\$0.023 * 3 = \$0.069$ (monthly)

The Modeling parameters are listed below

- von-Mises nonlinear hardening material model
  - Mass density, ρ, 2000 kg/m^3
  - Shear wave velocity, V_s, 500 m/s
  - Young's modulus, E, 1.1 GPa
  - Poisson's ratio, ν, 0.1
  - von Mises radius, k, 60 kPa
  - Nonlinear kinematic hardening, H_a, 30 MPa
  - Nonlinear kinematic hardening, C_r, 60
  - Shear strength ($\approx \sqrt{2/3}\ H_a/C_r$), S_u, 408 kPa
  - Isotropic hardening rate, K_{iso}, 0 Pa

SIMULATION TIME: With 8 cores on AWS EC2 c4.2xlarge instance, the running time for this example is 46 minutes.

211.8.1.3 Large Example

Elastic Simulation The Real-ESSI input files for this example are available HERE. The compressed package of Real-ESSI input files for this example is available HERE.

Real-ESSI modeling and simulation on AWS summary:

- DOFs in the Model: 210,000
- Number of Time Steps: 2065
- Running Time: 17 hours
- Disk Space: 45GB
- Recommended Machine: Amazon EC2 c4.8xlarge instance 36 cores.
- Estimated Bill in AWS Region Oregon/Ohio/Northern Virginia:
  - For simulation time: $\$1.591 * 17 = \$27.05$
  - For General Purpose (SSD) Storage: $\$0.1 * 45 = \$4.5$ (monthly)
  - For S3 Storage: $\$0.023 * 45 = \$1.035$ (monthly)
  - For Network Bandwidth if transfer: $\$0.09 * 45 = \$4.05$

SIMULATION TIME: With 32 cores on AWS EC2 c4.8xlarge instance, the running time for this example is 17 hours.

The Modeling parameters are listed below

- Soil
  - Unit weight, γ, 21.4 kPa
  - Shear velocity, Vs, 500 m/s
  - Young's modulus, E, 1.3 GPa
  - Poisson's ratio, ν, 0.25
  - Shear strength, S_u, 650 kPa
  - von Mises radius, k, 60 kPa
  - kinematic hardening, H_a, 30 MPa
  - kinematic hardening, C_r, 25
- Structure

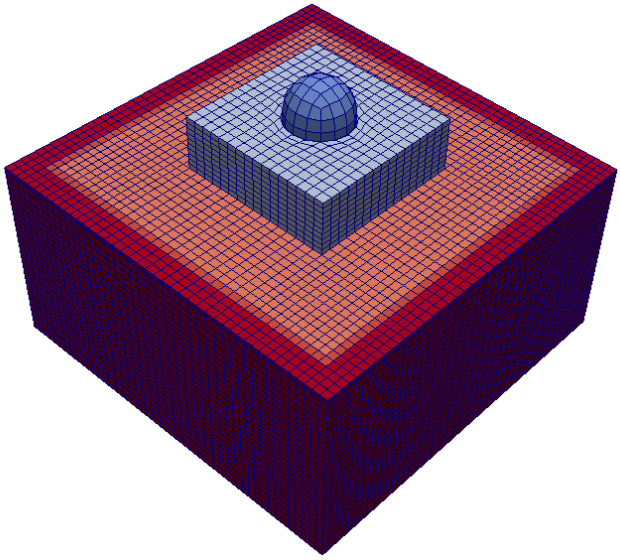


Figure 211.24: Simulation Model.

- Unit weight, γ, 24 kPa
- Young's modulus, E, 20 GPa
- Poisson's ratio, ν, 0.21

The input motion is a 3C wave from SW4.

Inelastic Simulation The Real-ESSI input files for this example are available HERE. The compressed package of Real-ESSI input files for this example is available HERE.

Real-ESSI modeling and simulation on AWS summary:

- DOFs in the Model: 210,000
- Number of Time Steps: 2065
- Running Time: 30 hours
- Disk Space: 45GB
- Recommended Machine: Amazon EC2 c4.8xlarge instance 36 cores.
- Estimated Bill in AWS Region Oregon/Ohio/Northern Virginia:
  - For simulation time: $\$1.591 * 30 = \$47.73$
  - For General Purpose (SSD) Storage: $\$0.1 * 45 = \$4.5$ (monthly)
  - For S3 Storage: $\$0.023 * 45 = \$1.035$ (monthly)
  - For Network Bandwidth if transfer: $\$0.09 * 45 = \$4.05$

SIMULATION TIME: With 32 cores on AWS EC2 c4.8xlarge instance, the running time for this example is 30 hours.

The Modeling parameters are listed below

- Soil
  - Unit weight, γ, 21.4 kPa
  - Shear velocity, Vs, 500 m/s
  - Young's modulus, E, 1.3 GPa
  - Poisson's ratio, ν, 0.25
  - Shear strength, S_u, 650 kPa
  - von Mises radius, k, 60 kPa
  - kinematic hardening, H_a, 30 MPa
  - kinematic hardening, C_r, 25
- Structure

  - Unit weight, γ, 24 kPa
  - Young's modulus, E, 20 GPa
  - Poisson's ratio, ν, 0.21

- Contact
  - Initial axial stiffness, k_n^{init}, 1e9 N/m
  - Stiffening rate, S_r, 1000 /m
  - Maximum axial stiffness, k_n^{max}, 1e12 N/m
  - Shear stiffness, k_t, 1e7 N/m
  - Axial viscous damping, C_n, 100 $N \cdot s/m$
  - Shear viscous damping, C_t, 100 $N \cdot s/m$
  - Friction ratio, μ, 0.25

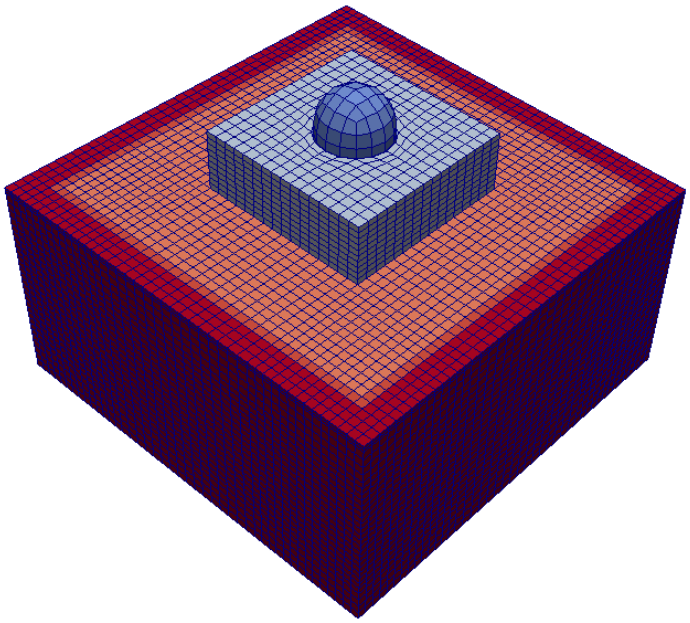


Figure 211.25: Simulation Model.

211.8.2 Real-ESSI AWS Manual, April 2023

Real-ESSI AWS manual developed for the Real-ESSI Short Course, in March, April 2023, is provided below.

1. Summary and Highlights

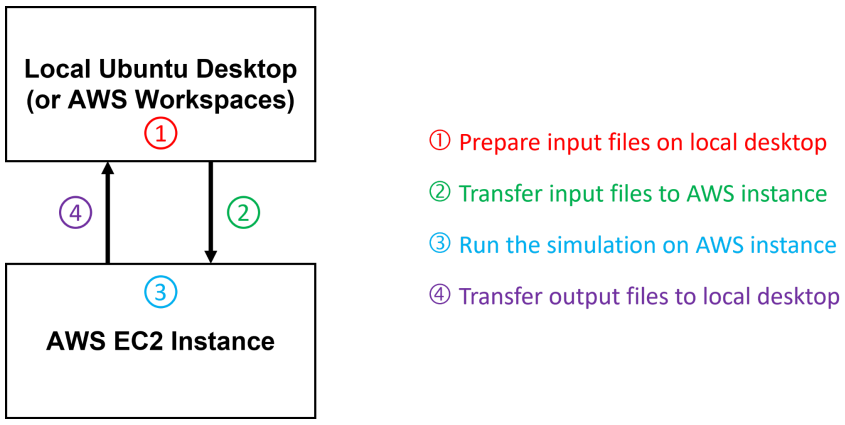
This chapter describes details of accessing and using Real-ESSI Simulator using remote computers, the so-called "cloud" computational resources. The current focus is on using Amazon Web Services (AWS) computers.

Note: If you have a local Ubuntu desktop, you may download and install the Debian package for Real-ESSI. The procedures are documented in Real-ESSI Simulator System Procurement Procedures.

2. Real-ESSI Cloud Computing Overview

Cloud computing refers to the accessing and computing over the Internet rather than on local computers. Cloud computing is a model for enabling on-demand access to a shared pool of configurable computing resources, which can be setup and released rapidly.[1]

Using Real-ESSI Cloud Service, users can get computing instances on demand without requiring a lot of maintenance and financial resources a common, local parallel computer cluster would require. In addition, users do not need to go through the installation of the dependent libraries, source-code compilation and the installation of other related software, for example preprocessing and post-processing environments. The complete Real-ESSI Simulator system is pre-configured and built within the image such that the Real-ESSI Simulator system is portable over the cloud. A stable, release version of Real-ESSI is built and can be used anywhere and anytime.



The suggested workflow for Real-ESSI cloud computing using AWS is shown above. The recommended workflow is a client-server style workflow, the most efficient and economical way to perform cloud computing. First, a local Ubuntu desktop (*AWS Workspaces is a good*

[1] This is an excerpt from Jeremic et al. (1989-2023)

substitute if you don't have a local Ubuntu desktop) is used to prepare input files, receive output files, and post-processing simulation results. Then, an AWS EC2 instance is used to conduct high-performance parallel computation from the Real-ESSI simulation. We will introduce detailed procedures in the following chapters.

3. Create AWS Account

3.1 AWS account types

There are two types of accounts on AWS, the Root user and the IAM user, each with unique credentials.

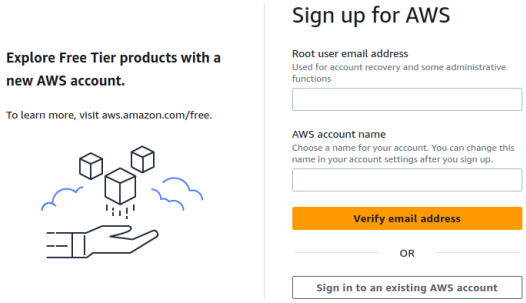
- The root user in Amazon Web Services (AWS) is the initial administrative user created when creating an AWS account. The root user has full access to all AWS services, financials, and resources in the account and can perform any action on them. **It is highly recommended to avoid using the root user account for regular day-to-day operations in AWS due to security reasons.**
- On the other hand, an IAM (Identity and Access Management) user is a user account that is created within your AWS account, separate from the root account. IAM users have a set of permissions that are defined by an AWS administrator (or yourself) to limit what actions they can perform in AWS. This allows you to grant specific permissions to users or groups of users without giving them full access to the AWS account.

IAM users can be created with unique usernames and credentials, and their permissions can be managed separately from the root user. This provides better security and allows you to grant different levels of access to different users or groups, based on their roles and responsibilities.

If you're a first-time user of AWS, your first step is to sign up for a Root user AWS account. When you sign up, AWS creates an AWS account with the details you provide and assigns the account to you. We also suggest activating multi-factor authentication (MFA) for the root user and assigning administrative access to a user. You can find complete documentation on AWS Account Management here.

3.2 Create a root user account

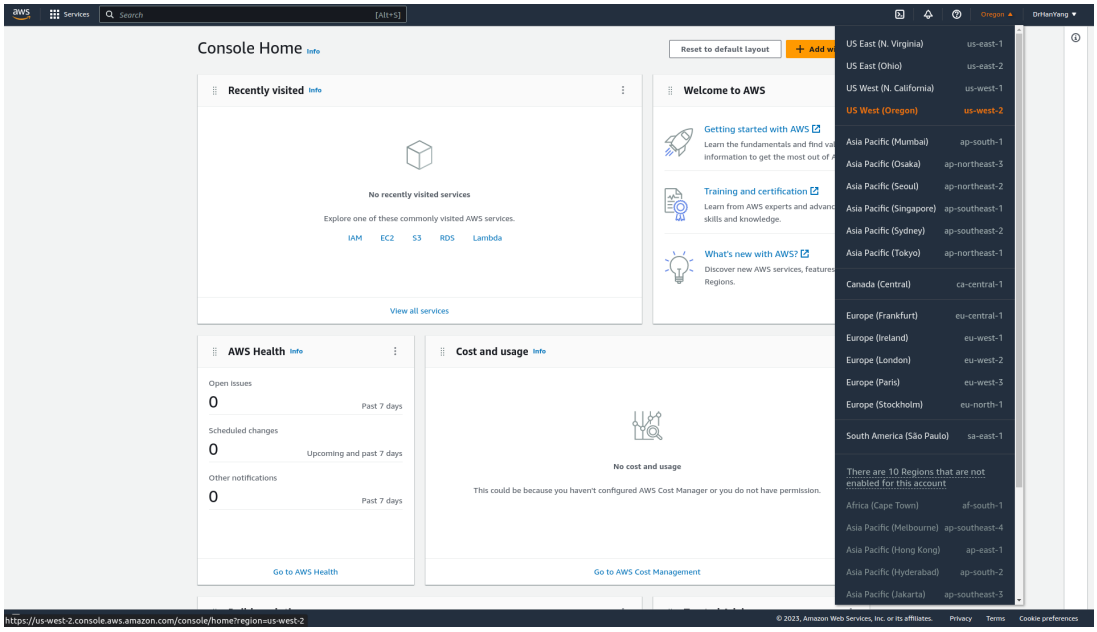
- To create your AWS account, open the AWS home page in your browser and choose **Create an AWS account**.
- Supply your email, then the code sent to your email address.



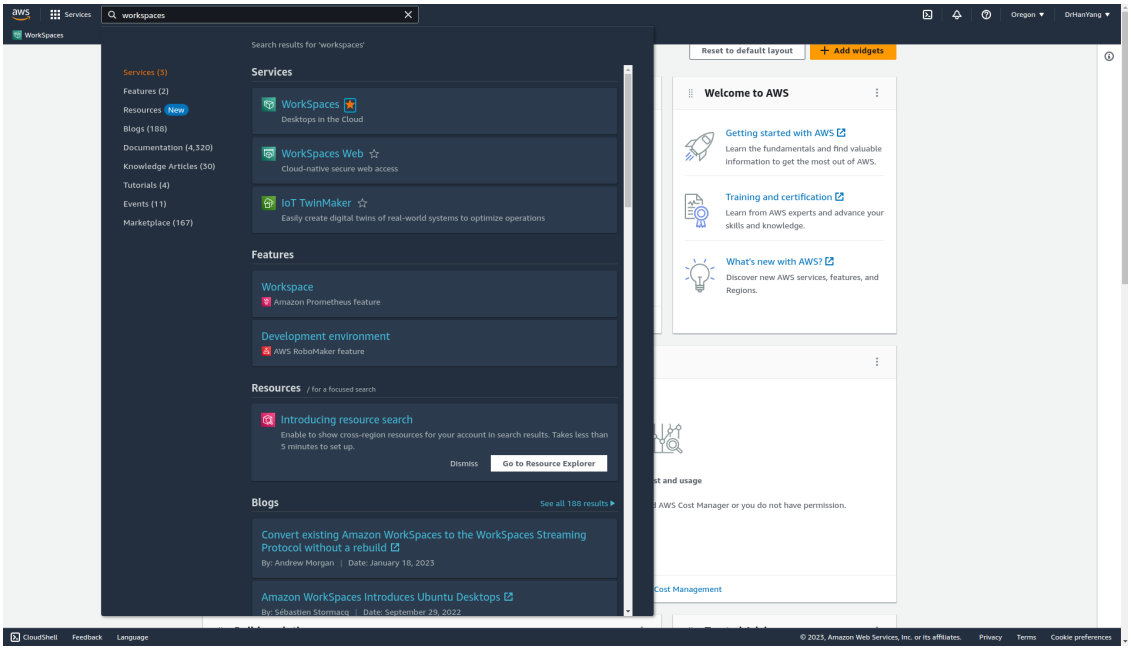
- Create a root user password.
- Provide contact information.
- Provide billing information.
- Confirm your identity.
- Select a support plan. The **Basic support - Free** option is enough for using Real-ESSI.
- Select **Complete sign up**

3.3 Initial setups for your new AWS account

- Sign **into** your root account.
- Switch your region to **US West (Oregon)**. Your region is located near the top right corner of your webpage.



- Search and select services for quick access later. In this manual, we will use several different AWS services. We can find them and add them to favorites. Go to the search bar near the top left of your webpage and search for the services we need. Click on the star symbol next to the service name to add it to your favorites. The service should appear near the top of your webpage for future quick access.



The services we need are **Workspaces** and **EC2**.

4. Amazon Workspaces

If you already have and want to use your own local Ubuntu desktop, skip this chapter.

Summary: This chapter presents setting up, launching, and connecting to your Amazon Workspace. Note that Amazon Workspace is used as a substitute for local Ubuntu desktops.

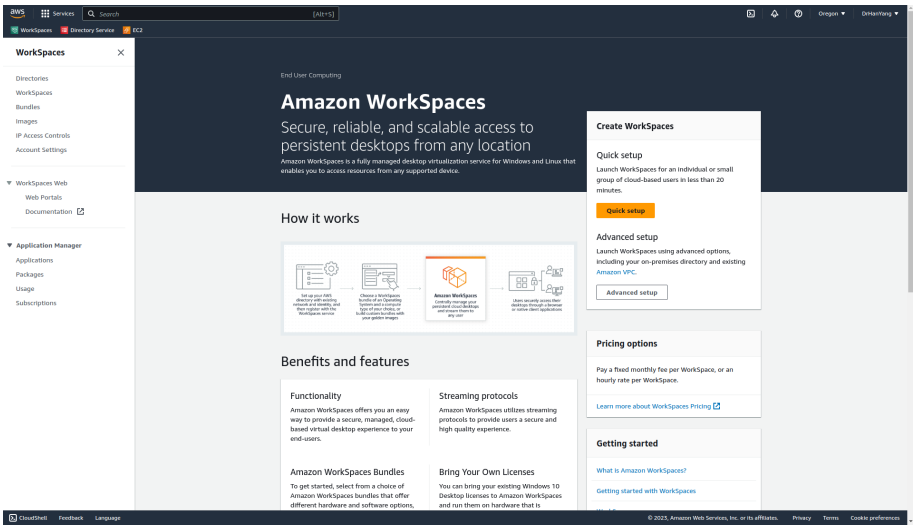
Amazon WorkSpaces enables you to provision virtual, cloud-based Microsoft Windows, Amazon Linux, or Ubuntu Linux desktops, known as *WorkSpaces*. WorkSpaces eliminates the need to purchase and set up your own Ubuntu desktop. Instead, users can access their virtual desktops from multiple devices or web browsers. Complete documentation regarding Amazon Workspaces can be found here.

4.1 WorkSpaces quick setup

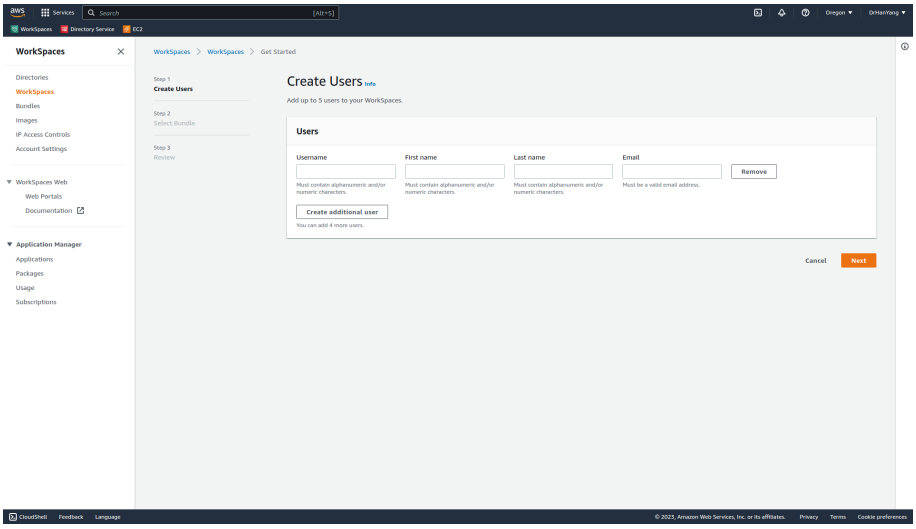
This tutorial uses the **Quick Setup** option to launch your WorkSpace. This option is available only if you have never launched a WorkSpace. Alternatively, for the full documentation see here.

Step 1: Launch the WorkSpace

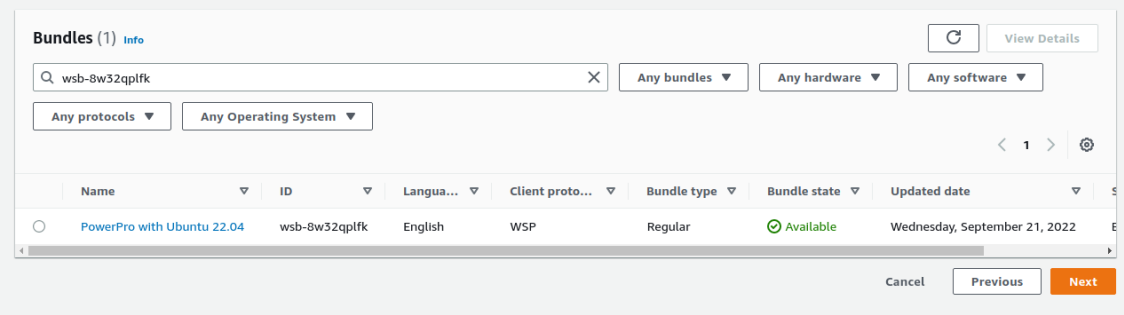
- Open the WorkSpaces console at https://console.aws.amazon.com/workspaces/.
- Choose **Quick setup**. If you don't see this button, either you have already launched a WorkSpace in this Region, or you aren't using one of the Regions that support Quick Setup. In this case, see Launch a virtual desktop using WorkSpaces.



- For **Create Users**, enter the **Username**, **First Name**, **Last Name**, and **Email**. Then choose **Next**. Note that you can enter multiple users here, but this doesn't mean they can use the same Workspace. Instead, multiple Workspaces will be created, one for each user.



- For **Bundles**, select a bundle (hardware and software) for the user with the appropriate protocol (PCoIP or WSP). For Real-ESSI cloud computing, choose PowerPro with Ubuntu 22.04 with the WSP protocol. You can find this bundle by its ID wsb-8w32qplfk.



- Review your **information**. Then choose **Create WorkSpace**.
- It takes approximately 20 minutes, up to 40 minutes, for your WorkSpace to be created. When the launch is complete, the status is AVAILABLE and an invitation is sent to the email address that you specified for each user. If the users don't receive their invitation emails, see Send an invitation email.

Step 2: Connect to the WorkSpace

After you receive the invitation email, you can connect to the WorkSpace using the client of your choice. After you sign in, the client displays the WorkSpace desktop.

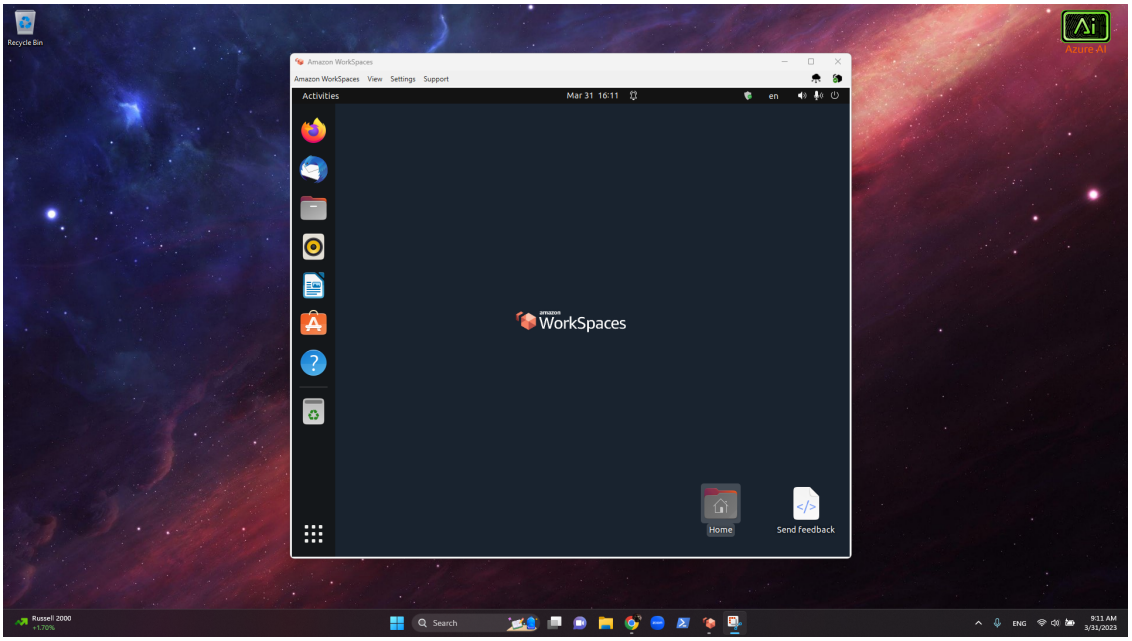
- If you haven't set up credentials for the user already, open the link in the invitation email and follow the directions. Remember the password that you specify as you will need it to connect to your WorkSpace.
- When prompted, download one of the client applications or launch **Web Access**. If you aren't prompted and you haven't installed a client application already, open https://clients.amazonworkspaces.com/ and download one of the client applications or launch **Web Access**.
- Start the client, enter the registration code from the invitation email, and choose **Register**.
- When prompted to sign in, enter the sign-in credentials, and then choose **Sign In**.
- (Optional) When prompted to save your credentials, choose Yes.

Step 3: Clean up (Optional)

If you are finished with the WorkSpace that you created for this tutorial, you can delete it. For more information, see Delete a WorkSpace.

4.2 Install Real-ESSI Debian package on your Workspace

Once you successfully connect to your Workspace, it will be the same as if you are working with a local Ubuntu desktop.



The next step is to install Real-ESSI on your Workspace. Full documentation can be found in Section 1.3 of the Real-ESSI Simulator Procurement Manual. Quick setup steps are summarized below.

Step 1: System libraries update/upgrade

Open a terminal and use the following commands.

```
sudo apt update
sudo apt upgrade
sudo apt autoremove
```

You will be asked to provide a password. The password is the same one you used to connect to your Workspace.

Step 2: Real-ESSI Debian package download

The Real-ESSI program Debian package can be downloaded here. Alternatively, contact Prof. Jeremic to arrange for a customized Real-ESSI Debian package.

Step 3: Real-ESSI Debian package install

Start the Real-ESSI Simulator Debian package install by removing the old installations of Real-ESSI. Then, go to the directory where you have downloaded the Real-ESSI Debian package. Install the Debian package, for example use the following command.

```
sudo apt install ./real-essi_23.01-1_amd64.deb
```

Note that some warning messages might appear but they don't affect the installation. After a successful installation, the sequential and parallel Real-ESSI executables, gmsh/gmESSI preprocessor, paraview/pvESSI post-processor/visualizer, Gmsh, and ParaView are all installed and ready to use.

Step 4: Load pvESSI plugin in ParaView

Start ParaView. Click **Tools**, then **Manage Plugins**. Click **Load New** and find the plugin **PVESSIReader.so** under directory /opt/paraview/lib/paraview-5.10/plugins/PVESSIReader/. Also, check the box **Auto Load** then close ParaView. Next time when ParaView is started, Real-ESSI output files can be visualized and post-processed.

Step 5: Install other useful programs

- **HDFView** can be used to open Real-ESSI output files, which are in HDF5 format.
- **Sublime Text** is the recommended editor for Real-ESSI input files and pre-processing files.

Documentation on how to install these programs can be found in Section 1.3.5 of the Real-ESSI Simulator Procurement Manual.

4.3 Build your model and prepare the input files

Once you have installed Real-ESSI on your Workspace, you are ready to start building your Real-ESSI model. You should finish preparing all the input files on your Workspace before moving on to the next Chapter. Full documentation on Real-ESSI pre-processing and input file formats is available.

Note that if the model is sufficiently small, you can simply run the simulation on your Workspace, without using the AWS Real-ESSI instance which is designed to be used in cases with large models.

5. Launch Real-ESSI Instance on AWS

A Real-ESSI instance can be launched either from the private image with authorization of Prof. Boris Jeremic or from the public image on AWS Marketplace (coming soon). Full documentation regarding launching an instance on AWS can be found here.

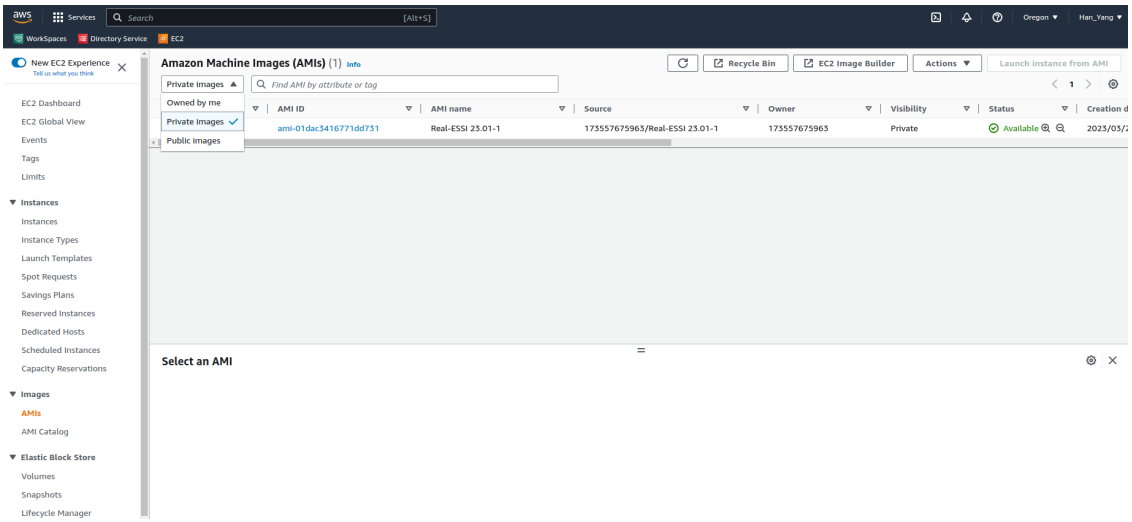
5.1 Launch Real-ESSI instance from AWS private images

Follow the steps below to launch instances from Real-ESSI Private Image.

Step 1: Request the Real-ESSI image

Real-ESSI image is currently a private Amazon Machine Images (AMI). After you get the 12-digit AWS account ID, email the AWS account ID to Prof. Boris Jeremic to obtain the Real-ESSI image.

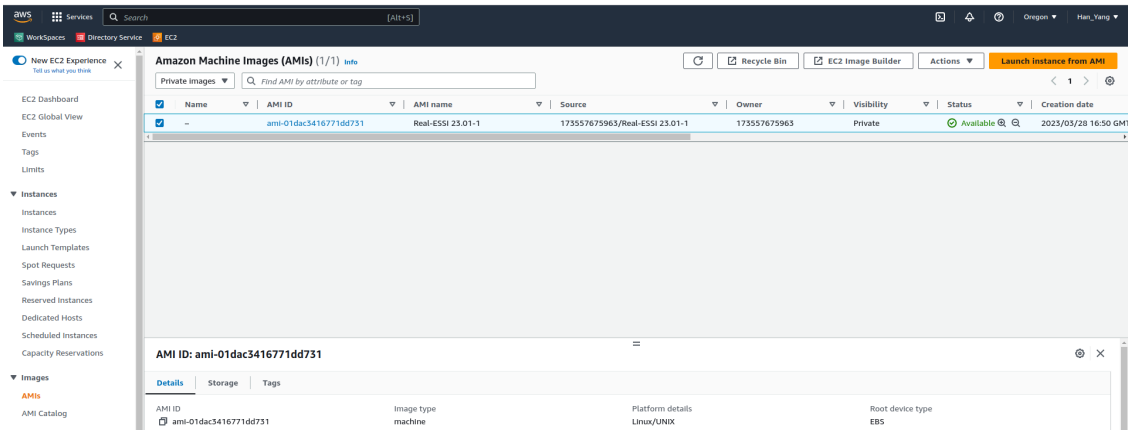
To check if you have access to the Real-ESSI image, open the Amazon EC2 console at https://console.aws.amazon.com/ec2/. Go to **AMIs** and choose **Private images** to see the Real-ESSI image. Currently, the Real-ESSI AMIs are available in the **Oregon** region. The region is shown in the top-right corner on the EC2 dashboard.



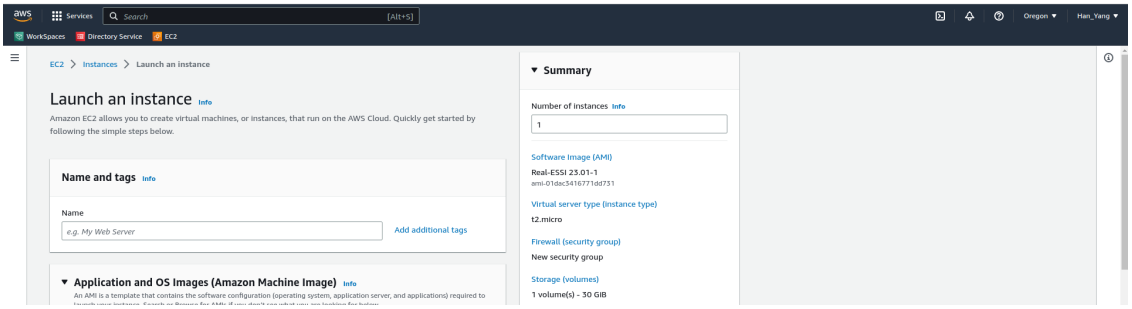
Step 2: Launch the Real-ESSI instance

You can launch the Real-ESSI instance using the AWS Management Console as described in the following procedure. This tutorial is intended to help you quickly launch your first instance, so it doesn't cover all possible options. For information about advanced options, see Launch an instance using the new launch instance wizard. For information about other ways to launch your instance, see Launch your instance.

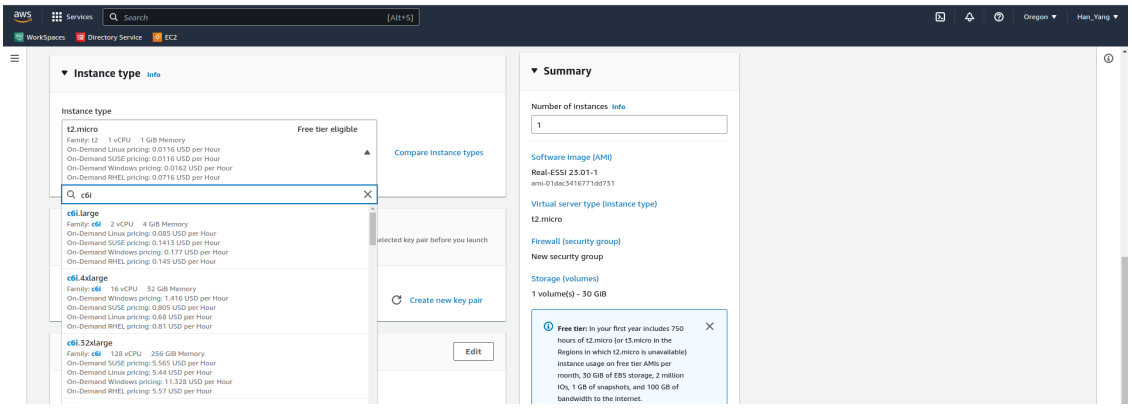
- Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
- From the EC2 console dashboard, Go to **AMIs** and choose **Private images** to see the Real-ESSI image. Choose the image and choose **Launch instance from AMI**.



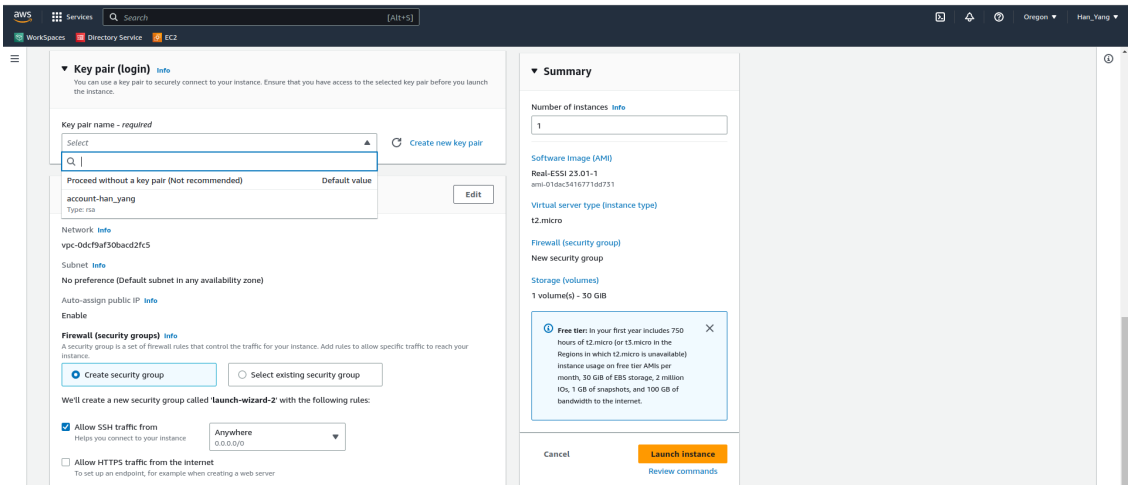
- Under **Name and tags**, for **Name**, enter a descriptive name for your instance.



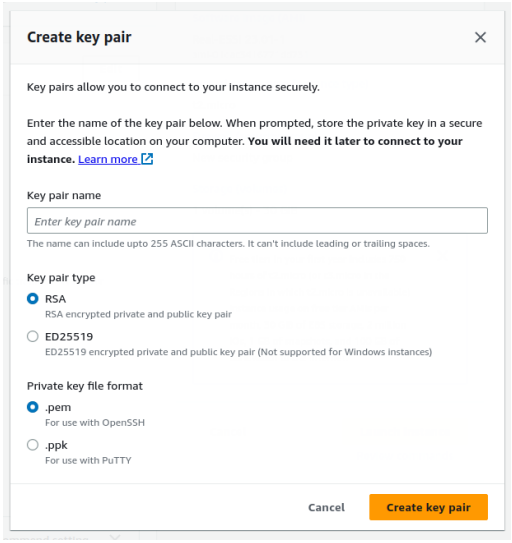
- Under **Instance type**, you can select the hardware configuration for your instance. For Real-ESSI instances, the compute-optimized **c6i** series is recommended. Click the drop-down list under **instance type** and type **c6i** in the search bar. Depending on the size of your model, you can choose an instance type with appropriate computing power.



- Under **Key pair (login)**, select the key pair for your instance.

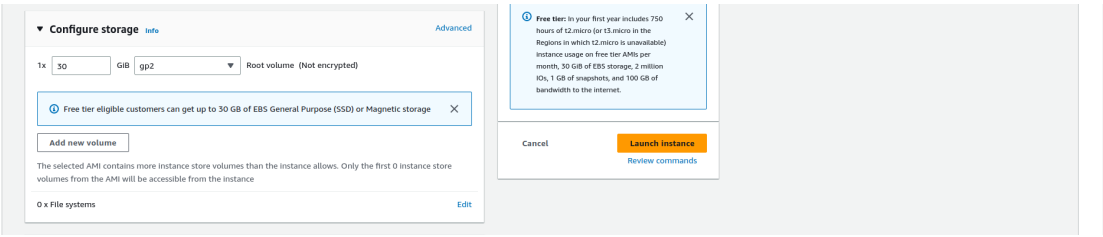


If you have not created a key pair before, choose **Create new key pair**.

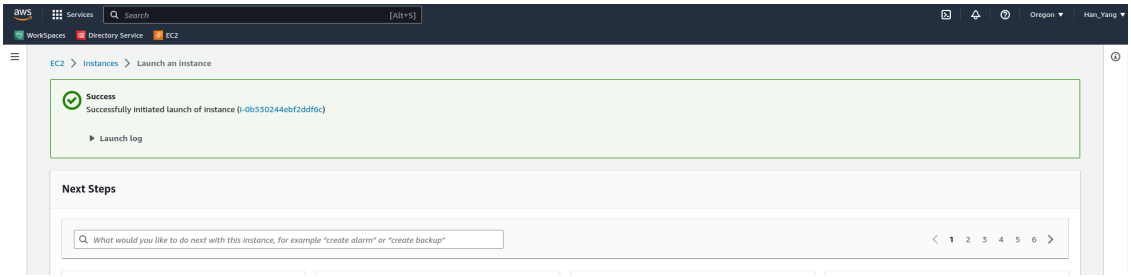


Enter a descriptive name for your key pair. Leave everything else the same. Then click **Create key pair**. You will be prompted to save the key pair. Note that the key pair cannot be recreated after you launch the instance, so please make sure you save it in a safe place. The key pair can be reused later when you launch other instances.

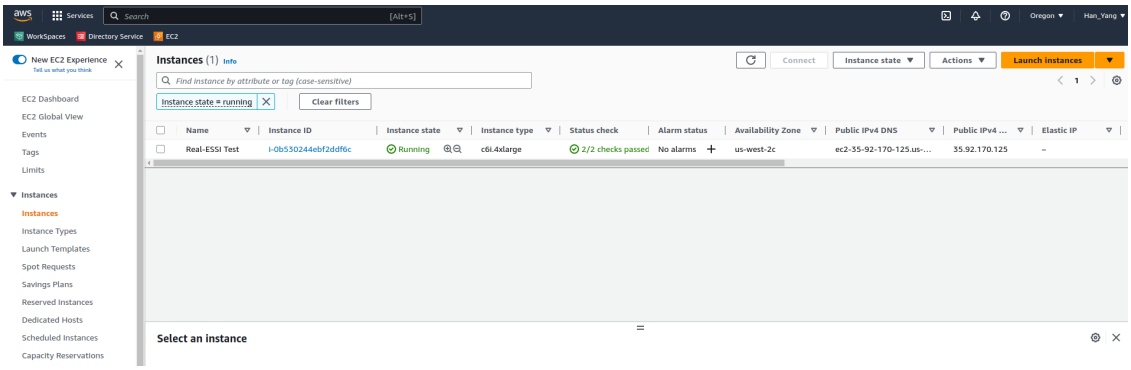
- Under **Configure storage**, change the Root volume depending on the size of your model and simulation options.



- Click **Launch instance**. You should see the message below if the launch is successful.



- You can view your running instances by clicking **Instances** on the left side list of your screen.



5.2 Launch Real-ESSI instance from AWS Marketplace

Coming soon…

6. Connect to Real-ESSI Instance

This chapter provides information about how to connect to a Real-ESSI instance after you have launched it, and how to transfer files between your local computer and your instance. For more information, please refer to the AWS documentation here. To troubleshoot connecting to your instance, see Troubleshoot connecting to your instance.

After you launch your instance, you can connect to it and use it the way that you'd use a computer sitting in front of you. The following instructions explain how to connect to your instance using an SSH client. For more connection options, see Connect to your Linux instance.

6.1 Prerequisites

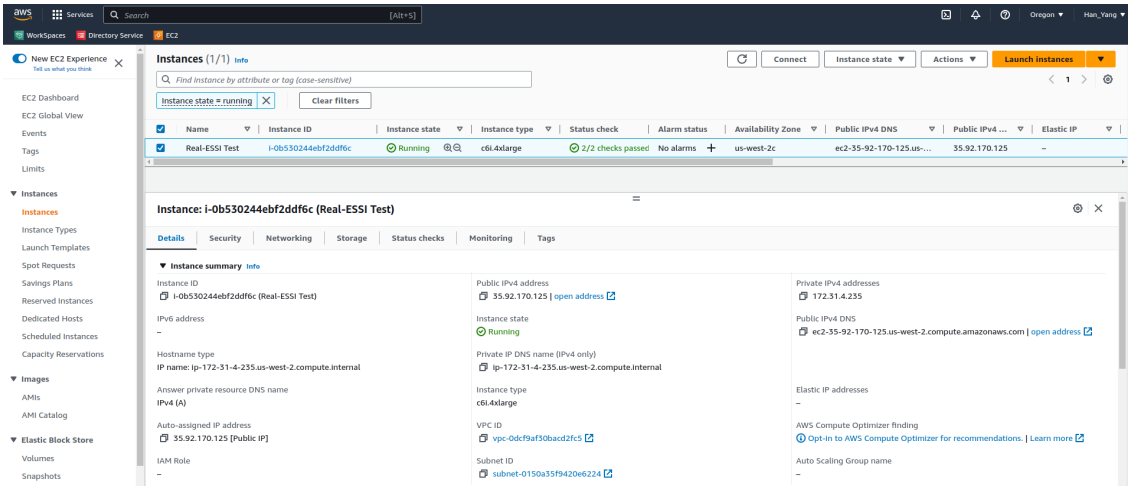
Before you connect to your Linux instance, complete the following prerequisites.

Check your instance status

After you launch an instance, it can take a few minutes for the instance to be ready so that you can connect to it. Check that your instance has passed its status checks. You can view this information in the **Status check** column on the **Instances** page.

Get the public IP address to connect to your instance

Click on your instance to show more information about it. You can find the public IPv4 address from either the summary or details. For example, the public IPv4 address is 35.92.170.125 for the instance shown below.



Locate the private key and set the permissions

You must know the location of your private key file to connect to your instance. For SSH connections, you must set the permissions so that only you can read the file.

Get the fully-qualified path to the location on your computer of the .pem file for the key pair that you specified when you launched the instance.

Use the following command to set the permissions of your private key file so that only you can read it. Replace key-pair-name with the actual name of your key pair.

```
chmod 400 key-pair-name.pem
```

If you do not set these permissions, then you cannot connect to your instance using this key pair. For more information, see Error: Unprotected private key file.

6.2 Connect to your Real-ESSI instance using an SSH client

Use the following procedure to connect to your Linux instance using an SSH client. If you receive an error while attempting to connect to your instance, see Troubleshoot connecting to your instance.

- In a terminal window, use the **ssh** command to connect to the instance. You specify the path and file name of the private key (.pem) and the IPv4 address for your instance. To connect to your instance, use the following command.

```
ssh -i /path/key-name.pem ubuntu@IPv4-address
```

  Replace /path/ with the full absolute path to your key pair. Replace key-name with the actual name of your key pair. Replace IPv4-address with the public IPv4 address of your instance.

  You will see a response like the following:

```
The authenticity of host 'ec2-198-51-100-1.compute-1.amazonaws.com
   (198-51-100-1)' can't be established.
   ECDSA key fingerprint is l4UB/neBad9tvkgJf1QZWxheQmR59WgrgzEimCG6kZY.
   Are you sure you want to continue connecting (yes/no)?
```

- (Optional) Verify that the fingerprint in the security alert matches the fingerprint that you previously obtained in (Optional) Get the instance fingerprint. If these fingerprints don't match, someone might be attempting a man-in-the-middle attack. If they match, continue to the next step.
- Enter `yes`.

  You will see a response like the following:

```
Warning: Permanently added 'ec2-198-51-100-1.compute-1.amazonaws.com'
   (ECDSA) to the list of known hosts.
```

- (Optional) Create a directory to organize your Real-ESSI simulation files. Replace test_folder with your folder name.

```
mkdir test_folder
```

6.3 Transfer Input Files to Real-ESSI Instance

This section describes how to transfer files with the secure copy protocol (SCP). The procedure is similar to the procedure for connecting to an instance with SSH.

- Open a terminal on your local Ubuntu desktop (or Amazon Workspace).
- Determine the file location on your local Ubuntu desktop (or Amazon Workspace) and the destination path on the instance. In the following example, the name of the private key file is key-name.pem, the file to transfer is main.fei, and the IPv4 address of the instance is IPv4-address. Enter the following command in your terminal.

```
scp -i /path1/key-name.pem /path2/main.fei ubuntu@IPv4-address:/path3/
```

There are three paths in this command.
➢ Replace /path1/ with the full absolute path to your key pair on your local desktop.
➢ Replace /path2/ with the full absolute path to the file to transfer on your local desktop.
➢ Replace /path3/ with the full absolute path to the destination on the Real-ESSI instance.

For example, your **scp** command may be:
`scp -i /home/han/Documents/han-key.pem /home/han/My_Model/main.fei ubuntu@52.26.2.245:/home/ubuntu/Test/`

- If you haven't already connected to the instance using SSH, you will see a response like the following:

```
The authenticity of host 'ec2-198-51-100-1.compute-1.amazonaws.com
(10.254.142.33)' can't be established.
RSA key fingerprint is
1f:51:ae:28:bf:89:e9:d8:1f:25:5d:37:2d:7d:b8:ca:9f:f5:f1:6f.
Are you sure you want to continue connecting (yes/no)?
```

(Optional) You can optionally verify that the fingerprint in the security alert matches the instance fingerprint. For more information, see (Optional) Get the instance fingerprint.

Enter `yes`.

- If the transfer is successful, the response is similar to the following:

```
Warning: Permanently added 'ec2-198-51-100-1.compute-1.amazonaws.com' (RSA)
to the list of known hosts.
main.fei                                  100%   164KB     1.3MB/s   00:00
```

- It's also possible to transfer entire folders using the **scp** command. In the following example, the name of the private key file is key-name.pem, the folder to transfer is folder, and the IPv4 address of the instance is IPv4-address. Enter the following command in your terminal.

```
scp -i /path1/key-name.pem -r /path2/folder ubuntu@IPv4-address:/path3/
```

There are three paths in this command.

➢ Replace /path1/ with the full absolute path to your key pair on your local desktop.
➢ Replace /path2/ with the full absolute path to the folder to transfer on your local desktop.
➢ Replace /path3/ with the full absolute path to the destination on the Real-ESSI instance.

For example, your **scp** command may be:

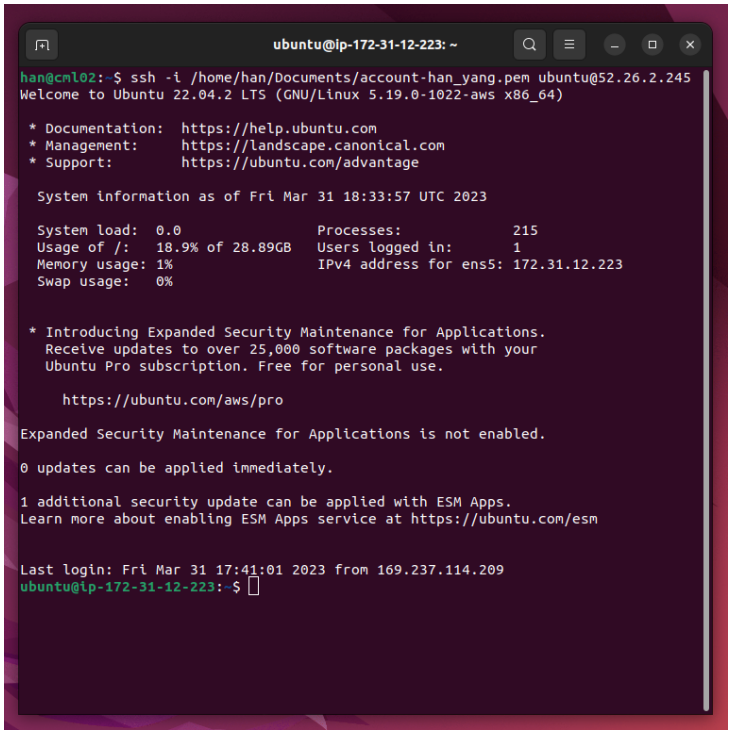
```
scp -i /home/han/Documents/han-key.pem -r /home/han/My_Model_Folder
ubuntu@52.26.2.245:/home/ubuntu/Test/
```

8. Run Simulations on Real-ESSI Instance

This chapter provides information about running your simulations on a launched Real-ESSI instance. Note that you should only attempt to do this after you have done the following:

- Create an AWS account
- Have access to a local Ubuntu desktop (or Amazon Workspace)
- Launch a Real-ESSI instance on AWS
- Connect to the Real-ESSI instance you have launched

Once you have successfully transferred your input files, open a new terminal and connect to the Real-ESSI instance. Your terminal should look like the following:



- Go to the directory where you transferred your input files. Use the following command:

```
cd /path/
```

  Replace /path/ with the full absolute path to your input files on the Real-ESSI instance.

- Change the permission of your file so that it can be read and executed. Use the following command:

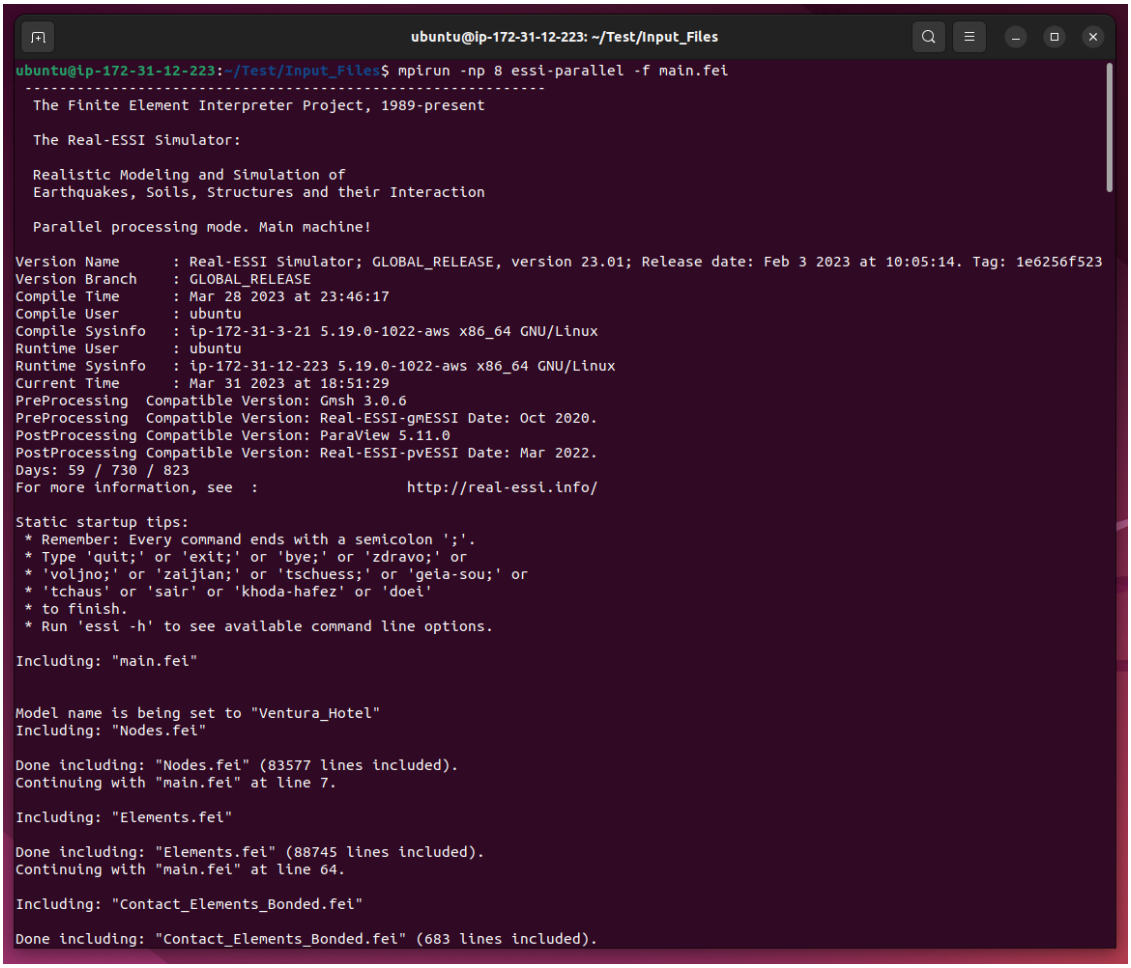
```
chmod a+rx main.fei
```

- Enter the following command to start your simulation:

```
mpirun -np num_pro essi-parallel -f main.fei
```

  Replace num_pro with the number of processes you want to use to run the simulation. Note that this number must be smaller than the number of available processes on your Real-ESSI instance. Consider using a different instance type with more available processes if your current set up is not enough for your model.

- Once the simulation has successfully started, you should see something like the following:

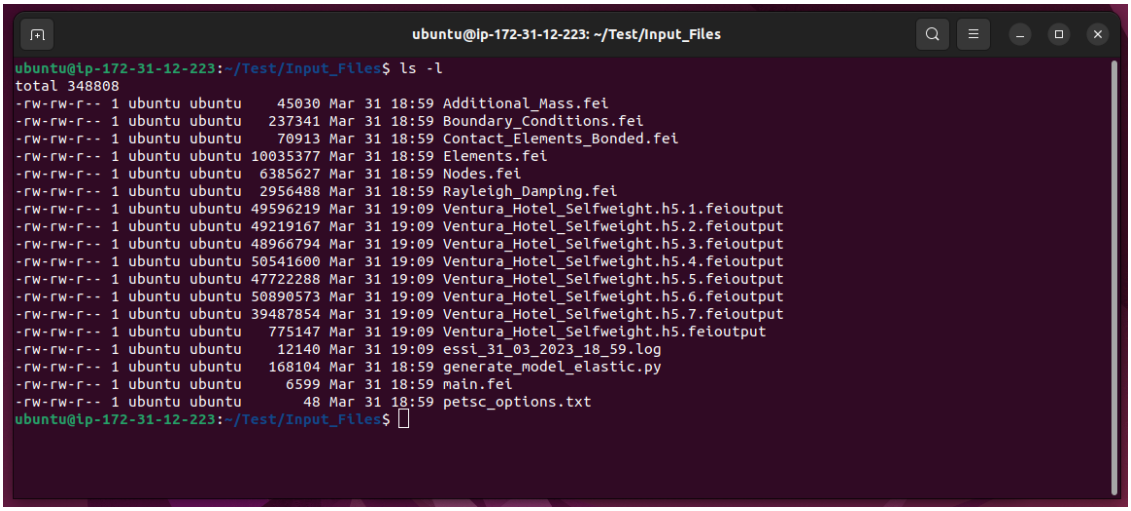


Now you just need to wait for the simulation to finish.

- Once the simulation is finished, you can use the following command to list all the files in your current directory:

```
ls -l
```

You should see something like the following:



Notice that all the output files and a log file are now present in your working directory. Note that all Real-ESSI output files have the suffix .feioutput. For more information, refer to the Real-ESSI Simulator Output Format Manual.

9. Transfer Output Files to Local Desktop (or Amazon Workspace)

This chapter provides information about how to transfer the output files of your Real-ESSI simulation back to your local desktop (or Amazon Workspace). Note that you should only attempt to do this after your simulation has finished.

The **scp** command is used to transfer the output files from the launched Real-ESSI instance to your local Ubuntu desktop (or Amazon Workspace).

- Open a terminal on your local Ubuntu desktop (or Amazon Workspace).
- In the following example, the name of the private key file is key-name.pem and the IPv4 address of the instance is IPv4-address. Enter the following command in your terminal.

```
scp -i /path1/key-name.pem ubuntu@IPv4-address:/path2/*.feioutput /path3/
```

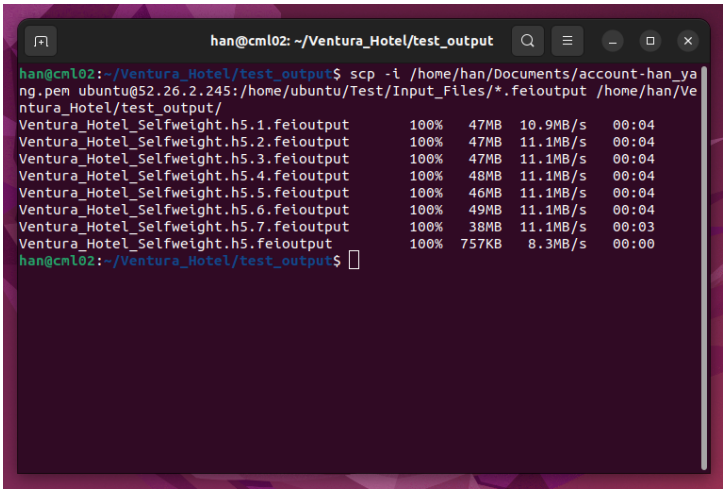
There are three paths in this command.
➢ Replace /path1/ with the full absolute path to your key pair on your local desktop.
➢ Replace /path2/ with the full absolute path to the location on the Real-ESSI instance.
➢ Replace /path3/ with the full absolute path to the location on your local desktop.

Note that the format *.feioutput means that the scp command will be executed for all the files with the suffix .feioutput. This is useful since you will have multiple output files that need to be transferred.

For example, your **scp** command may be:

```
scp -i /home/han/Documents/han-key.pem
ubuntu@52.26.2.245:/home/ubuntu/Test/*.feioutput /home/han/output/
```

- If the transfer is successful, you should see something like the following:



- Now that you have all the output files on your local Ubuntu desktop (or Amazon Workspace), you can proceed to work with them as you wish, e.g. post-processing. For more information on what you can do with your output files, refer to the Real-ESSI Simulator Post-Processing Manual.

211.8.3 AWS for Education

Amazon Web Services provides grants for educators and students from member institution[2] through AWS Educate program. AWS Educate offers cloud content, training, collaboration tools and AWS technology at no cost. Some of the AWS Educate program benefits:

- For Educators
  - $200 in AWS credits per educator - at member institutions.
  - $75 in AWS credits per educator - at non-member institutions.
  - Free AWS Technical Essentials eLearning course.
  - Free access to AWS content for classes.
- For Students
  - $100 in AWS credits per student - at member institutions.
  - $40 in AWS credits per student - at non-member institutions.
  - Access to AWS Technical Essentials Training Course (a $600 value).

If you have an email address from an educational institutions, you can use Real-ESSI on AWS for free through AWS Educate.

[2] List of member institution is available at this LINK.

211.8.4 AWS for Government

211.8.4.1 AWS GovCloud

211.8.4.2 AWS Secret Region

Chapter 212

Hardware Platform Design and Development

(1996-1999-2009-2011-)

212.1 Chapter Summary and Highlights

212.2 Introduction

Parallel computer used for simulations is based on a Beowulf concept (Sterling et al., 1995; Reschke et al., 1996; Sterling et al., 1998, 1999; Warren et al., 1998; Ridge et al., 1997). Hardware for a specific application to parallel computing for elastic-plastic finite elements has gone through a number of iterations (Jeremić et al., 1998, 1999) and is still evolving as the hardware components change in time and as our algorithms change/improve.

The choice of hardware platform is for a cost effective, off the shelf PC components, with about 2GB of memory per CPU/core and plenty of disk space (about 4TB for large dynamic runs). Linux is chosen as an operating system as it offers the best performance, is available in open source, which ensures that significant number of developers can contribute their expertise and can be customized to suite the needs of a parallel hardware and software. Microsoft Windows or Apple IOS operating systems are not best suited for parallel computing as their main development goal is user friendliness and not efficiency.

212.3 The NRC ESSI Computer

212.3.1 Version: December 2010

Operating System: Linux Fedora Core 14.

Kernel: `2.6.35.10-74.fc14.x86_64`

Compute Nodes (two):

- CPU: 2 $\times$ Intel Xeon E5620 Westmere 2.4 GHz Quad Core (8 threads) 32nm CPU with 256 KB Cache/core and 12MB Shared L3, DDR3-1066, 5.86GT/sec QPI, 80W
- RAM: 6 $\times$ 4GB DDR3 1333 MHz ECC/Registered Memory (24GB Total Memory 1066MHz)
- Disk: 8 $\times$ 500 GB Seagate Constellation ES 3.5" SATA/300 ST3500514NS 32MB Cache, 3Gb/s, NCQ, 7200RPM, 1.2 million hours MTBF Maximum Sustained Transfer Rate: 140 MB/sec (Linux Software RAID10)

Network: single GigaBit

212.3.2 Version: April 2012

In addition to the previous version.

Operating System:

Kernel:

Controller Node 1 (one):

- CPU: 2 x Opteron 6234 (2.4GHz, 12-Core, G34, 16MB L3 Cache) 115W TDP, 32nm
- RAM: 32GB (8 x 4GB) Operating at 1333MHz Max (DDR3-1333 ECC Registered DIMMs)
- NICs: Integrated Intel 82576 Dual-Port Gigabit Ethernet Controller
- Disk: 8 × 2TB Toshiba MK2002TSKB (3Gb/s, 7.2K RPM, 64MB Cache) 3.5" SATA

Compute Nodes, 8 (eight):

- CPU: 2 × Opteron 6234 (2.4GHz, 12-Core, G34, 16MB L3 Cache) 115W TDP, 32nm
- RAM: 32GB (8 x 4GB) Operating at 1333MHz Max (DDR3-1333 ECC Registered DIMMs)
- NICs:
  - Intel 82576 Dual-Port Gigabit Ethernet Controller
  - InfiniBand: ConnectX-2 QDR IB 40Gb/s Controller with QSFP Connector
- Disk: 1TB Toshiba MK1002TSKB (3Gb/s, 7.2K RPM, 64MB Cache) 3.5" SATA

Network (dual):

- HP ProCurve Switch 1810-48G 48 Port 10/100/1000 ports Web Managed Switch
- IB Switch: Mellanox MIS5030Q-1SFCA 36-port QDR switch; Cables: 9 x 3mtr QSFP-QSFP - Rating: QDR

Part 300

Verification and Validation

Chapter 301

Verification and Validation Introduction

(2003-2007-2009-2017-)

301.1 Chapter Summary and Highlights

301.2 Important Literature

Suggested reading:

Roache (1998);

Oberkampf et al. (2002); Oberkampf (2003); Oberkampf et al. (2007); Oberkampf and Trucano (2008); Oberkampf and Roy (2010); Oberkampf and Pilch (2017);

Roy and Oberkampf (2011);

Babuška and Oden (2004); Babuska et al. (2004); Oden et al. (2005); Oden et al. (2010a); Oden et al. (2010b); Szabó and Actis (2011) Szabó and Actis (2012)

ASME-VV-10 (2019) ASME-VV-20 (2009) ASME-VV-40 (2018)

ISO-90003 (2018)

NASA (2008) NASA (2016)

301.3 Verification and Validation

301.3.1 Definitions

Some definitions, as seen in ASME-VV-10 (2019).

Code is a computer implementation of the algorithm developed to facilitate formulation and approximations, approximate solutions a physical problem.

Model is a representation of a system, in our case a soil and/or structure with all the loads. physical conditions. Representation includes conceptual mathematical and computational models can also include physical

Verification and validation terms revision have been used interchangeably. However, it's important to follow precise definitions:

verification has to do with mathematics

validation has to do with physics

Objectives The objective of the verification and validation are

- demonstrate credibility of simulation results
- assess reliability of computer software and numerical methods used in simulation, and to
- assess the accuracy of simulations, with respect to available experimental observations.

- How do we use experimental simulations to develop and improve models
- How much can (should) we trust model implementations (verification)
- How much can (should) we trust numerical simulations (validation)

301.3.2 Trusting Simulation Tools

- Verification: The process of determining that a model implementation accurately represents the developer's conceptual description and specification. Mathematics issue. *Verification provides evidence that the model is solved correctly.*
- Validation: The process of determining the degree to which a model is accurate representation of the real world from the perspective of the intended uses of the model. Physics issue. *Validation provides evidence that the correct model is solved.*

301.3.3 Importance of V & V

- V & V procedures are the primary means of assessing accuracy in modeling and computational simulations
- V & V procedures are the tools with which we build confidence and credibility in modeling and computational simulations

301.3.4 Maturity of Computational Simulations

NRC committee (1986) identified stages of maturity in CFD

- Stage 1: Developing enabling technologies (scientific papers published)
- Stage 2: Demonstration of and Confidence in technologies and tools (capabilities and limitations of technology understood)
- Stage 3: Compilation of technologies and tools (capabilities and limitations of technology understood)
- Stage 4: Spreading of the effective use (changes the engineering process, value exceeds expectations)
- Stage 5: Mature capabilities (fully dependable, cost effective design applications)

301.3.5 Role of Verification and Validation

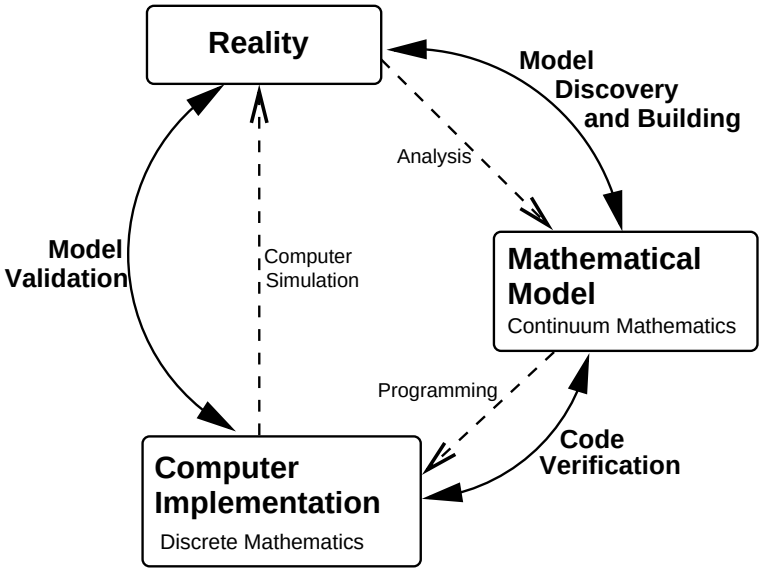


Figure 301.1: Role of Verification and Validation per Oberkampf et al. (2002).

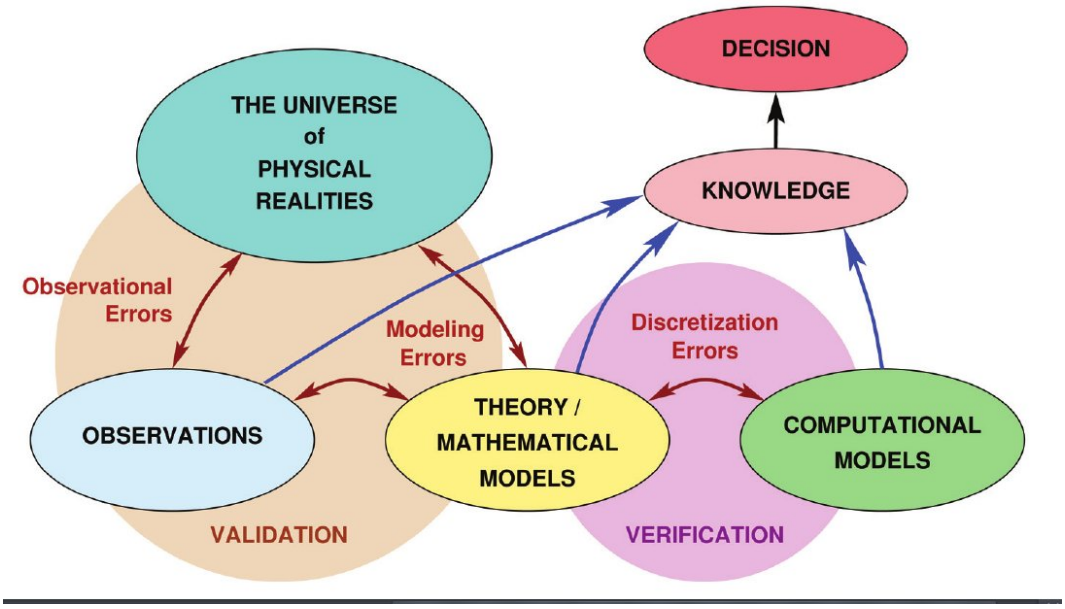


Figure 301.2: Role of Verification and Validation per Oden et al. (2010a).

301.3.5.1 Alternative V & V Definitions

IEEE V & V definitions (1984):

- Verification: The process of determining whether the products of a given phase of the software development cycle fulfill the requirements established during the previous phase
- Validation: The process of evaluating software at the end of the software development process to ensure compliance with software requirements.
- Other organization have similar definitions:
  - Software quality assurance community
  - American Nuclear Society (safety analysis of commercial nuclear reactors)
  - International Organization for Standardization (ISO)

301.3.5.2 Certification and Accreditation

- Certification: A written guarantee that a system or component complies with its specified requirements and is acceptable for operational use (IEEE (1990)).
  - Written guarantee can be issued by anyone (code developer, code user, independent code evaluator)
  - Code certification is more formal than verification and validation documentation
- Accreditation: The official certification that a model or simulation is acceptable for use for a specific purpose (DOD/DMSO (1994))
  - Only officially designated entities can accredit
  - Normally appointed by the customers/users of the code or legal authority
  - Appropriate for major liability or public safety applications

301.3.5.3 Independence of Computational Confidence Assessment

1. V&V conducted by the computational tool developer; *No Independence*
2. V&V conducted by a user from same organization
3. V&V conducted by a computational tool evaluator contracted by developer's organization
4. V&V conducted by a computational tool evaluator contracted by the customer

5. V&V conducted by a computational tool evaluator contracted by the a legal authority; *High Independence*

301.3.6 Simulation-Informed Decision Making

Based on Oberkampf and Pilch (2017)...

301.3.6.1 Purpose of Modeling and Simulation

Decision process:

- Low risk decisions: Simulation-based
- High risk decisions: computer simulations together with testing, Simulation-informed decisions process when simulation is integrated

Computer simulation is an Information Product:

- Supports a decision-making process
- Provides an improved understanding of the uncertainties and risk sustained by the use of the simulation results, a Simulation Customer.

Computer simulations are used to explore the

- design space
- use-misuse space

of a systems where physical testing is impractical and unaffordable...
What if studies over operating environment.
Operating environment:

- Normal
- Accident
- Misuse

301.3.7 Decision Making by Industry and by Regulatory Authorities

Based on NAFEMS short course on Credibility and Decision Making, June 2021, online...

Oberkampf...

Industry decision process to achieve decision result:

- return of investment, profit margin
- organizational goals
- competition
- personal goals
- organizational/personal value system
- experience with available options
- risk tolerance vs potential reward
- familiarity with information sources
- reliability of information sources

Regulatory Authority decision making:

- Risk to public
- Risk to environmental safety
- loss of political support for your regulatory function
- very risk averse value system

Information sources for decision support

- Previous experience with
  - Similar system of process, for example operating and reliability history
  - Modeling and simulation information, for example good and bad experiences
- Experimental data from testing prototype systems and subsystems
  - testing over portion of the operating envelope (application domain)

  - limited testing in adverse/abnormal environments, hazard loads...
  - limited testing of failure modes

- Credibility of modeling and simulation information depends on THREE elements:

  1. Suitable Training and Experienced personnel (example Sandia NL Fracture challenge (Boyce et al., 2014)), is a necessary element but not sufficient
  2. Quality Control of Modeling and Simulation Process, to achieve SIMUALTION CREDIBILITY
     - Physics modeling fidelity (geometric fidelity, spatial scales, temporal scales, initial conditions, boundary conditions, matgerial characteristics)
     - Verification activities (software quality assurance, static testing, dynamic testing, traditional analytic solutions, manufactured solutions, order of accuracy assessment)
     - Validation activities (validation experiments, hierarchical experiments, validation simulations, validation metrics, spatial discretization error, temporal discretization)
     - Uncertainty quantification (parametric uncertainty, model-form (modeling/epistemic) uncertainty, normal environments, abnormal environments, hostile environments, sensitivity analysis, extrapolation uncertainty)
  3. Assessment of Maturity of Modeling and Simulation (M&S) Results
     - Improved clarity of credibility in M&S results
       * Explicit statement of assumptions
       * Explicit statement of approximations
       * Explicit statement of limitations of simulation
       * Explicit statement of restrictions of simulation
       * Use of blind prediction of experimental tests
       * Use of uncertainty quantification of results
     - Use of M&S maturity assessment techniques
       * Predictive Capability Maturity Model (PCMM)
       * NASA standard for M&S (NASA, 2008, 2016)
       * ASME VVUQ techniques (ASME-VV-10, 2019; ASME-VV-40, 2018)

Responsibility for VVUQ

- Software Developer

  - Software quality assurance, bug fixes
  - Testing on different hardware and OS platforms
  - Code verification (determination of observed order of convergence)
  - Solution verification tools to estimate numerical solution errors
  - Model validation
  - Documentation of all components, models, algorithms, API, V&V, examples
  - Reproducibility of the simulation

- Simulation Producer, Analyst
  - Solution verification
  - Model parameter calibration using experimental data
  - Model validation, together with experimentalistis
  - Probabilistic simulation and uncertainty quantification
  - Software quality assurance
  - Code verification, in particular coverafe of features used
  - Documentation of simulation
  - Reproducibility of the simulation

- Simulation, Analysis Manager
  - Specifying how will simulation be used
  - Specifying what is the purpose of simulation to be used
  - Developing quality requirements for simulation
  - Developing review requirements for simulation
  - Assure, allocate adequate resources for the simulation
  - Assure, allocate adequate time for the simulation
  - Define documentation requirements for simulations
  - Define documentation requirements for experiments
  - Document in detail assumption, approximations and limitations of the simulation
  - Ensure training for analysis, modeling and simulation, personnel
  - Ensure expertise level of analysis, modeling and simulation, personnel

- Simulation Customer
  - Specifying what is the purpose of simulation to be used
  - Developing quality requirements for simulation
  - Developing review requirements for simulation
  - Allocate adequate resources for analyzing simulation results
  - Define documentation requirements for simulations
  - Ensure understanding of assumption, approximations and limitations of the simulation
  - Ensure expertise level of analysis users

Simulation Credibility Versus Risk, a Trade

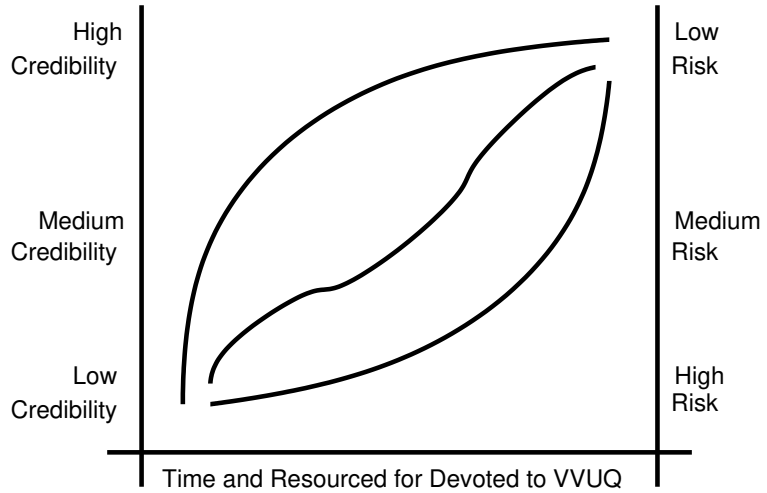


Figure 301.3: Simulation credibility versus risk, a trade that must be observed (Oberkampf...)

What one can find Under the Simulation Hood can be Surprising.

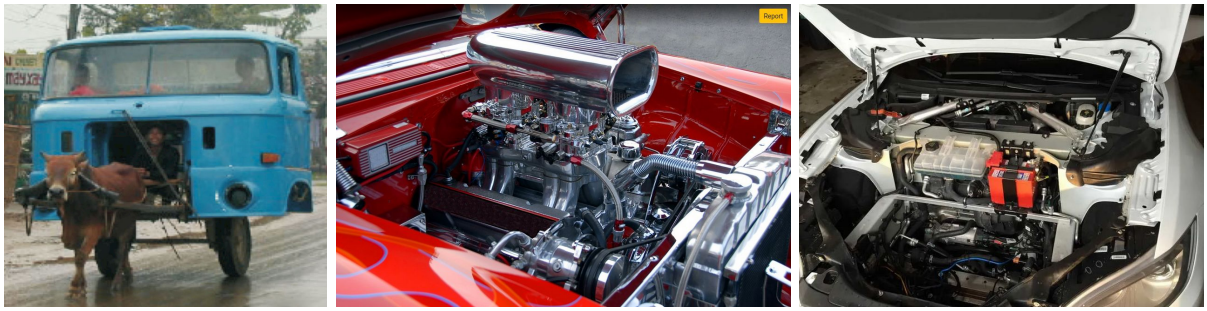


Figure 301.4: Under the Simulation Hood...

Simulation is not a magic bullet!

Simulation is Fragile!

301.3.8 Simulation Governance

Based on Szabó and Actis (2011, 2012)

Simulation Governance (SG) is needed to address prediction challenges in engineering practice. SG is a process to rank analysis models and to improve them over time with new experimental data. Control of numerical and modeling error is essential! Performance, relative and absolute, of analysis models is objectively evaluated.

301.3.8.1 Modeling, Experimental, Analytic and Numerical

Based on Szabó and Actis (2011, 2012)

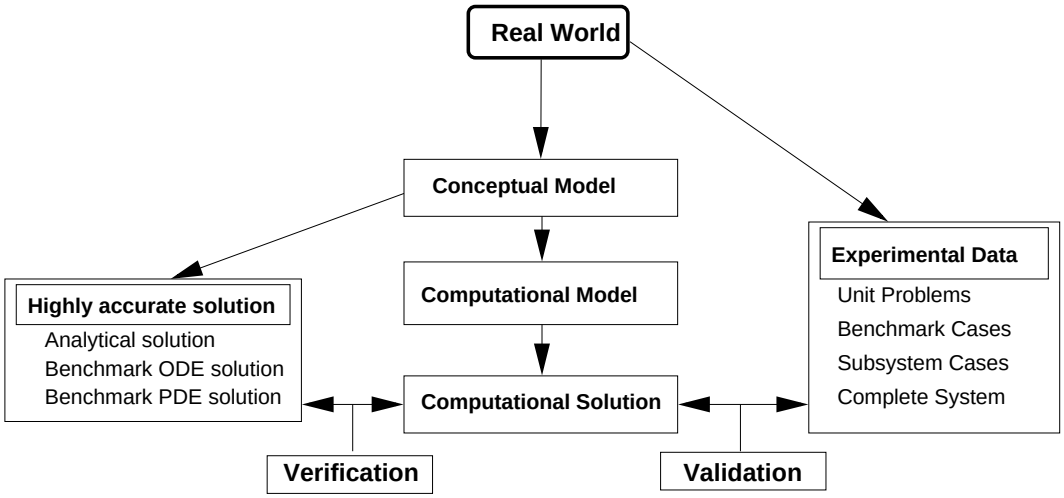
- Model is a transformation of data D that describes physical reality, into the data of interest F, results
- Data D includes all geometrical information, material information, calibration data and loading information...
- Data D features many uncertainties
- These uncertainties are transferred into corresponding uncertainties in F

- Transformation, D $\rightarrow$ F consists of operations that include modeling and simulation of mathematical problems, mechanics problems, statistical problems,
- Epistemic, parametric and aleatory, modeling uncertainties are mixed
- Simulation governance is set to minimize, control epistemic, modeling uncertainties,

Choice of mathematical model.

- Choice of mathematical model involves simplifying assumptions that restrict the scope of applicability of the model
- Are these simplifying assumptions justified for a particular application?
- This question can be addressed by performing virtual experiments
- Virtual experiments, sensitivity studies to determine if simplifying, restrictive modeling assumptions affect the date of interest to a significant degree
- Using virtual experiments Analyst, engineer can make informed decisions regarding the choice of modeling level of sophistication
- Software for virtual experiments should be able to analyze mathematical problem independent of choice of discretization
- For fixed discretization, one should be able to choose alternative mathematical model and investigate modeling change on the data of interest, results
- Similarly, for fixed mathematical problems, one should be able to develop a sequence of models that data of interest, results converge to their exact values...

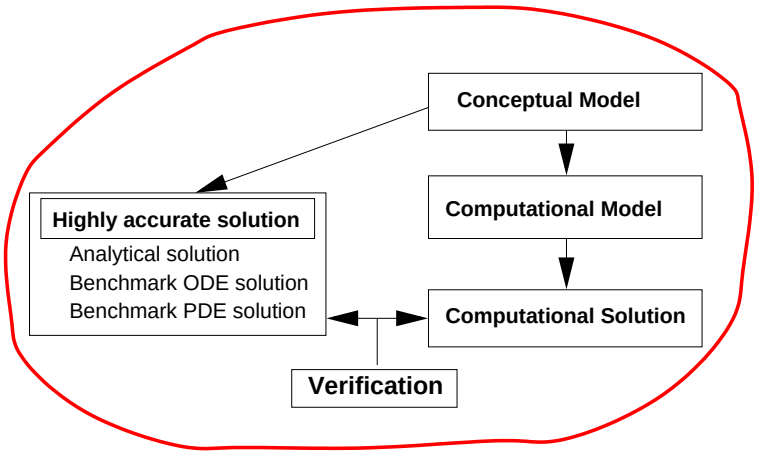
301.3.9 Detailed Look at Verification and Validation



301.3.9.1 On Verification

Verification: The process of determining that a model implementation accurately represents the developer's conceptual description and specification.

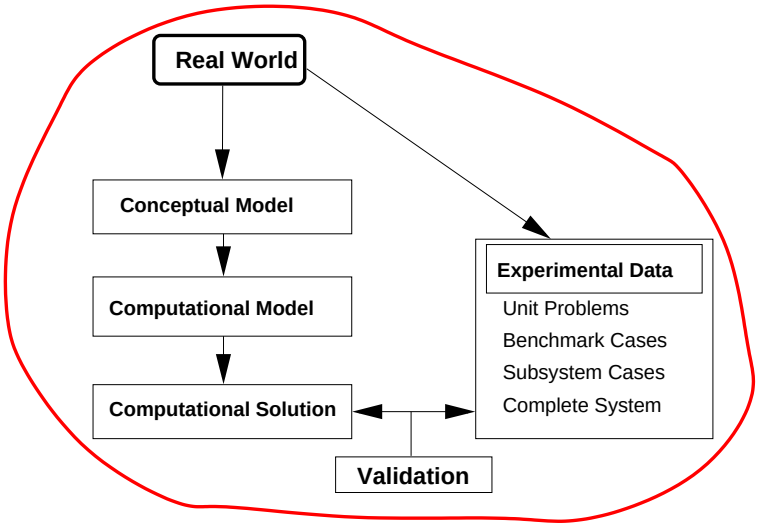
- Identify and remove errors in computer coding
  - Numerical algorithm verification
  - Software quality assurance practice
- Quantification of the numerical errors in computed solution



301.3.9.2 On Validation

Validation: The process of determining the degree to which a model is accurate representation of the real world from the perspective of the intended uses of the model.

- Tactical goal: Identification and minimization of uncertainties and errors in the computational model
- Strategic goal: Increase confidence in the quantitative predictive capability of the computational model



Goals of Validation Quantification of uncertainties and errors in the computational model and the experimental measurements

- Goals on validation
  - Tactical goal: Identification and minimization of uncertainties and errors in the computational model
  - Strategic goal: Increase confidence in the quantitative predictive capability of the computational model
- Strategy is to reduce as much as possible the following:
  - Computational model uncertainties and errors
  - Random (precision) errors and bias (systematic) errors in the experiments
  - Incomplete physical characterization of the experiment

Validation Procedure Uncertainty

- Aleatory uncertainty → inherent variation associated with the physical system of the environment (variation in external excitation, material properties...). Also know known as irreducible uncertainty, variability and stochastic uncertainty.
- Epistemic uncertainty → potential deficiency in any phase of the modeling process that is due to lack of knowledge (poor understanding of mechanics...). Also known as reducible uncertainty, model form uncertainty and subjective uncertainty

Types of Physical Experiments

- Traditional Experiments
  - Improve the fundamental understanding of physics involved
  - Improve the mathematical models for physical phenomena
  - Assess component performance
- Validation Experiments
  - Model validation experiments
  - Designed and executed to quantitatively estimate mathematical model's ability to simulate well defined physical behavior
  - The simulation tool (SimTool) (conceptual model, computational model, computational solution) is the customer

Validation Experiments

- A validation experiment should be jointly designed and executed by experimentalist and computationalist
  - Need for close working relationship from inception to documentation
  - Elimination of typical competition between each
  - Complete honesty concerning strengths and weaknesses of both experimental and computational simulations
- A validation Experiment should be designed to capture the relevant physics
  - Measure all important modeling data in the experiment

  - Characteristics and imperfections of the experimental facility should be included in the model

- A validation experiment should use any possible synergism between experiment and computational approaches
  - Offset strength and weaknesses of computations and experiments
  - Use high confidence simulations for simple physics to calibrate of improve the characterization of the experimental facility
  - Conduct experiments with a hierarchy of physics complexity to determine where the computational simulation breaks (remember, SimTool is the customer!)
- Maintain independence between computational and experimental results
  - Blind comparison, the computational simulations should be predictions
  - Neither side is allowed to use fudge factors, parameters
- Validate experiments on unit level problems, hierarchy of experimental measurements should be made which present an increasing range of computational difficulty
  - Use of qualitative data (e.g. visualization) and quantitative data
  - Computational data should be processed to match the experimental measurement techniques
- Experimental uncertainty analysis should be developed and employed
  - Distinguish and quantify random and correlated bias errors
  - Use symmetry arguments and statistical methods to identify correlated bias errors
  - Make uncertainty estimates on input quantities needed by the SimTool

301.4 Prediction

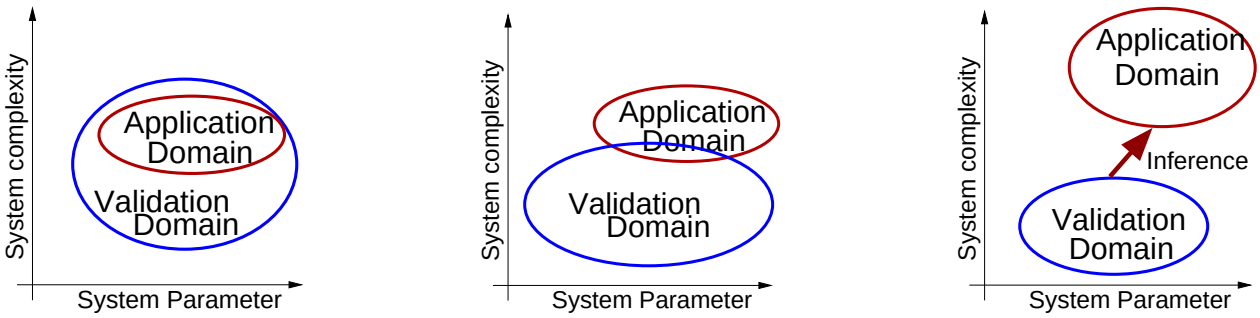
- Prediction: use of computational model to foretell the state of a physical system under consideration under conditions for which the computational model has not been validated
- Validation does not directly make a claim about the accuracy of a prediction
  - Computational models are easily misused (unintentionally or intentionally)
  - How closely related are the conditions of the prediction and specific cases in validation database
  - How well is physics of the problem understood

301.4.1 Relation Between Validation and Prediction

Quantification of confidence in a prediction:

- How do I quantify validation and its inference value in a predictions?
- How do I quantify verification and its inference value in a prediction?
- How far are individual experiments in my validation database from my physical system of interest?

301.5 Application Domain



- Rarely applicable to engineering systems (certainly not for infrastructure objects like bridges, buildings, port facilities, dams...)
- Even if the engineering system is small, environmental influences (generalized loads, conditions, wear and tare) are hard to predict
- Human factors (take Mars rover example with a memory overflow, operator forgot to flush the memory...)
- Inference $\Rightarrow$ Based on physics or statistics
- Validation domain is actually an aggregation of tests and thus might not be convex (bifurcation of behavior)
- Experimental facilities provide validation domain that is (for the most part) exclusively non–overlapping with the application domain.

301.5.1 Importance of Models and Numerical Simulations

- Verified and Validated models can be used for assessing behavior of
  - components or
  - complete systems,
- with the understanding that the environmental influences cannot all be taken into the account prior to operation
- but with a good model, their influence on system behavior can be assessed as need be (before or after the event)

301.5.2 Prediction under Uncertainty

- Ever present uncertainty needs to be estimated for predictions
- Identify all relevant sources of uncertainty
- Create mathematical representation of individual sources
- Propagate representation of sources through modeling and simulation process (Probabilistic Elastic Plastic Theory)

301.6 Intended use of Model

(ASME-VV-10, 2019), from Executive Summary, section 1

Verification assess the numerical accuracy of a computational model regardless of the physics being modeled.

- Code verification, addressing errors in the software and numerical algorithms
- Calculation verification, estimating numerical error due to under-resolved discrete representation of the mathematical model

Validation assess the degree to which the computational model is an accurate representation of the physics being modeled.

301.6.1 System Being Modeled

Relevant Physics

Modeling and experimental activities are guided by response quantities of interest.

301.6.2 Simulation Governance

Szabó and Actis (2011)

Szabó and Actis (2012)

Chapter 302

Source Code Verification

(1989-1990-1994-1995-2002-2005-2007-2009-2010-2017-2019-2021-)

(In collaboration with Dr. Yuan Feng, and Prof. Han Yang)

302.1 Chapter Summary and Highlights

Hatton (1997); Roache (1998); Oberkampf et al. (2002); Oberkampf (2003); Oden et al. (2005); Babuška and Oden (2004); Oden et al. (2010a); Oden et al. (2010b);

302.1.1 Numerical Algorithm Verification

302.1.2 Software Quality Assurance

Chapter 303

Code Stability Verification

(2002-2016-2017-2019-2021-)

(In collaboration with Dr. Yuan Feng and Prof. Han Yang)

303.1 Chapter Summary and Highlights

303.2 Introduction to Code Stability

This activity addresses source code stability. Source code verification is addressed elsewhere.

303.3 Motivation

In the software development process, Real-ESSI program is a framework and new features are continuously added. From time to time, some revisions for one feature may affect the normal operation of other existing components, like a specific element or material. To guarantee the stability and correctness of Real-ESSI program for each update. A group of test cases are collected in an automatic test suite. Then, after each revision, the developers are required to successfully pass all the test cases before the revisions are finally accepted to the trunk branch.

The features of the automatic Real-ESSI test is listed below:

- Automatic comparison of the maximum displacement output between the original and the new ESSI results.
- Automatic comparison of all the displacement and stress/strain output between the original and the new ESSI results.
- Automatic comparison of the terminal output/log between the original and the new ESSI results.
- Relative difference between the original and the new ESSI results.
- Colorful diagnostic information in the terminal.
- Support for the report in HTML format.
- Version information of both the original and the new ESSI.
- Number of passed cases and the statistics.

303.4 The framework of the automatic test

In practice, all the test cases are collected in one main folder and each test case has an independent subdirectory. In addition, *Bash* and *Python* are employed go over each leaf directory of the test cases

folder to execute essi and compare the results. A verification report is generated automatically after all the test cases are executed.

Regarding the test cases for the version stability, it is not necessary to choose the great model with lots of elements. The goal of version stability is to guarantee that the revision for one feature should not affect the normal operation of other commands. So the selection rule of test cases is to cover as much Real-ESSI DSL (domain specific language) commands as possible.

303.5 Installation and Tutorial

303.5.1 Installation

Makefile, *Bash* and *Python* are used to run all the test cases. So the automatic test is portable over various Linux platform as long as *Makefile*, *Bash* and *Python* are available. Besides, *git* is also required to download the test suite. In addition, if the user wants an additional report in HTML format, another package called *aha* is required. Install *aha* package on Ubuntu by using this command.

The automatic test suite is distributed within Real-ESSI source code.

```
${Real-ESSI}/CompGeoMechUCD_Miscellaneous/examples/
```

Notes: the automatic test will call the executable *essi* in the system/user PATH. So please make sure you have compiled or installed Real-ESSI first and then run this automatic test suite.

303.5.2 Tutorial

303.5.2.1 Run all verification test cases

In order to run all test cases to verify the installation, users can run

```
cd $RealESSI_PATH/
bash run_all_verification_sequential.sh $EXECUTABLE_PATH
bash run_all_verification_parallel.sh $EXECUTABLE_PATH
```

In addition to the conventional "-DDEBUG_MODE=DEBUG or OPTIMIZED", Users are required to do the test for executables compiled using the compiler options "-DDEBUG_MODE=O1 or O2" to fully verify the code. The verification will list all the results and errors. The errors may be big when the mesh is too coarse, or when Poisson's ratio is too high.

Furthermore, if developers want to verify against a previous version, developers can run

```
cd $RealESSI_PATH/
```

```
bash run_code_stability.sh $GitTAG
```

to test the verification results against one previous git version. The script will automatically checkout to the previous git-tag in detached mode and compile the old essi. After running test cases, developers can checkout to the original testing branch by running

```
cd $RealESSI_PATH/
git checkout $TestingBranch
```

In addition, if users want to clean the test results, users can run

```
cd $RealESSI_PATH/
bash clean_all_verification.sh
```

303.5.2.2 Run a single type of verification test cases

The usage of automatic test is written after the build process of source-code in Section 209.7.

For a single type of verification, for instance, in the folder

```
cd $RealESSI_PATH/CompGeoMechUCD_Miscellaneous/examples/analytic_solution
```

there are two clean options available in the main folder.

- The first one is *make clean*. This will only clean the new essi output results, including HDF5 files, terminal logs, and comparison logs.
- The second one is *make cleanall*. This will clean both the new and old essi output results for version stability test.

303.6 The underlying implementation of the automatic test

In most cases, the developers are not required to read and modify the underlying implementation of the automatic test. However, a basic introduction to the underlying implementation will help the future developers to customize the automatic test suite when necessary.

303.6.1 Generate the original results

When the command below is called,

```
bash generate_original.sh
```

Three things will be done. First, *make cleanall* will be called. This means that all the new and original output will be removed. Second, *essi* will be called to run each test cases and to generate the original HDF5 and original terminal output logs. Third, a bash command is employed to rename the HDF5 file and terminal log by adding *original* at the end of the filename.

303.6.2 Run essi and make comparison

When the command below is called,

```
bash make_comparison.sh
```

Four things will be done.

- First, *essi* will be called again to re-run each test cases. Please note that this *essi* should be the newly compiled *essi* in the development and debug stages.
- Second, a python-based comparison function will be called to compare the maximum displacement in the HDF5 output for each test case.
- Third, another python-based comparison function will be called to compare all the displacement and stress/strain results in the HDF5 output for each test case.
- Fourth, the terminal output log will be compared.

303.6.2.1 The terminal output/log comparison

The motivation to compare the terminal output is to avoid the unnecessary debug-purpose messages in the terminal output. During the debug stages, developers usually print out the variable values in the terminal. However, in the production stages, the debug-purpose messages should be disabled.

During the terminal output/log comparison, some lines are always different for each *essi* execution. These lines should be removed from comparison to avoid the false mismatch.

- The first type of different lines are the version information. The *essi* version information includes the *essi* compile time and execution time. They are always different for each execution. Therefore, these lines are extracted and the version information is printed out at the end of the comparison report.
- The second type of different lines are the *ETA* information. *ETA* stands for estimated time of arrival, which is a prediction for the *essi* execution time. However, the *ETA* information is inaccurate and they are different for each execution. Therefore, the *ETA* information is not compared either.

303.6.2.2 Reduce the comparison items and comparison time

During the debug stages, the developers might only want to compare the HDF5 output and the developers want to keep the verbose terminal messages so developers do not want to compare the terminal output log. Besides, during the debug stages, the developers may not want the time-consuming HDF5 output comparison. Therefore, to reduce the comparison items and comparision time, the developers can comment out the last two lines in *make_comparison.sh*.

303.7 Report Sample

For the sake of convenience, automatic test provides colorful diagnostic information. The green color is for the passed (matched) test case, while the red color is for the failed (mismatched) test case. The illustrative results are shown below. In addition, the automatic test also reports the relative path (location) of the test case. So if one of the test cases failed, developers can locate the subdirectory easily and check the mismatched model.

303.7.1 Passed test case

```
[++++++++++]
[==========] Running test cases...
[Location   ] test_cases/contact_elements/Four_Bar_Contact_Under_Monotonic_Shear_Loading
[RUN        ] ----------Testing results----------------
[-----------] Original_value    New_value       error     flag
[Passed Step] +2.000000e-03   +2.000000e-03     +0.00     pass
[Passed Step] +1.800000e-02   +1.800000e-02     +0.00     pass
[Passed Step] +1.000000e-02   +1.000000e-02     +0.00     pass
[Passed Step] +1.700000e-01   +1.700000e-01     +0.00     pass
[Case Passed] ----------Done this case!----------------
```

Figure 303.1: The report sample for a passed test case

303.7.2 Failed test case

In Figure 303.2, the *location* means the value location in the displacement results matrix of a HDF5 file. In the displacement results matrix, the column number is the step number and the row number is the dof (degree of freedom) number.

```
[-----------] ================================================
[-----------] |  Generalized_Displacements have mismatches:  |
[-----------] ================================================
[-----------] |  Location    |Original_value |     New_value |
[-----------] | (10, 1)      | +5.714642e-06 | +3.000000e-06 |
[-----------] | (15, 0)      | +2.076513e-05 | +1.000000e-05 |
[     Failed] -----------Displacement has mismatches!----------------
```

Figure 303.2: The report sample for a failed test case

```
[++++++++++++]
[============] Original ESSI version information:
[------------] Version : --NOT FROM GIT REPO--
[------------] Compiled: Jun 26 2016 at 23:18:45
[------------] Time Now: Jul 13 2016 at 15:24:20
[++++++++++++]
[============] New ESSI version information:
[------------] Version : --NOT FROM GIT REPO--
[------------] Compiled: Jun 26 2016 at 23:18:45
[------------] Time Now: Jul 13 2016 at 15:31:59
[============]
```

Figure 303.3: The report sample for the version information

```
[++++++++++++]
[------------]
[ Statistics] Passed cases / All cases= 61/61
```

Figure 303.4: The report sample for statistics information

303.7.3 Version information

303.7.4 Statistics

303.8 Future contribution

The automatic test is a test framework. It is easy to contribute your new test cases to the framework. The newly added test case must meet the following two requirements.

- The test case should be added as an independent leaf subdirectory within the *test_cases* folder.
- The test case should have a *main.fei* as the main model file.

Chapter 304

Validation Experiments

(2021-)

304.1 Chapter Summary and Highlights

304.2 Design of Experiments

Design of Experiments (DoE)

Chapter 305

Verification and Validation for Constitutive Problems

(1989-1991-1992-1994-1999-2003-2007-2009-2010-2017-2018-2019-2021-)

(In collaboration with Dr. Yuan Feng and Prof. Han Yang)

305.1 Chapter Summary and Highlights

305.2 Verification of Constitutive Integration

In this section, the accuracy analysis of the implicit algorithm is assessed. Examples of simple models (von Mises and Drucker-Prager) for accuracy analysis are demonstrated to verify general implicit algorithm. Convergence performance analysis is conducted. More details on accuracy analysis and consistent tangent stiffness are explained. Numerical simulation examples are demonstrated using the implemented framework. Special concerns are on the comparison of experimental data and numerical results of Dafalias-Manzari model.

305.2.1 Error Assessment

There are various error measures for the integration algorithms. Simo and Hughes (1998), Manzari and Prachathananukit (2001) used the relative stress norm by Equation 305.1,

$$\delta^r = \frac{\sqrt{(\sigma_{ij}-\sigma^*_{ij})(\sigma_{ij}-\sigma^*_{ij})}}{\sqrt{\sigma^*_{pq}\sigma^*_{pq}}} \tag{305.1}$$

where σ^*_{ij} is the 'exact' stress solution, and σ_{ij} the calculated stress solution. Alternatively, Jeremić and Sture (1997) used the normalized energy norm by Equation 305.2,

$$\delta^n = \frac{\left\|\sigma_{ij}-\sigma^*_{ij}\right\|}{\left\|p^{unit}\right\|} \tag{305.2}$$

where $\left\|\sigma_{ij}\right\|^2 = \sigma_{ij}D_{ijkl}\sigma_{kl}$, and D_{ijkl} is the elastic compliance fourth-order tensor, p^{unit} is the 'unit' energy norm for normalization.

The relative stress norm by Equation 305.1 is more reasonable since two points having the same $\left\|\sigma_{ij}-\sigma^*_{ij}\right\|$ but different $\sigma^*_{pq}\sigma^*_{pq}$ should have different error measures. However, this norm becomes singular and possible meaningless when $\sigma^*_{pq}\sigma^*_{pq}$ close to zero. The normalized energy norm by Equation 305.2 have no such singularity problem but it may give the same error index for two points having the same $\left\|\sigma_{ij}-\sigma^*_{ij}\right\|$ but different $\sigma^*_{pq}\sigma^*_{pq}$. In this work, we use these two error measure methods, but for simplicity, Equation 305.2 is modified into

$$\delta^r = \frac{\sqrt{(\sigma_{ij}-\sigma^*_{ij})(\sigma_{ij}-\sigma^*_{ij})}}{\sqrt{\sigma^0_{pq}\sigma^0_{pq}}} \tag{305.3}$$

where $\sigma^0_{pq}\sigma^0_{pq}$ is evaluated at some non-zero initial isotropic stress state. That is, the normalized error is evaluated by Equation 305.3, and the relative error is evaluated by Equation 305.1.

In our examples, the initial stress state point is set p^0 = 100 kPa, q^0 = 0 kPa, θ^0 = 0, which is the σ^0_{pq} in Equation 305.3. The one-step predicted stress state point for the implicit algorithm is within the range of $0.1 \leq p \leq 100$ kPa, $0 \leq q \leq 100$ kPa, $0 \leq \theta \leq \pi/3$. The 'exact' solution is actually unknown for most elastoplastic problems. Here the 'exact' solution is simply replaced by 50 substep solution of the explicit algorithm in the same one-step prediction incremental. All these error evaluations are within the material constitutive level.

The first test examples are von Mises models with the uniaxial yield strength k = 50 kPa, with linear elasticity parameters are Young's modulus $E = 1\times10^5$ kPa, and Poisson's ratio $\nu = 0.25$.

Figures 305.1 and 305.2 show the iso-error maps for the von Mises model with linear isotropic hardening. The linear hardening modulus $H = 2\times10^4$ kPa. The blue lines represents the yield surface boundary. It can be seen that the error magnitudes are as small as 10^{-10} to 10^{-9}, which implies that the solutions by implicit algorithm for this linear isotropic hardening von Mises model are numerically accurate.

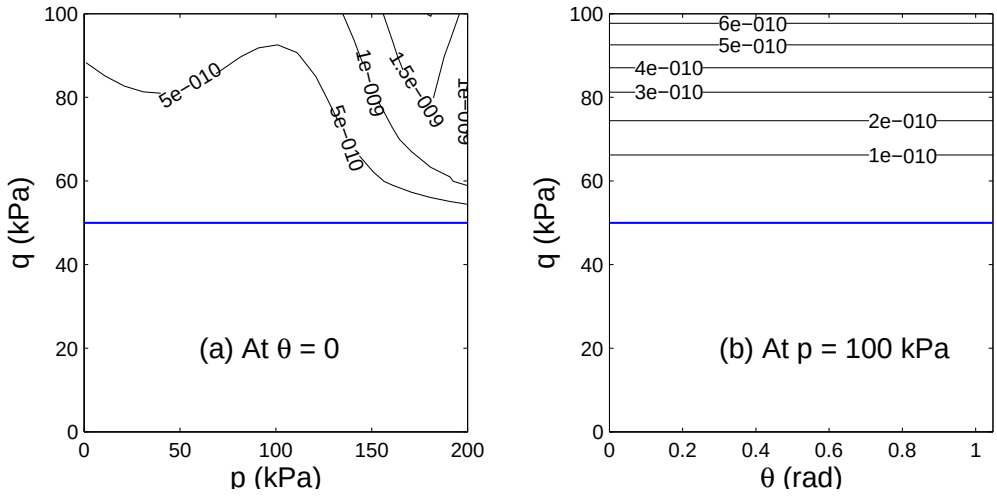


Figure 305.1: Normalized iso-error maps of von Mises model with linear isotropic hardening.

Figures 305.3 and 305.4 show the iso-error maps for the von Mises model with Armstrong-Frederick translational kinematic hardening. The hardening parameters are $h_a = 5\times10^4$ kPa and $C_r = 2.5\times10^3$. It can be seen that errors are very small which proves the good performance of the implicit algorithm. The iso-error map gives a good trend, i.e., the further away from the yield surface, the errors become more pronounced; the normalized errors are pressure-independent, which fits well the feature of von Mises model; the iso-error lines in the $q-\theta$ figure are parallel to the yield surface and are independent of the Lode's angle θ, which again fits well with von Mises model which is only q-related.

The second test examples are Drucker-Prager model with yield surface constant q/p = 0.8. Linear

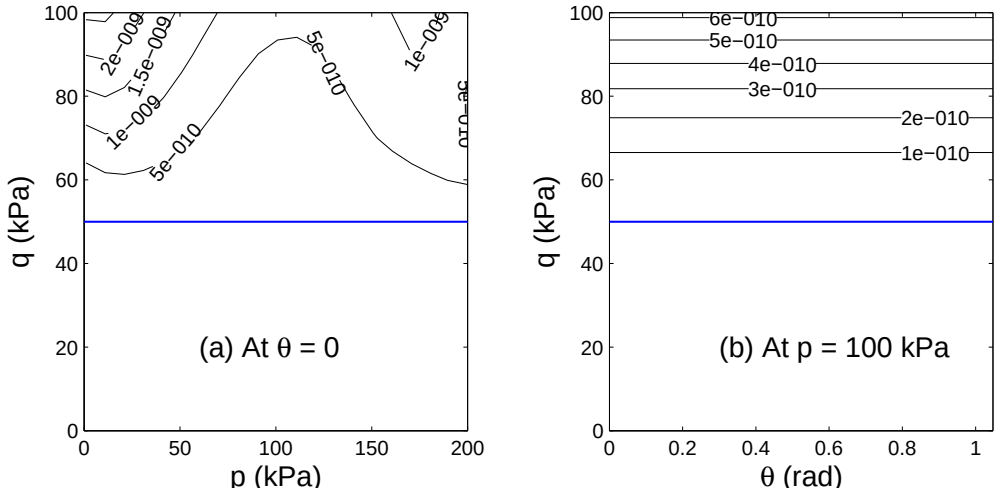


Figure 305.2: Relative iso-error maps of von Mises model with linear isotropic hardening.

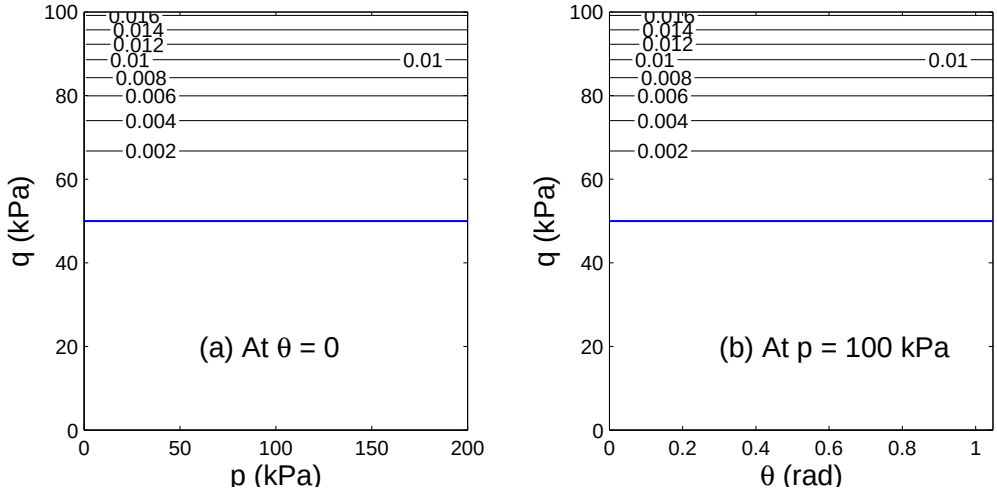


Figure 305.3: Normalized iso-error maps of von Mises model with Armstrong-Frederick kinematic hardening.

elasticity parameters are Young's modulus $E = 1\times10^5$ kPa, and Poisson's ratio $\nu = 0.25$.

The iso-error maps for perfectly plastic Drucker-Prager model are shown in Figures 305.5 and 305.6. The blue lines represents the yield surface boundary. It can be seen that the error magnitudes are as small as 10^{-11} to 10^{-9}. Again, these errors are so small that we can consider that the implicit algorithm give accurate solutions numerically.

Another Drucker-Prager model is with Armstrong-Frederick rotational kinematic hardening, and the parameters are $h_a = 20$, $C_r = 2$. The iso-error maps are shown in Figures 305.7 and 305.8. Unlike

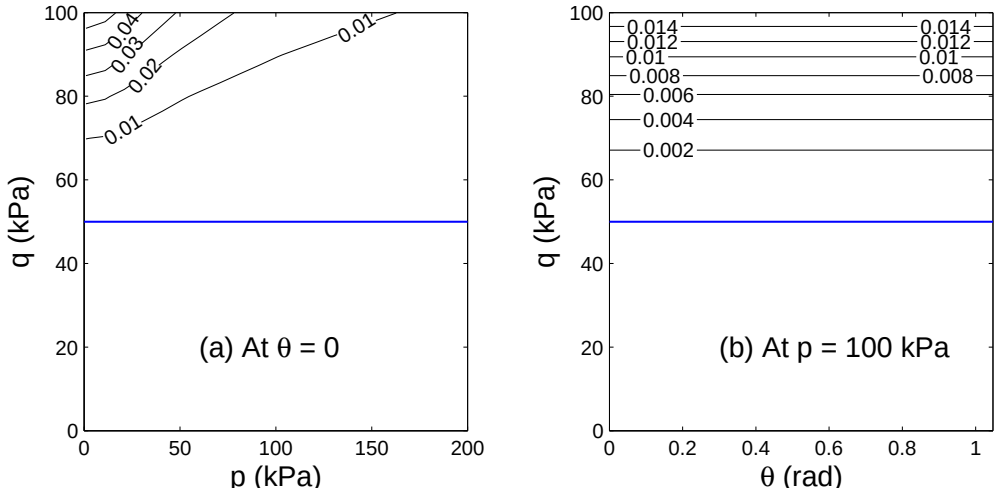


Figure 305.4: Relative iso-error maps of von Mises model with Armstrong-Frederick kinematic hardening.

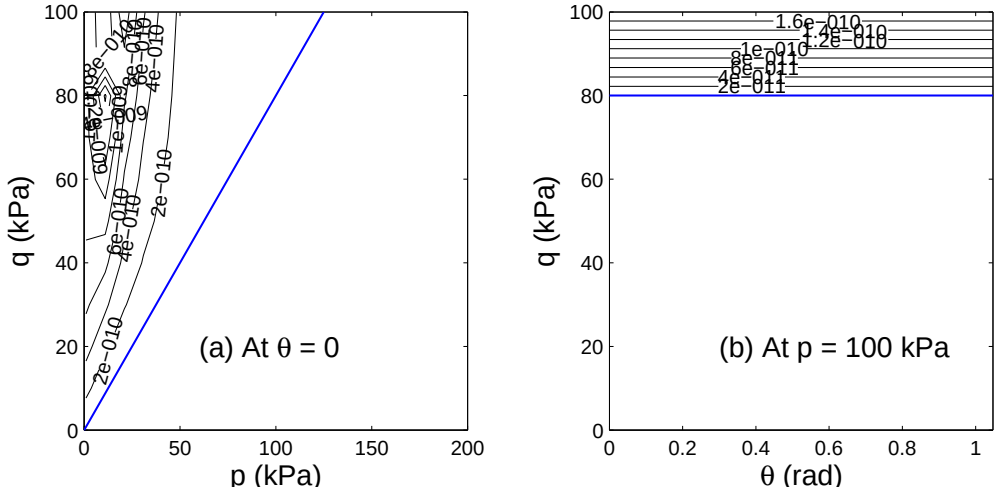


Figure 305.5: Normalized iso-error maps of Drucker-Prager perfectly plastic model.

von Mises model, the normalized errors are pressure-dependent, which fits well the feature of Drucker-Prager model; the iso-error lines in the $q-\theta$ figure are parallel to the yield surface and are independent of the Lode's angle θ, which still fits well with Drucker-Prager model which does not consider the third stress invariant, Lode's angle θ. From the relative iso-error maps in Figure 305.8, very dense iso-error lines are investigated in the region of small pressure, which is evidently due to the cone apex singularity of Drucker-Prager yield surface.

From the error analysis by the above von Mises and Drucker-Prager models, One finds that the implemented implicit algorithm can offer accurate solutions for simple models with simple hardening laws,

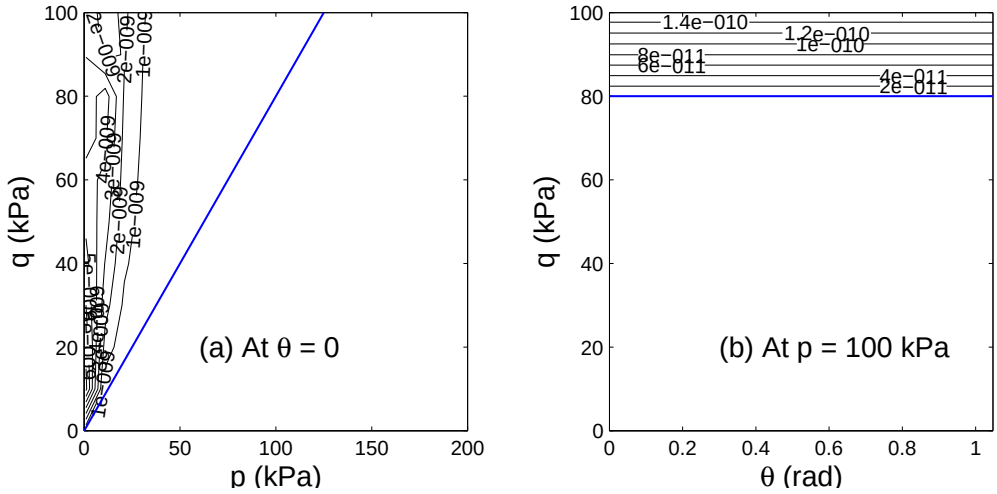


Figure 305.6: Relative iso-error maps of Drucker-Prager perfectly plastic model.

e.g. von Mises model with linear hardening and Drucker-Prager model with perfectly plastic hardening (no hardening). Complicated hardening laws increases the error even for simple plastic models, although the errors are still small. These observations match the well known conclusion that the error of the implicit algorithm is pretty dependent on the smoothness of the solution. The implemented implicit algorithm proves very robust for von Mises and Drucker-Prager model with simple or complicated hardening laws.

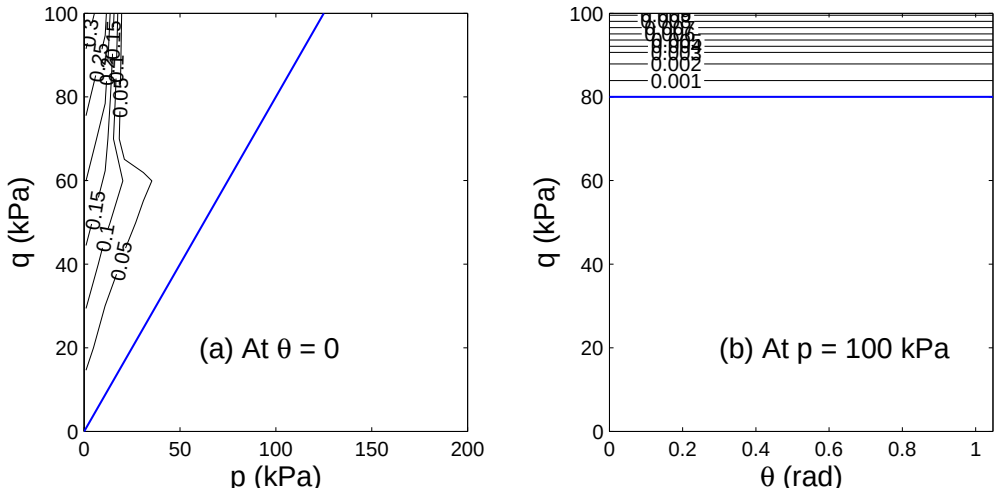


Figure 305.7: Normalized iso-error maps of Drucker-Prager model with Armstrong-Frederick kinematic hardening.

Figures 305.9 and 305.10 present the iso-error maps of Dafalias-Manzari model. The initial void ratio is 0.8, and the other parameters are from Dafalias and Manzari (2004a). The blue lines represents

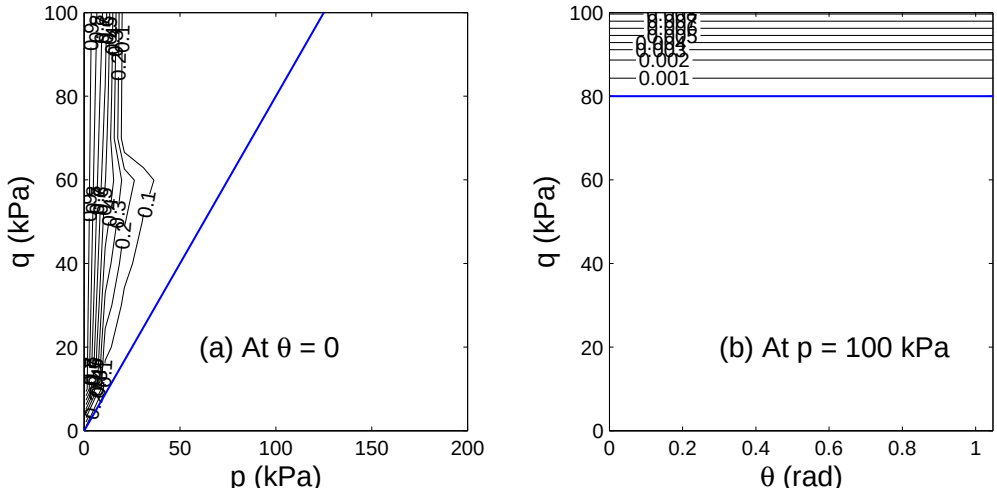


Figure 305.8: Relative iso-error maps of Drucker-Prager model with Armstrong-Frederick kinematic hardening.

the yield surface boundary (slope ratio $m = 0.01$). Unlike von Mises and Drucker-Prager models, the iso-error lines in the $q - \theta$ figure of Dafalias-Manzari model are not parallel to the yield surface and are dependent of the Lode's angle θ, which was one of the highlighting improvements upon the previous version (Manzari and Dafalias, 1997). From Figure 305.10, when the predicted stress q close to 100 kPa, or or about 100 times the yield strain increment, the relative errors can reach up to 100%, which implies that even for implicit algorithm, Dafalias-Manzari model still requires small strain increments. However, when $q < 30$ kPa, or about 30 times of the yield strain increment, the relative errors are less than 5%, excepts at the region close to the yield surface apex.

It should be pointed out that errors for the complex Dafalias-Manzari model are much bigger than those of simple models (e.g. von Mises and Drucker-Prager), due to its high non-linearity. However, if the predicted stress (or in other words, the strain increment) is small enough, the algorithm errors are within a small tolerant range.

Figures 305.9 and 305.10 are based on an approach of averaged elastic moduli. Instead, Figures 305.11 and 305.12 present iso-error maps based on constant elastic moduli approach.

305.2.2 Constitutive Level Convergence

In the implemented implicit algorithm, the iteration continues until the absolute value of yield function and the residue norm of considering variables are less than some small tolerances, or if by equations,

$$|f| \leq Tol1; \quad r_{norm} = \|\mathbf{r}\| \leq Tol2 \tag{305.4}$$

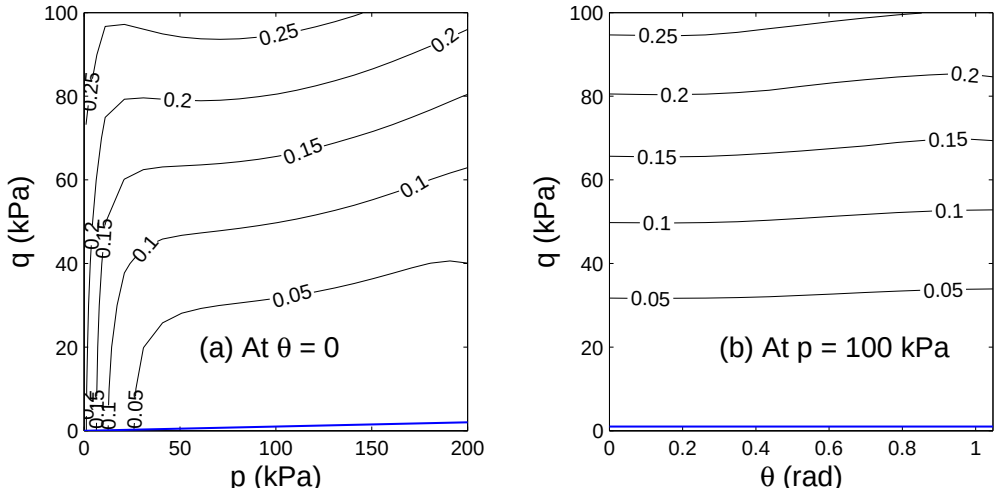


Figure 305.9: Normalized iso-error maps of Dafalias-Manzari model with average elastic moduli.

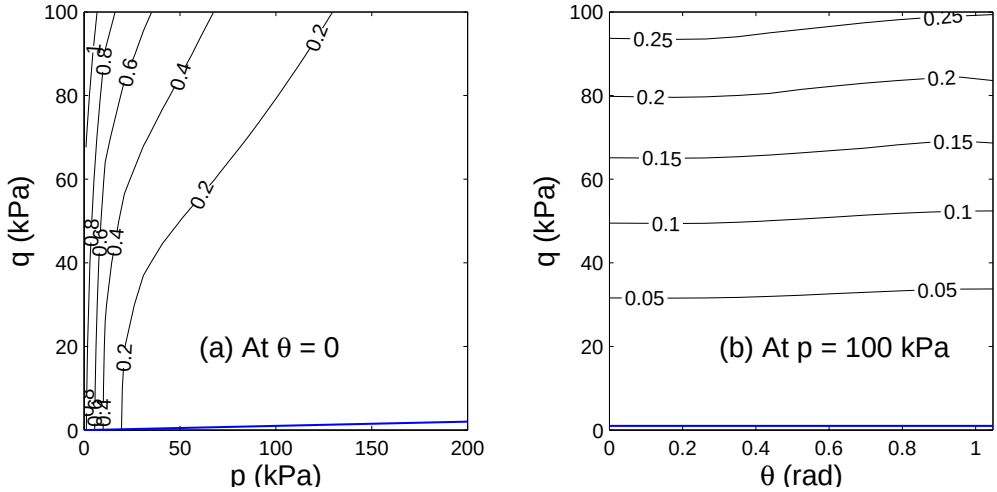


Figure 305.10: Relative iso-error maps of Dafalias-Manzari model with average elastic moduli.

Three examples including simple von Mises model with linear isotropic hardening, relative complicated Drucker-Prager model with Armstrong-Frederick kinematic hardening, and even more complicated Dafalias-Manzari model considering fabric dilation effect are presented here to show the constitutive level convergence performances for the implemented implicit algorithm. In all these examples, both $|f|$ and r_{norm} v.s. iteration numbers are plotted. Iteration number 0 represents the 'virtual' iteration number before return mapping implicit iteration cycle. $|f|$ at iteration number 0 thus means $|f|$ at the first predicted stress for each load increment; there is no value of r_{norm} at iteration number 0. A tolerance of $Tol1 = Tol2 = 1\times10^{-7}$ is for both $|f|$ and r_{norm}. The iteration stops when $|f| \leq Tol1$

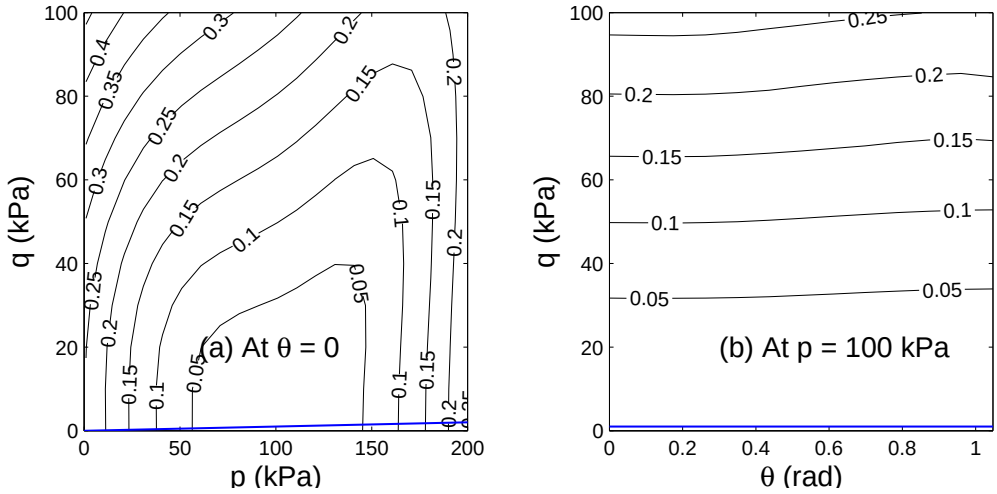


Figure 305.11: Normalized iso-error maps of Dafalias-Manzari model with constant elastic moduli.

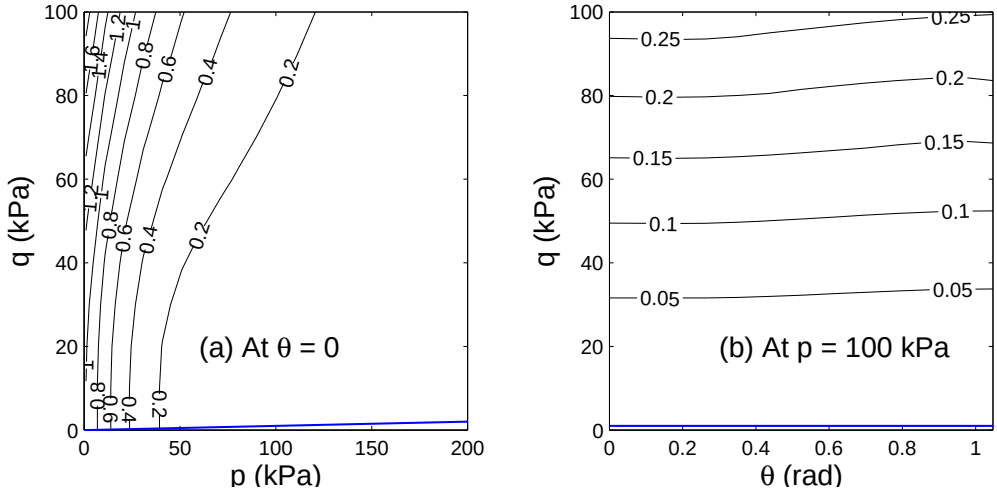


Figure 305.12: Relative iso-error maps of Dafalias-Manzari model with constant elastic moduli.

and $r_{norm} =\leq Tol2$ are satisfied, even if there is only one iteration number. The initial stress is an isotropic stress state of p_0 = 100 kPa. The undrained-like load increment is adopted by strain control as $\epsilon_{11} = -2\epsilon_{22} = -2\epsilon_{33} = n\times\Delta\epsilon$, where n is the load increment number and $\Delta\epsilon$ is the strain increment interval, ϵ_{ij} are strain components.

Figure 305.13 shows the typical constitutive level convergence performance for von Mises model with linear isotropic hardening. The input parameters are Young's Modulus $E = 1\times10^5$ kPa, Poisson's ratio ν = 0.25, the material strength k = 50 kPa, and the linear isotropic hardening modulus $H = 2\times10^4$ kPa. The strain increment interval $\Delta\epsilon$ is set 2×10^{-4}. It can be seen that for this simple example, only

two iteration steps are needed and $|f|$ and r_{norm} are far smaller than the tolerances and in fact close to the machine floating error value, or in other words, the stresses are exactly at the yield surface and the residue norm is zero.

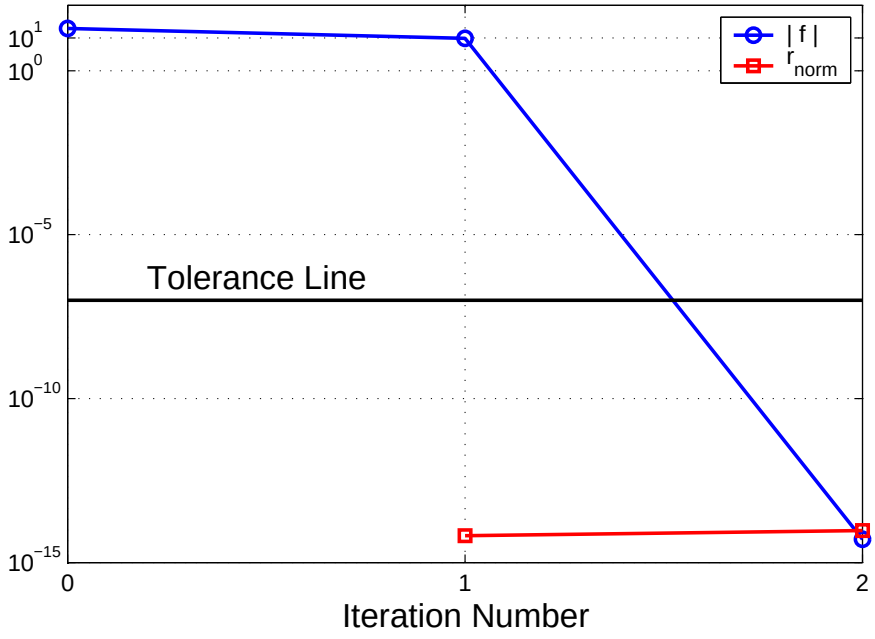


Figure 305.13: Typical convergence for von Mises model with linear isotropic hardening (tolerance value 1×10^{-7}).

Figure 305.14 shows the typical constitutive level convergence performance for Drucker-Prager model with Armstrong-Frederick kinematic hardening. The input parameters are Young's Modulus E = 1×10^4 kPa, Poisson's ratio ν = 0.25, the material q/p ratio is 0.8, and the Armstrong-Frederick parameters are h_a = 20, C_r = 2. The strain increment interval $\Delta\epsilon$ is set -2×10^{-4}. For this example, both $|f|$ and r_{norm} are stably decreasing with the increasing iteration number; However, $|f|$ and r_{norm} show different rates; $|f|$ needs 5 iteration steps while r_{norm} needs 7 iteration steps; The convergence rate of r_{norm} lags behind that of $|f|$.

Figure 305.15 shows the typical constitutive level convergence performance for the complicated Dafalias-Manzari model considering fabric dilation effect. The input parameters are as in Table 305.1, and the initial void ration is set as 0.8. Different from the above examples, The strain increment interval $\Delta\epsilon$ is set a much smaller value of -1×10^{-5}. In this example, again, both $|f|$ and r_{norm} are stably decreasing with the increasing iteration number; However, $|f|$ and r_{norm} show different rates; $|f|$ needs less iteration steps than r_{norm}; The convergence rate of r_{norm} lags behind that of $|f|$. It should be mentioned here for this complicated Dafalias-Manzari model considering fabric dilation effect,

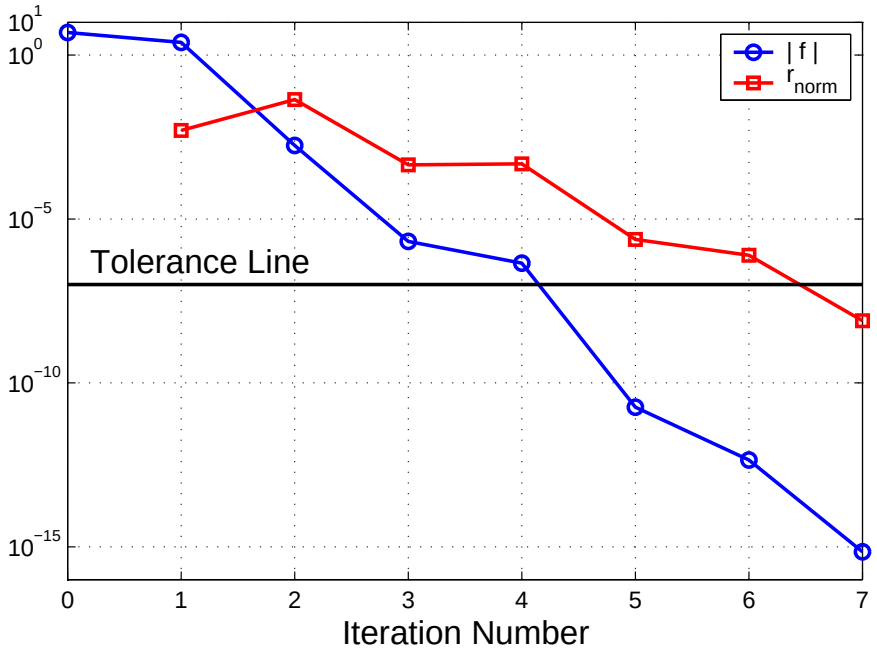


Figure 305.14: Typical convergence for Drucker-Prager model with Armstrong-Frederick kinematic hardening (tolerance value 1×10^{-7}).

the typical constitutive level convergence performance is similar to that of Drucker-Prager model with Armstrong-Frederick kinematic hardening, but with much smaller strain increment interval.

From the above examples, it is clear that the simpler the model is , the better constitutive level convergence performances are observed. This is consistent to the error assessment in section 305.2.1. Generally, the implemented implicit algorithm shows stable constitutive level convergence performances provided an appropriate small strain increment interval for the material model.

305.3 Validation of Constitutive Model Predictions

305.3.1 Dafalias Manzari Material Model

Validation is performed by comparing experimental (physical) results and numerical (constitutive) simulations for the Toyoura sand. It should be noted that we have not done validation against 2D or 3D tests (say centrifuge tests) as characterization of sand used in centrifuge experiments is usually less than complete for use with advanced constitutive models. Moreover, as our approach seeks to make predictions of prototype behavior, scaling down models (and using them for comparison with numerical predictions) brings forward issues of physics of scaling which we would rather stay out of. The material parameters

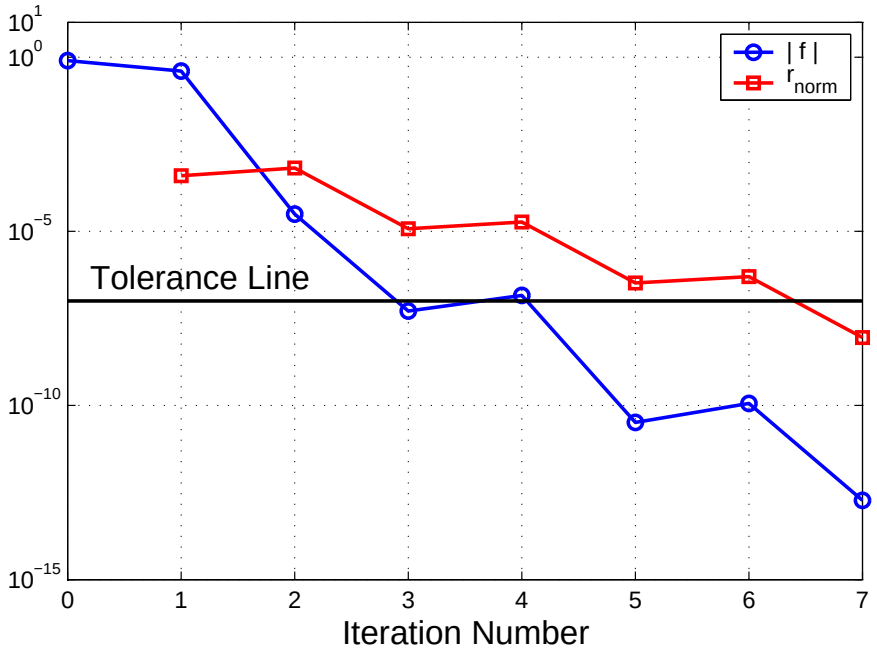


Figure 305.15: Typical convergence for Dafalias-Manzari model (tolerance value 1×10^{-7}).

used are from Dafalias and Manzari (2004a) and are listed in Table (305.1). Several simulated results are compared with the experimental data published by Verdugo and Ishihara (1996).

Figure (305.16) presents both loading and unloading triaxial drained test simulation results for a relatively low triaxial isotropic pressure of 100 kPa but with different void ratios of $e_0 = 0.831, 0.917, 0.996$ at the end of isotropic compression. During the loading stage, one can observes the hardening and then softening together with the contraction and then dilation for the denser sand, while only hardening together with contraction for the looser sand. The significance of the state parameter to the soil modeling is clear from the very different responses with different void ratios at the same triaxial isotropic pressure.

Figure (305.16) also shows both loading and unloading triaxial drained test simulation results for a relatively high triaxial isotropic pressure of 500 kPa but with different void ratio of e_0=0.810, 0.886, 0.960 at the end of isotropic compression. Similar phenomenon are observed as with tests (physical and numerical) for relatively low triaxial isotropic pressure. However, due to the higher confinement pressure, the stress-strain responses are higher at the same strain, which proves the significant pressure dependent feature for the sands.

Figure (305.17) presents both loading and unloading triaxial undrained test simulation results for a dense sand with the void ratio of e_0=0.735 at the end of isotropic compression but with different

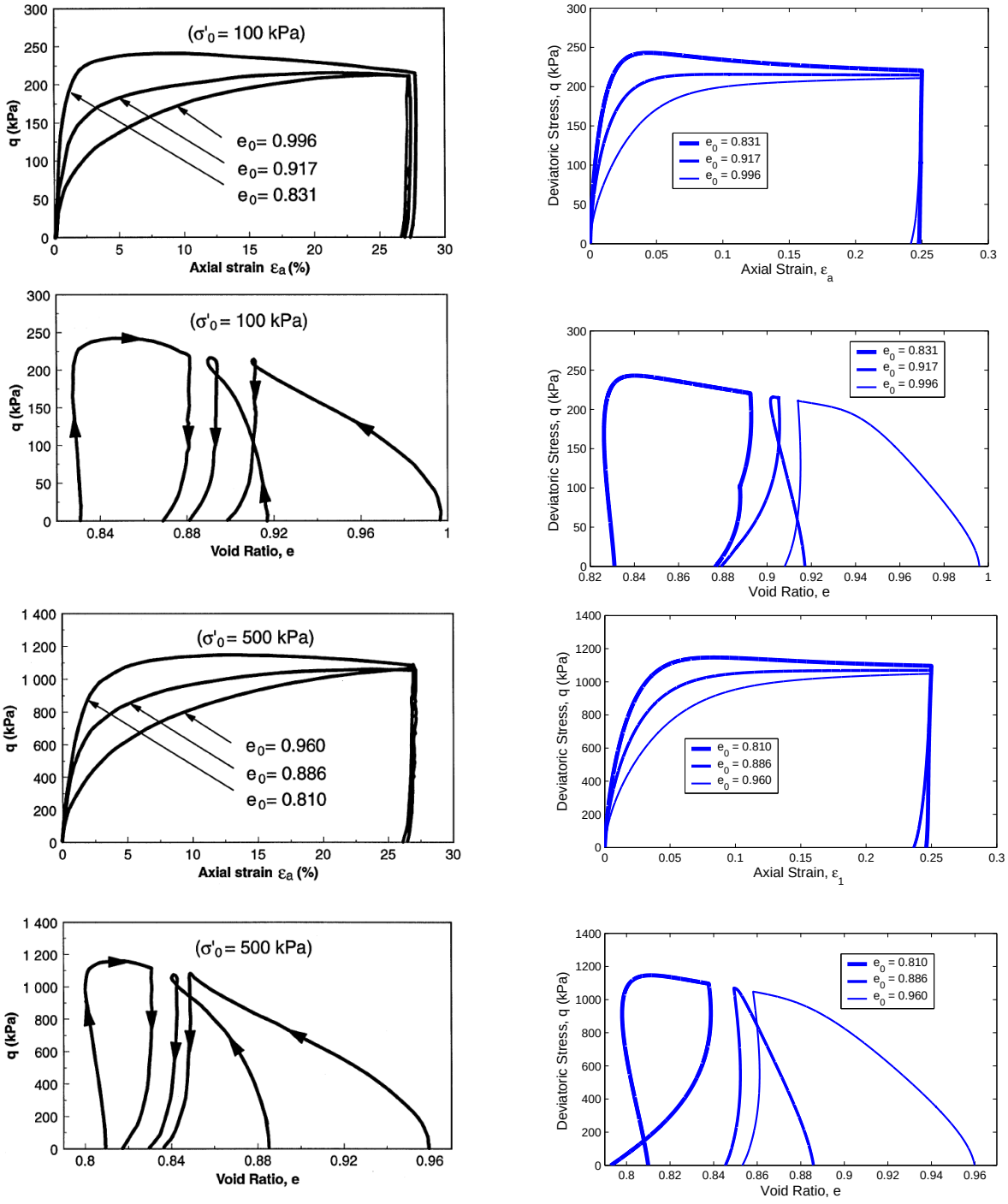


Figure 305.16: Left: Experimental data; Right: Simulated results.

Table 305.1: Material parameters of Dafalias-Manzari model.

material parameter		value	material parameter		value
Elasticity	G_0	125 kPa	Plastic modulus	h_0	7.05
	ν	0.05		c_h	0.968
Critical sate	M	1.25		n_b	1.1
	c	0.712	Dilatancy	A_0	0.704
	λ_c	0.019		n_d	3.5
	ξ	0.7	Fabric-dilatancy	z_{max}	4.0
	e_r	0.934		c_z	600.0
Yield surface	m	0.01			

isotropic compression pressures of $p_0 = 100$, 1000, 2000, 3000 kPa. During the loading stage, one observes that each of responses are close to the critical state line for the very various range of isotropic compression pressures. For the higher isotropic compression pressure, the contraction response with softening is clearly observed, while for the smaller isotropic compression pressure, it is a dilation response without softening.

Close matching of physical testing data with constitutive predictions represents a satisfactory validation of our material model. This validation with previous verification gives us confidence that predictions (presented in next section) represent well the real, prototype behavior.

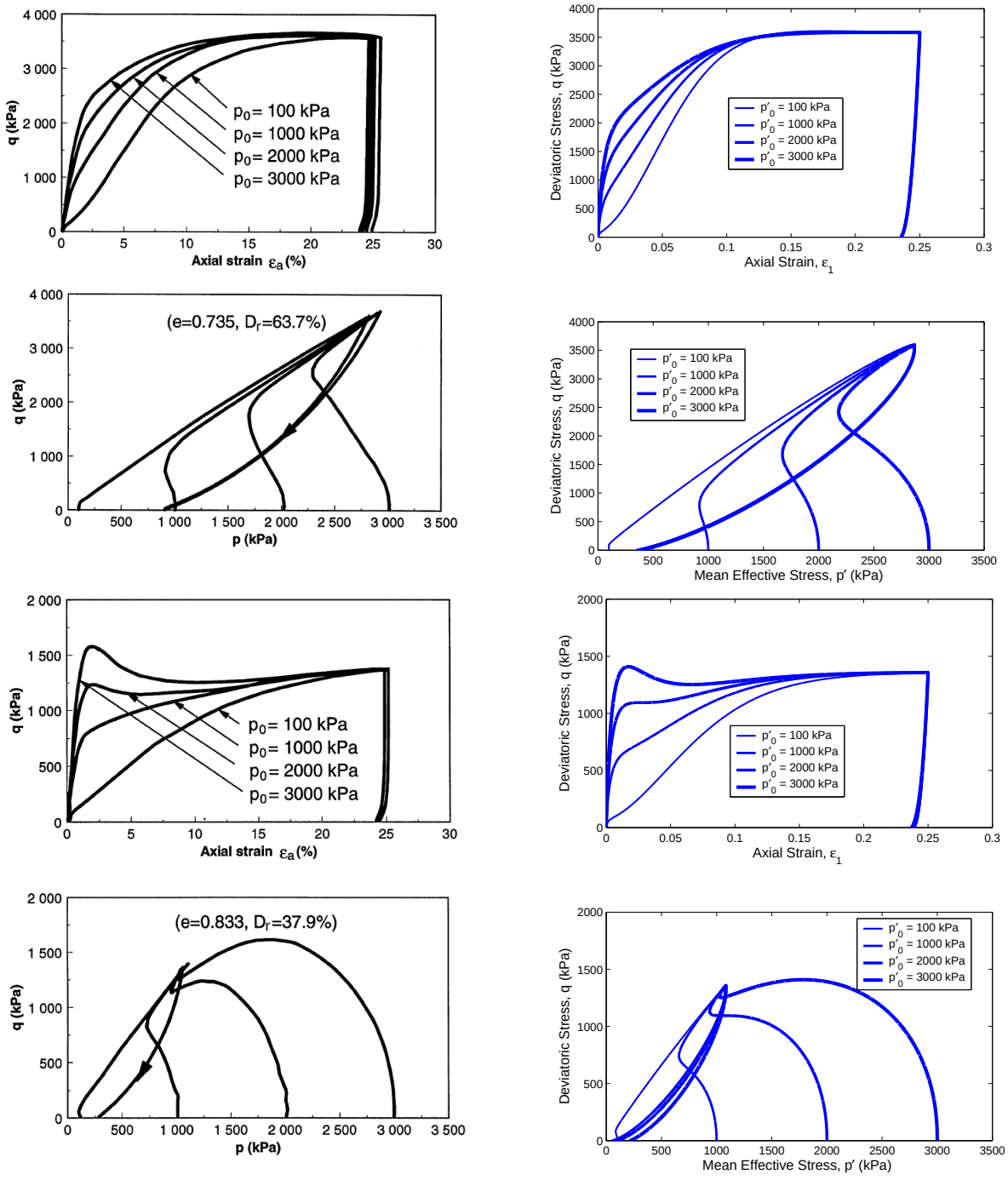


Figure 305.17: Left: Experimental data; Right: Simulated results.

Chapter 306

Verification and Validation for Static and Dynamic Finite Element Level Solution Advancement Algorithms

(1994-2003-2009-2012-2015-2017-2019-)

(In collaboration with Prof. José Antonio Abell Mena and Dr. Yuan Feng)

306.1 Chapter Summary and Highlights

306.2 Verification for Static Solution Advancement

306.3 Verification for Dynamic Solution Advancement

Spectral Radii, Argyris and Mlejnek (1991)...

- Test consists of a 2-DOF system periods $T_1 = 4.0\,\mathrm{s}$ and $T_2 = 1.0\,\mathrm{s}$.
- Different integration steps.
- Different values of method parameters (α, β, γ).
- Results are compared with theoretical predictions for algorithmic damping and period shifts.

306.3.1 Verification for Dynamic Solution Advancement, Newmark Method

(Newmark, 1959)

Iff

$$\gamma \geq \frac{1}{2}, \quad \beta \geq \frac{1}{4}(\gamma + \frac{1}{2})^2 \tag{306.1}$$

the procedure is unconditionally stable and second-order accurate.

Different values for γ and β can be used to create various integration methods:

- For $\gamma = 0.5$ (and corresponding $\beta = 0.25$) there is no numerical damping.
- Any γ value greater than 0.5 will introduce numerical damping.
- Trapezoidal rule or average acceleration method for $\beta = 1/4$ and $\gamma = 1/2$,
- Linear acceleration method for $\beta = 1/6$ and $\gamma = 1/2$,
- Explicit, central difference method for $\beta = 0$ and $\gamma = 1/2$.
- Strongest numerical damping values is obtained for values $\beta = 1$ and $\gamma = 2/3$, as spectral ratio $\rho_\infty = 0$ (Hughes (1987), page 502)

For more details see chapter 108 on page 537.

Following Argyris and Mlejnek (1991); Hughes (1987), to calculate the analytic ξ and analytic $\bar{\omega}$ a matrix A is constructed. The explicit definition of amplification matrix A for the Hilber Hughes Taylor (HHT) family of algorithms (where Newmark is obtained by setting $\gamma = 0$) is

$$A = \frac{1}{D}\begin{bmatrix} 1+\alpha\beta\Omega^2 & 1 & \frac{1}{2}-\beta \\ -\gamma\Omega^2 & 1-(1+\alpha)(\gamma-\beta)\Omega^2 & 1-\gamma-(1+\alpha)(\frac{1}{2}\gamma-\beta)\Omega^2 \\ -\Omega^2 & -(1+\alpha)\Omega^2 & -(1+\alpha)(\frac{1}{2}-\beta)\Omega^2 \end{bmatrix} \tag{306.2}$$

where

$$\begin{aligned} D =& 1+(1+\alpha)\beta\Omega^2 \\ \Omega =& \omega\triangle t \\ \omega =& (K/M)^{\frac{1}{2}} \end{aligned} \tag{306.3}$$

The eigenvalue of amplification matrix A will be two complex conjugate roots $\lambda_{1,2}$ and a so-called spurious root λ_3 which satisfy $|\lambda_3| < |\lambda_{1,2}| \leq 1$. The roots $\lambda_{1,2}$ will be

$$\lambda_{1,2} = A \pm Bi \tag{306.4}$$

Then, the analytic damping ratio ξ and analytic period $\bar{\omega}$ becomes

$$\begin{aligned} \bar{\xi} =& -ln(A^2+B^2) \\ \bar{\omega} =& \bar{\Omega}/\triangle t \\ \bar{\Omega} =& arctan(B/A) \end{aligned} \tag{306.5}$$

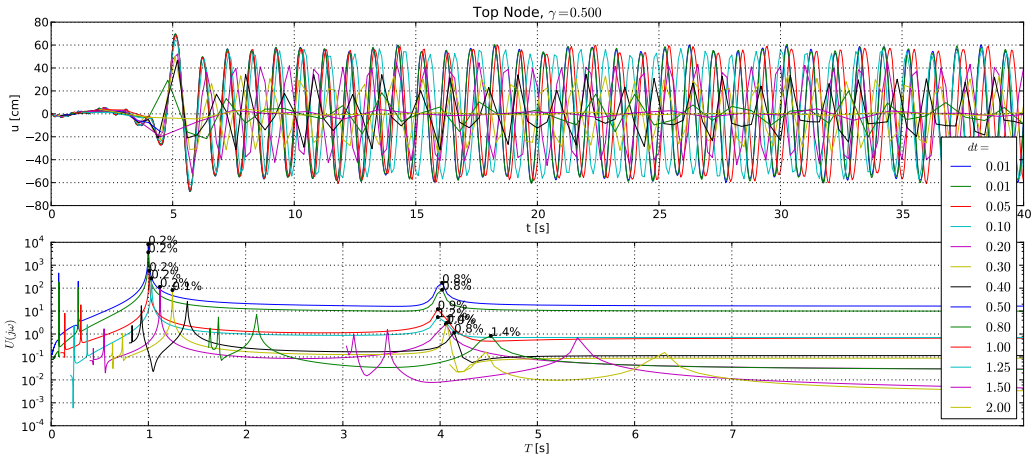


Figure 306.1: Verification: Dynamic solution advancement, Newmark method (p2a-newmark-05top)...

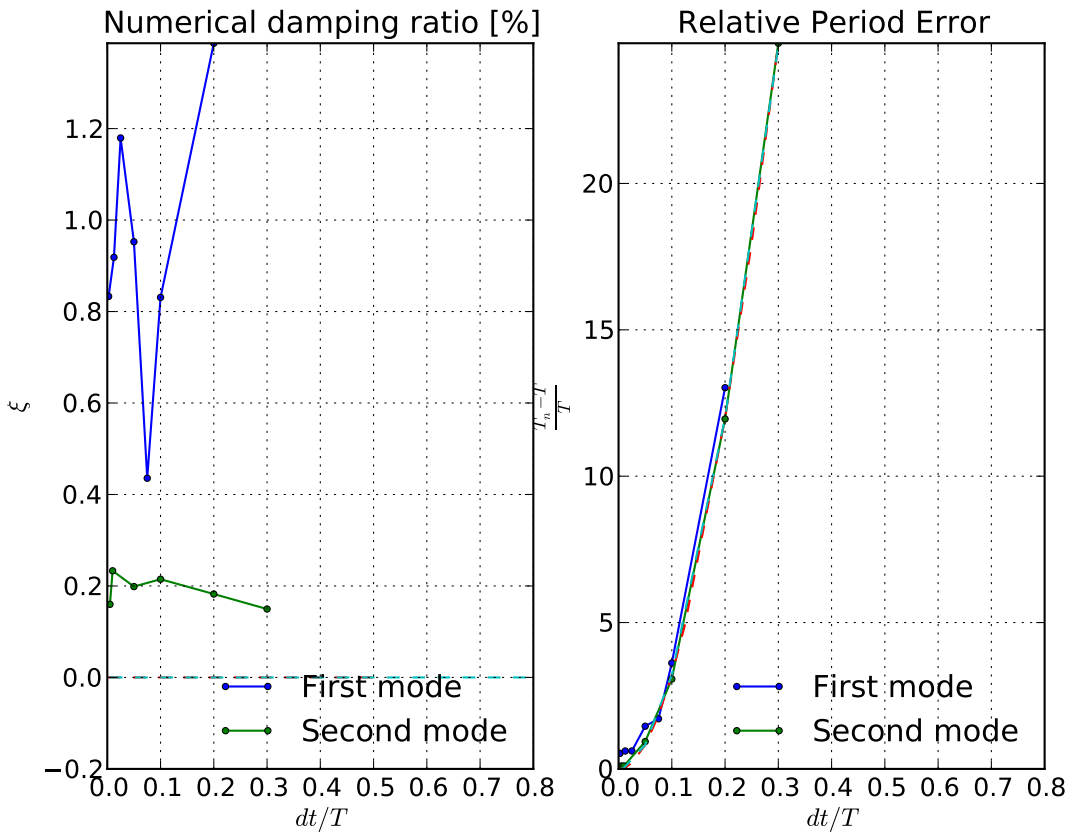


Figure 306.2: Verification: Dynamic solution advancement, Newmark method (p2a-newmark-05errors)...

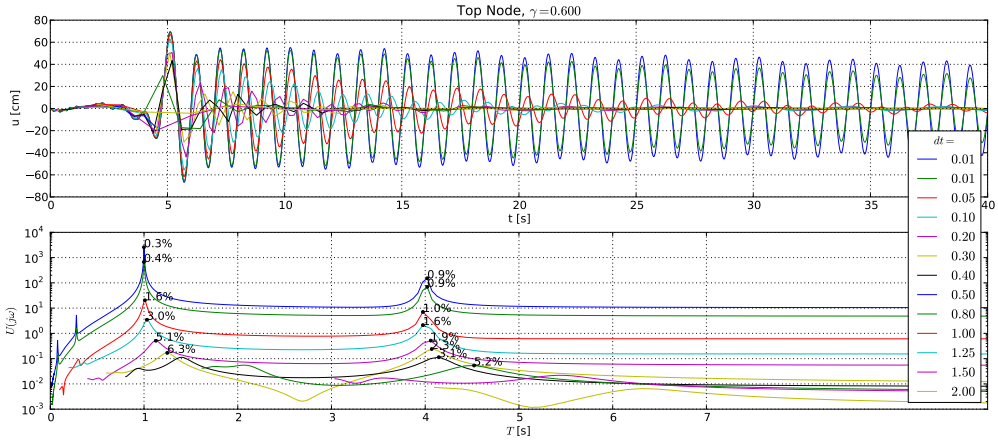


Figure 306.3: Verification: Dynamic solution advancement, Newmark method (p2a-newmark-06top)...

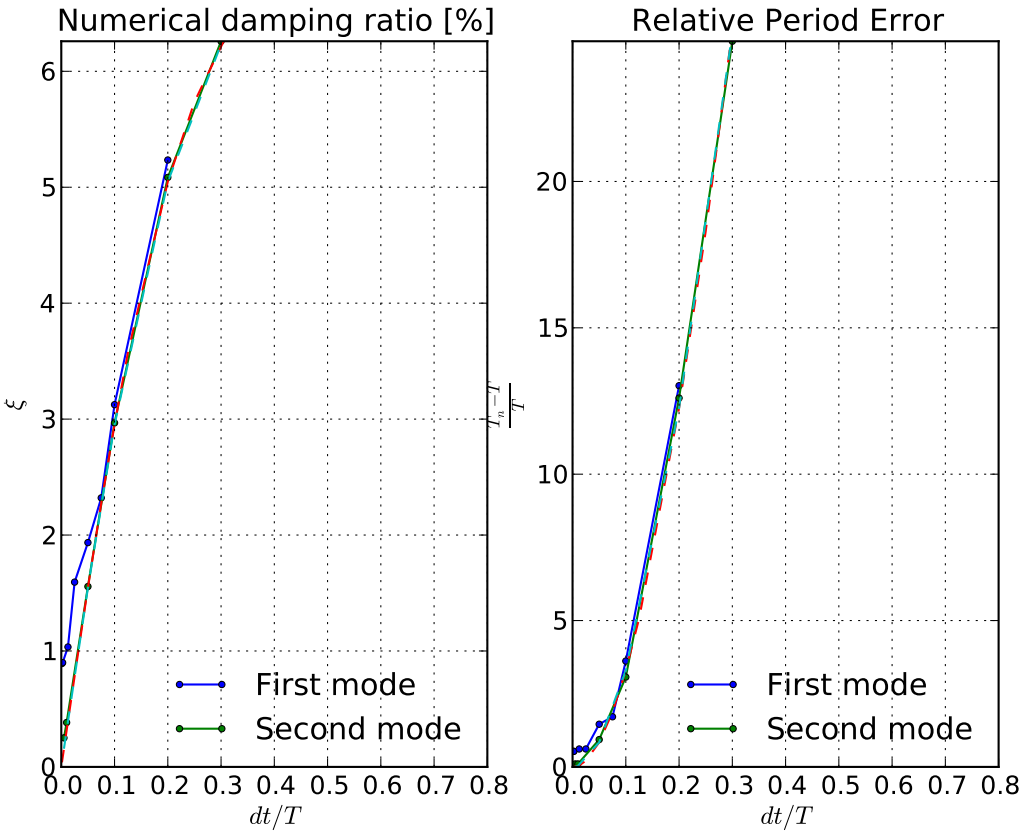


Figure 306.4: Verification: Dynamic solution advancement, Newmark method (p2a-newmark-05errors)...

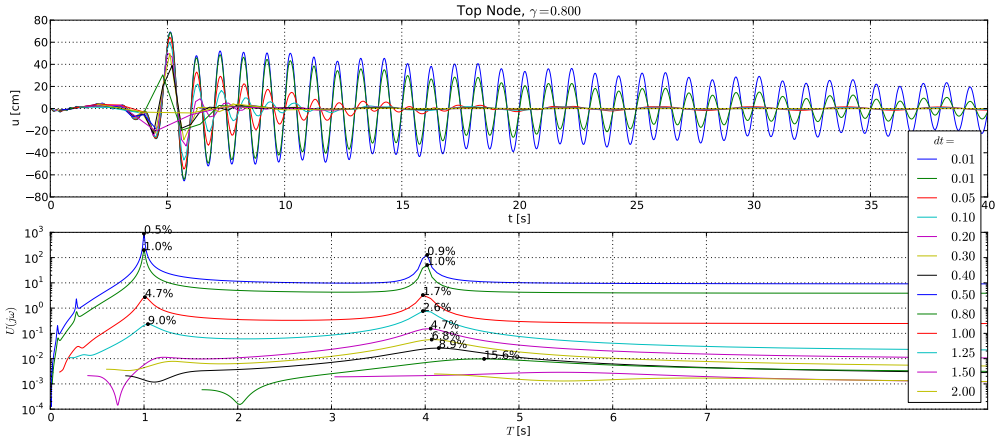


Figure 306.5: Verification: Dynamic solution advancement, Newmark method (p2a-newmark-08top)...

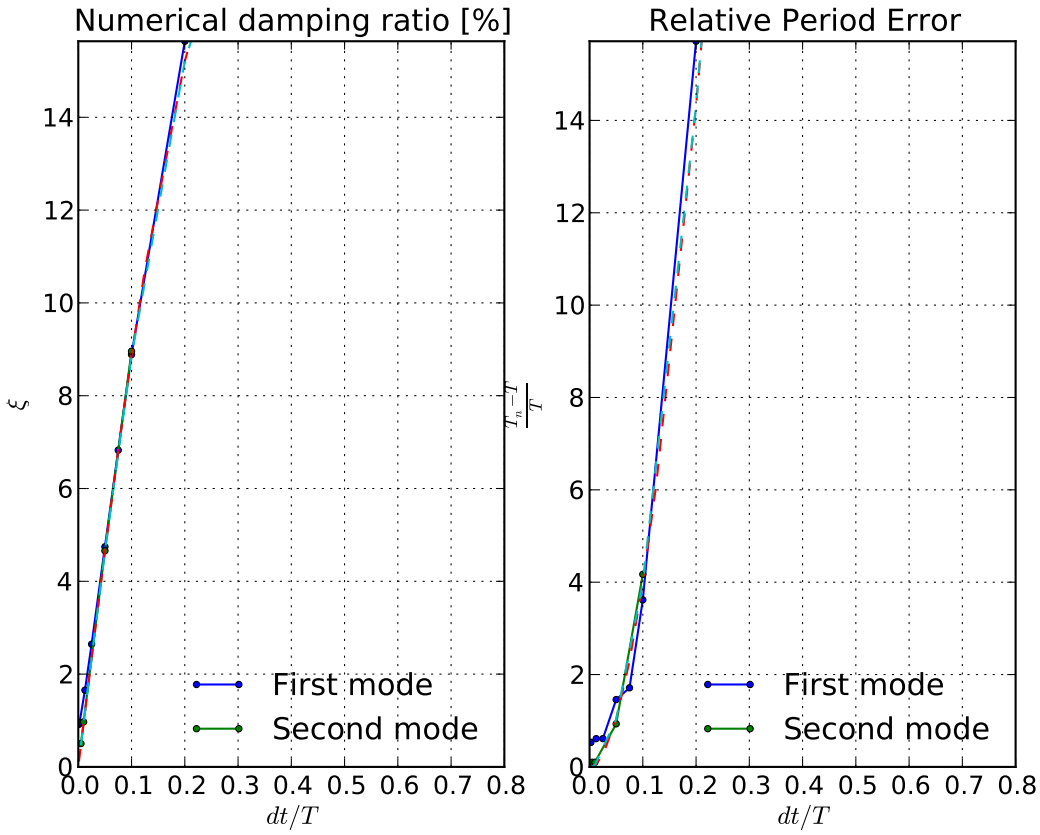


Figure 306.6: Verification: Dynamic solution advancement, Newmark method (p2a-newmark-08errors)...

```
dt gamma damping[\%] Th. damp.[\%] T shift [\%] Th. T. shift[\%]
================================================================================
0.0100 1.5000 3.398552 3.139529 0.00000000 -0.13145647
0.0050 0.5000 0.295322 -0.000000 0.00000000 -0.00822413
0.0500 0.5000 0.368976 0.000000 -1.01010101 -0.81712426
0.1250 0.5000 0.027518 0.000000 -4.95382032 -4.94456582
0.0050 0.5100 0.298179 0.015705 0.00000000 -0.00822660
0.0500 0.5100 0.395657 0.154550 -1.01010101 -0.81736312
0.1250 0.5100 0.358214 0.357055 -4.95382032 -4.94584070
0.0050 0.6000 0.346769 0.157054 0.00000000 -0.00847079
0.0500 0.6000 1.600134 1.545504 -1.01010101 -0.84101009
0.1250 0.6000 3.571434 3.570958 -5.04201681 -5.07208349
0.0050 0.7000 0.418286 0.314108 0.00000000 -0.00921077
0.0500 0.7000 3.113067 3.091044 -1.01010101 -0.91266978
0.1250 0.7000 7.130075 7.144404 -5.13036165 -5.45499548
0.0050 0.9000 0.625858 0.628215 0.00000000 -0.01217067
0.0500 0.9000 6.177709 6.182372 -1.01010101 -1.19934156
```

306.3.2 Verification Example Description.

A one degree of freedom (DOF) example was made to verify the Newmark and HHT algorithm for Real-ESSI simulator. The example was plot below in Fig.(306.7). The beam stiffness and the mass were designed to make the natural period to be 1 second. In the first loading stage, the beam was given a horizontal force to generate an initial displacement. By the way, at the top node, all DOFs were fixed except the DOF along initial displacement. Then, in the second loading stage, the beam start free vibration.

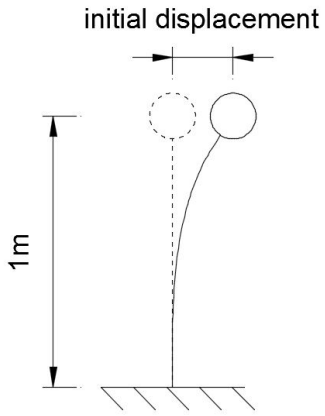


Figure 306.7: Verification example description.

The results were listed and plotted below.

Table 306.1: Verification results for the Newmark solution advancement algorithm.

dt	γ	Measured ξ	Analytic ξ	Measured T shift	Analytic T shift
0.005	0.5	0.0030	0.0000	0.0000	0.0001
0.01	0.5	0.0030	0.0000	0.0000	0.0003
0.05	0.5	0.0037	0.0000	0.0101	0.0082
0.1	0.5	0.0034	0.0000	0.0309	0.0321
0.005	0.6	0.0035	0.0016	0.0000	0.0001
0.01	0.6	0.0042	0.0031	0.0000	0.0003
0.05	0.6	0.0160	0.0155	0.0101	0.0084
0.1	0.6	0.0296	0.0295	0.0309	0.0329
0.005	0.7	0.0042	0.0031	0.0000	0.0001
0.01	0.7	0.0063	0.0063	0.0000	0.0004
0.05	0.7	0.0311	0.0309	0.0101	0.0091
0.1	0.7	0.0590	0.0590	0.0309	0.0356
0.005	0.8	0.0051	0.0047	0.0000	0.0001
0.01	0.8	0.0093	0.0094	0.0000	0.0004
0.05	0.8	0.0465	0.0464	0.0101	0.0103
0.1	0.8	0.0882	0.0886	0.0309	0.0399
0.005	0.9	0.0063	0.0063	0.0000	0.0001
0.01	0.9	0.0130	0.0126	0.0000	0.0005
0.05	0.9	0.0618	0.0618	0.0101	0.0120
0.1	0.9	0.1180	0.1181	0.0417	0.0460

Verification Results for Newmark Solution Advancement Algorithm. The Real-ESSI model fei/DSL files for the results above can be downloaded HERE.

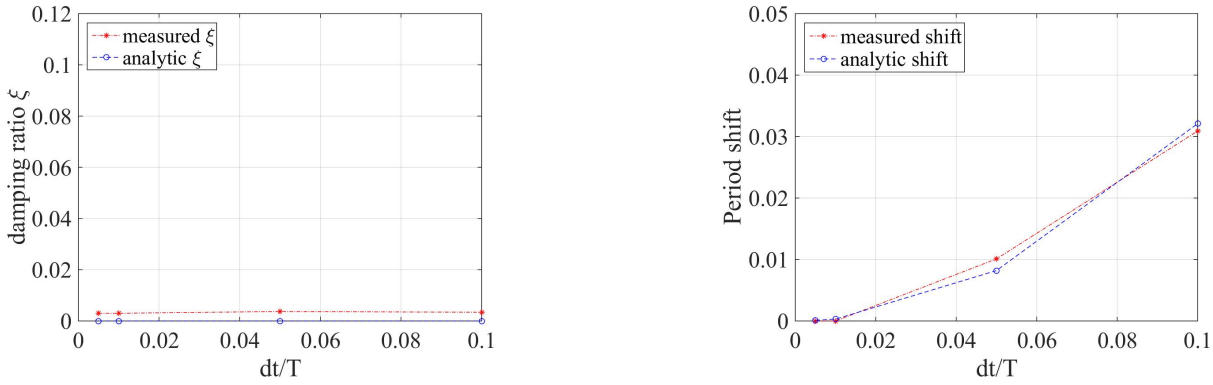


Figure 306.8: Comparison for Newmark algorithm with $\gamma = 0.5$. Damping ratio comparison, Period shift comparison.

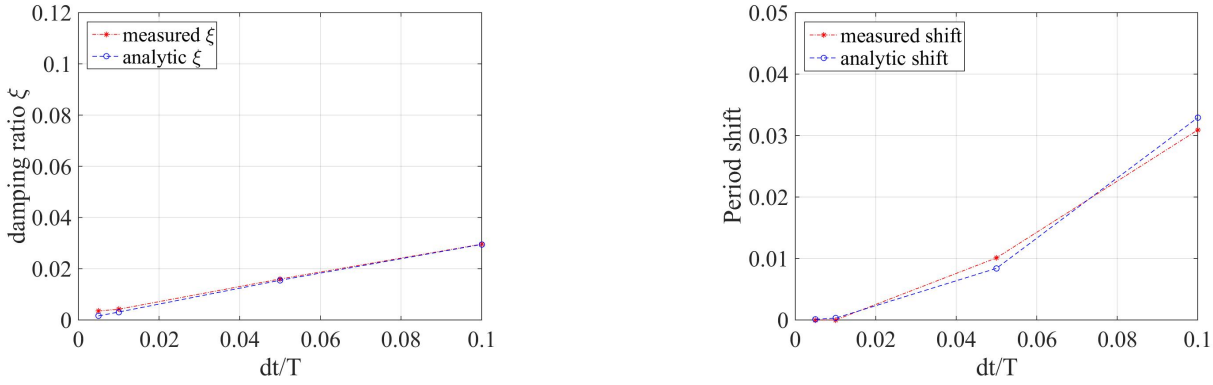


Figure 306.9: Comparison for Newmark algorithm with $\gamma = 0.6$. Damping ratio comparison, Period shift comparison.

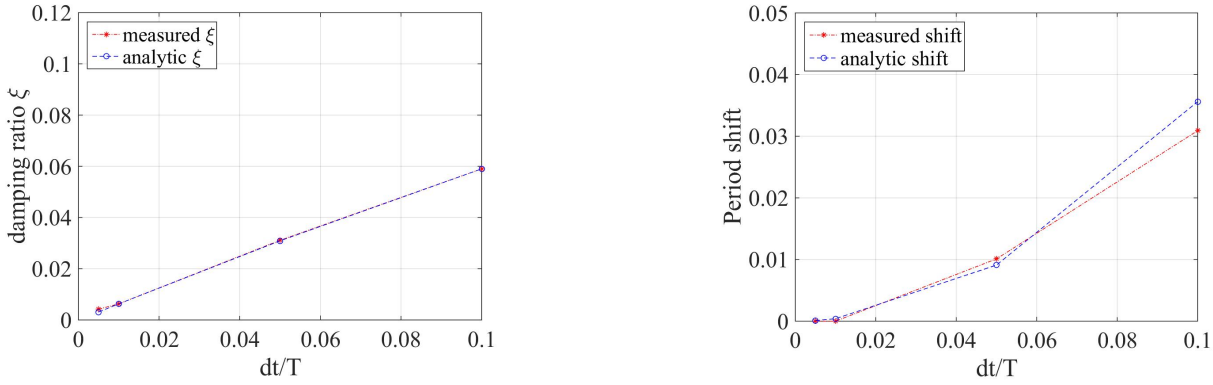


Figure 306.10: Comparison for Newmark algorithm with $\gamma = 0.7$. Damping ratio comparison, Period shift comparison.

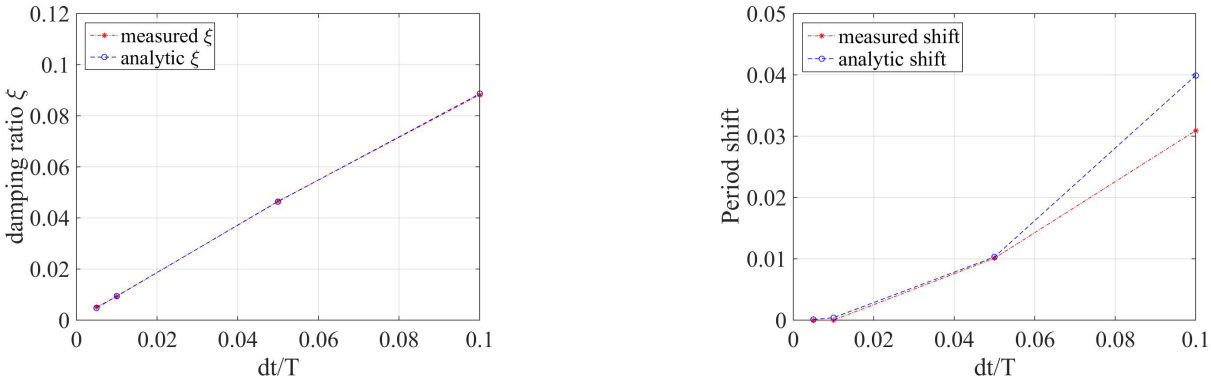


Figure 306.11: Comparison for Newmark algorithm with γ = 0.8. Damping ratio comparison, Period shift comparison.

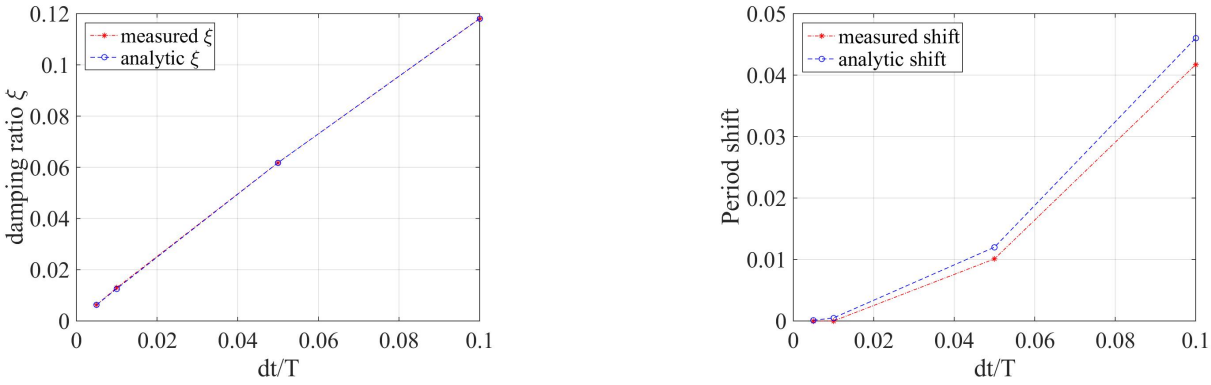


Figure 306.12: Comparison for Newmark algorithm with γ = 0.9. Damping ratio comparison, Period shift comparison.

306.3.3 Verification for Dynamic Solution Advancement, Hilber-Hughes-Taylor Method

(Hilber et al., 1977), (Hughes and Liu, 1978a) and (Hughes and Liu, 1978b)

If the parameters α, β and γ satisfy

$$-1/3 \leq \alpha \leq 0, \quad \gamma = \frac{1}{2}(1-2\alpha), \quad \beta = \frac{1}{4}(1-\alpha)^2 \tag{306.6}$$

it is unconditionally stable and second-order accurate (Argyris and Mlejnek, 1991; Hughes, 1987).

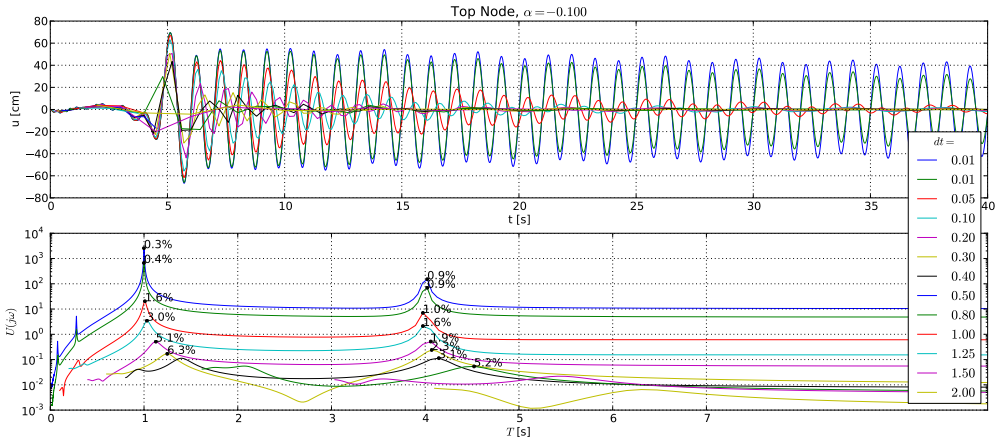


Figure 306.13: Verification: Dynamic solution advancement, Hilber-Hughes-Taylor method (p2a-hht-01top)...

Summary

```
dt alpha damping[%] Th. damp.[%] T shift [%] Th. T. shift[%]
=============================================================================================
      0.0050 -0.0000 0.295322 -0.000000 0.00000000 -0.00822413
      0.0100 -0.0000 0.302864 0.000000 0.00000000 -0.03289003
      0.0500 -0.0000 0.368976 0.000000 -1.01010101 -0.81712426
      0.1000 -0.0000 0.339004 0.000000 -3.09278351 -3.20749106
      0.2500 -0.0000 0.458456 -0.000000 -17.64705882 -17.96772753
      0.0050 -0.0100 0.295394 0.000004 0.00000000 -0.00846709
      0.0100 -0.0100 0.303170 0.000030 0.00000000 -0.03386091
      0.0500 -0.0100 0.357316 0.003645 -1.01010101 -0.84065236
      0.1000 -0.0100 0.376804 0.025924 -3.09278351 -3.29331088
      0.2500 -0.0100 1.026595 0.213468 -17.64705882 -18.29362011
      0.0050 -0.1000 0.295948 0.000031 0.00000000 -0.01032073
      0.0100 -0.1000 0.305522 0.000251 0.00000000 -0.04126821
      0.0500 -0.1000 0.305401 0.029968 -1.01010101 -1.02022480
      0.1000 -0.1000 0.432090 0.210613 -4.16666667 -3.95057551
      0.2500 -0.1000 1.807861 1.706433 -20.48192771 -20.93638414
      0.0050 -0.2000 0.296354 0.000050 0.00000000 -0.01167737
      0.0100 -0.2000 0.307270 0.000396 0.00000000 -0.04668953
```

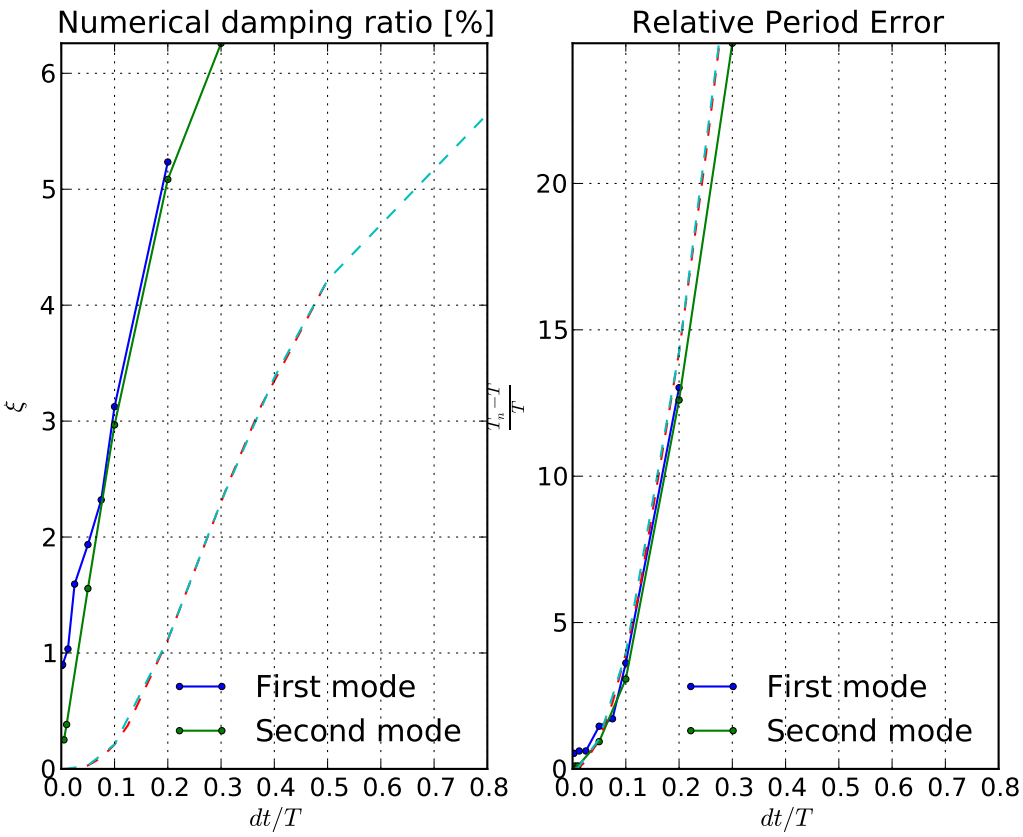


Figure 306.14: Verification: Dynamic solution advancement, Hilber-Hughes-Taylor method (p2a-hht-01errors)...

```
        0.0500 -0.2000 0.347865 0.047176 -1.01010101 -1.15178012
        0.1000 -0.2000 0.519421 0.328633 -4.16666667 -4.43495111
```

306.3.3.1 Verification Results for Hilber Hughes Taylor (HHT) Solution Advancement Algorithm.

The Real-ESSI model fei/DSL files for the results above can be downloaded HERE.

Table 306.2: Verification results for Hilber Hughes Taylor (HHT) solution advancement algorithm.

dt	α	Measured ξ	Analytic ξ	Measured T shift	Analytic T shift
0.005	-0.0	0.002953	0.000000	0.000000	0.000082
0.01	-0.0	0.003029	0.000000	0.000000	0.000329
0.05	-0.0	0.003690	0.000000	0.010101	0.008171
0.1	-0.0	0.003390	0.000000	0.030928	0.032075
0.005	-0.01	0.002954	0.000000	0.000000	0.000085
0.01	-0.01	0.003032	0.000000	0.000000	0.000339
0.05	-0.01	0.003573	0.000036	0.010101	0.008407
0.1	-0.01	0.003768	0.000259	0.030928	0.032933
0.005	-0.05	0.002957	0.000000	0.000000	0.000094
0.01	-0.05	0.003043	0.000001	0.000000	0.000374
0.05	-0.05	0.003227	0.000167	0.010101	0.009276
0.1	-0.05	0.009382	0.001184	0.030928	0.036111
0.005	-0.1	0.002959	0.000000	0.000000	0.000103
0.01	-0.1	0.003055	0.000003	0.000000	0.000413
0.05	-0.1	0.003054	0.000300	0.010101	0.010202
0.1	-0.1	0.004321	0.002106	0.041667	0.039506
0.005	-0.2	0.002964	0.000000	0.000000	0.000117
0.01	-0.2	0.003073	0.000004	0.000000	0.000467
0.05	-0.2	0.003479	0.000472	0.010101	0.011518
0.1	-0.2	0.005194	0.003286	0.041667	0.044350

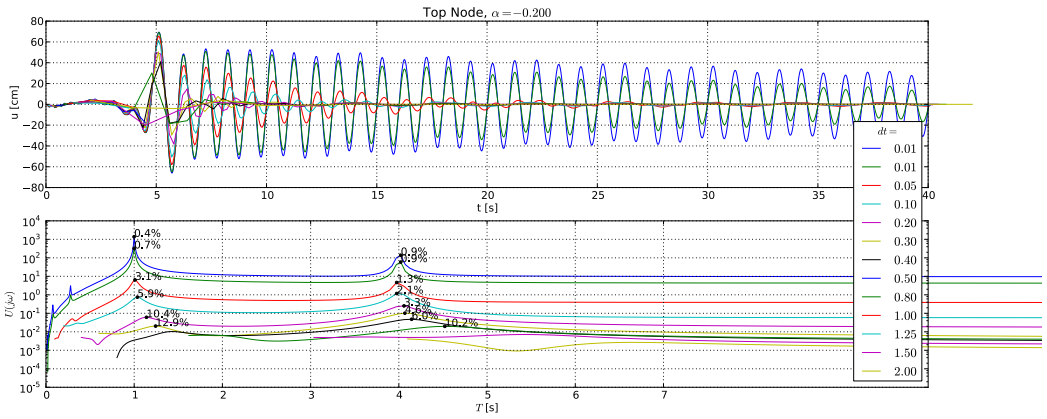


Figure 306.15: Verification: Dynamic solution advancement, Hilber-Hughes-Taylor method (p2a-hht-02top)...

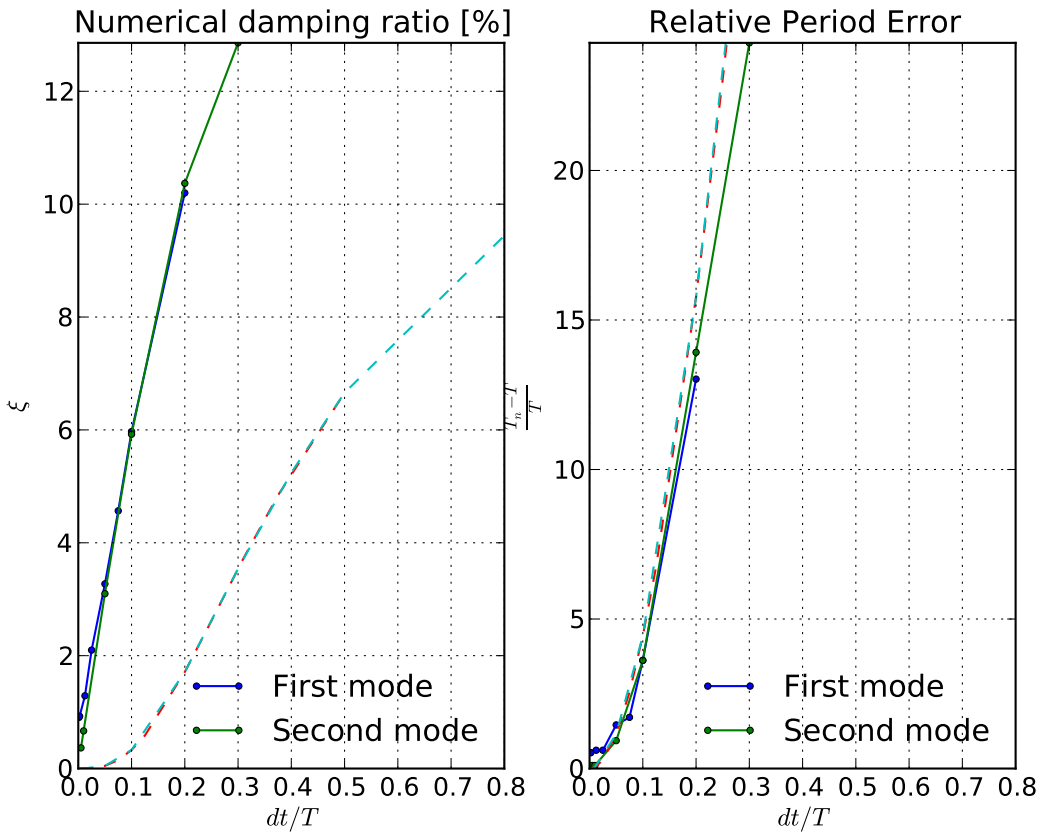


Figure 306.16: Verification: Dynamic solution advancement, Hilber-Hughes-Taylor method (p2a-hht-01errors)...

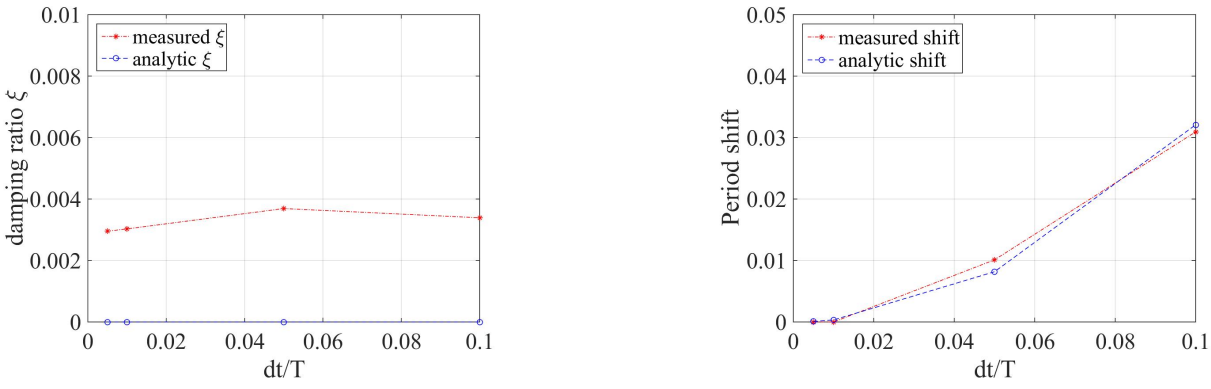


Figure 306.17: Comparison for HHT algorithm with $\alpha = -0.0$. Damping ratio comparison, Period shift comparison.

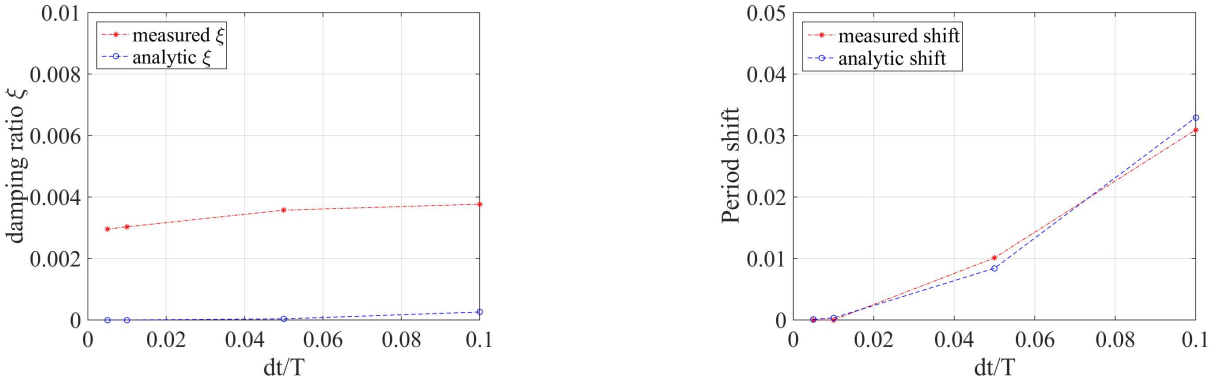


Figure 306.18: Comparison for HHT algorithm with $\alpha = -0.01$. Damping ratio comparison, Period shift comparison.

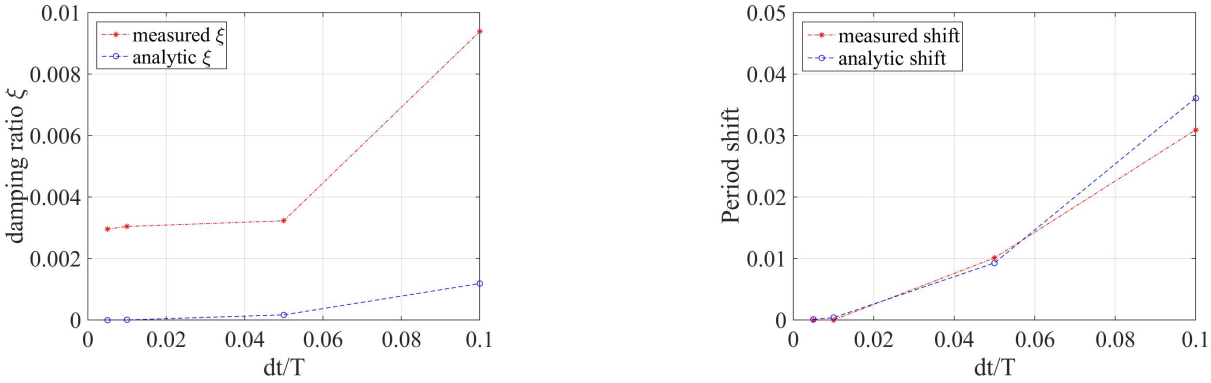


Figure 306.19: Comparison for HHT algorithm with $\alpha = -0.05$. Damping ratio comparison, Period shift comparison.

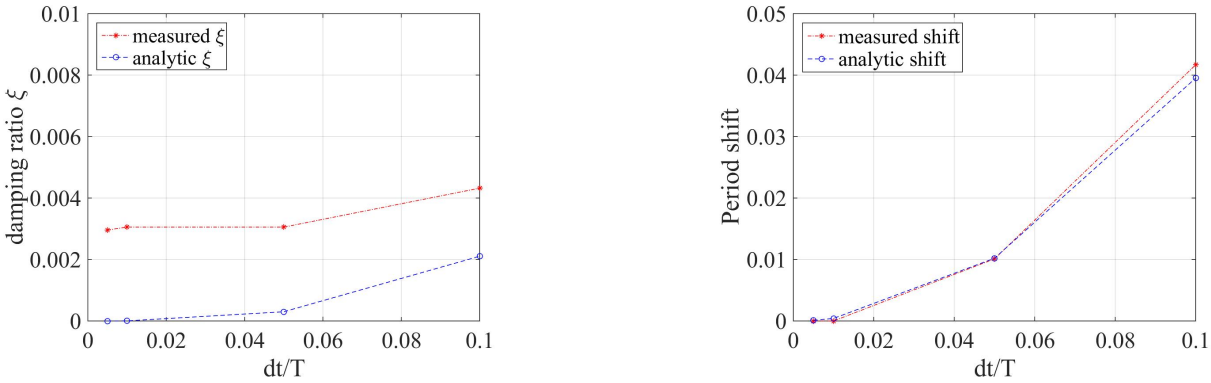


Figure 306.20: Comparison for HHT algorithm with $\alpha = -0.10$. Damping ratio comparison, Period shift comparison.

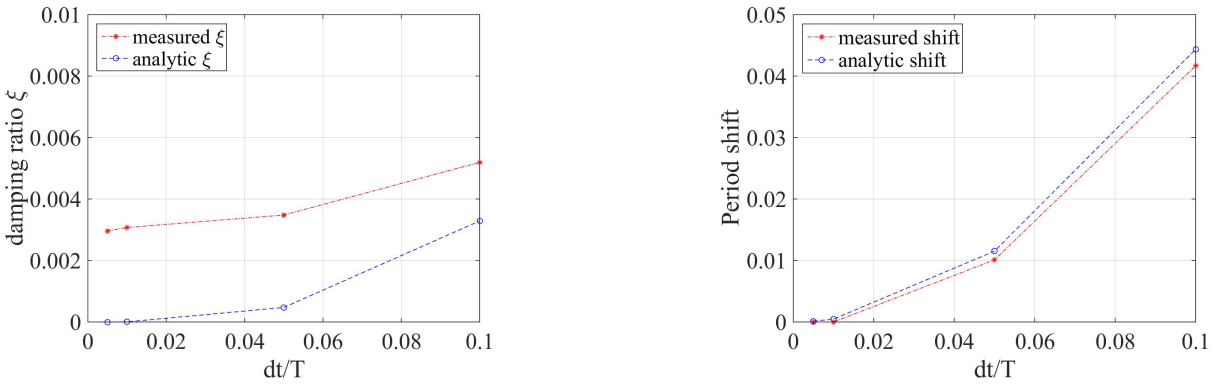


Figure 306.21: Comparison for HHT algorithm with $\alpha = -0.20$. Damping ratio comparison, Period shift comparison.

Chapter 307

Verification and Validation for Static and Dynamic Behavior of Single Phase, Solid Elements

(1989-1994-2011-2015-2017-2019-)

(In collaboration with Dr. Yuan Feng, and Prof. Han Yang)

307.1 Chapter Summary and Highlights

307.2 Verification of Static, Single Phase Solid Modeling and Simulation

307.2.1 Beam theory

This section provides basic beam theory that is used for verification solutions.

Problem description: Length=6m, Width=1m, Height=1m, F=100N, E=1E8Pa, $\nu = 0.0$. The force direction was shown in Figure (307.1).

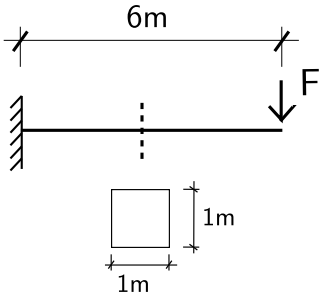


Figure 307.1: Problem description for cantilever beams.

The basic idea to calculate the shear deformation of a beam is

$$\delta = \frac{FL}{GA_v} \tag{307.1}$$

where A_v is the not the gross cross sectional area of the beam. A_v should be the shear area. Thus,

$$\kappa = \frac{A}{A_v} \tag{307.2}$$

where κ is the form factor, shear correction factor or shear deformation coefficient, A is the gross sectional area and A_v is the shear area of the section.

REWRITE, use bibtex! The history of κ value is long.

1. Timoshenko (1940)[1] define the form factor for rectangular section is 1.5.

2. Cowper (1970)[2] gave the formula for the form factor:

$$\kappa = \frac{12 + 11\nu}{10(1 + \nu)} \tag{307.3}$$

[1] Strength of materials, Timoshenko, Krieger Pub Co, 1940

[2] Cowper, G. R. "The shear coefficient in Timoshenko's beam theory." Journal of applied mechanics 33.2 (1966): 335-340.