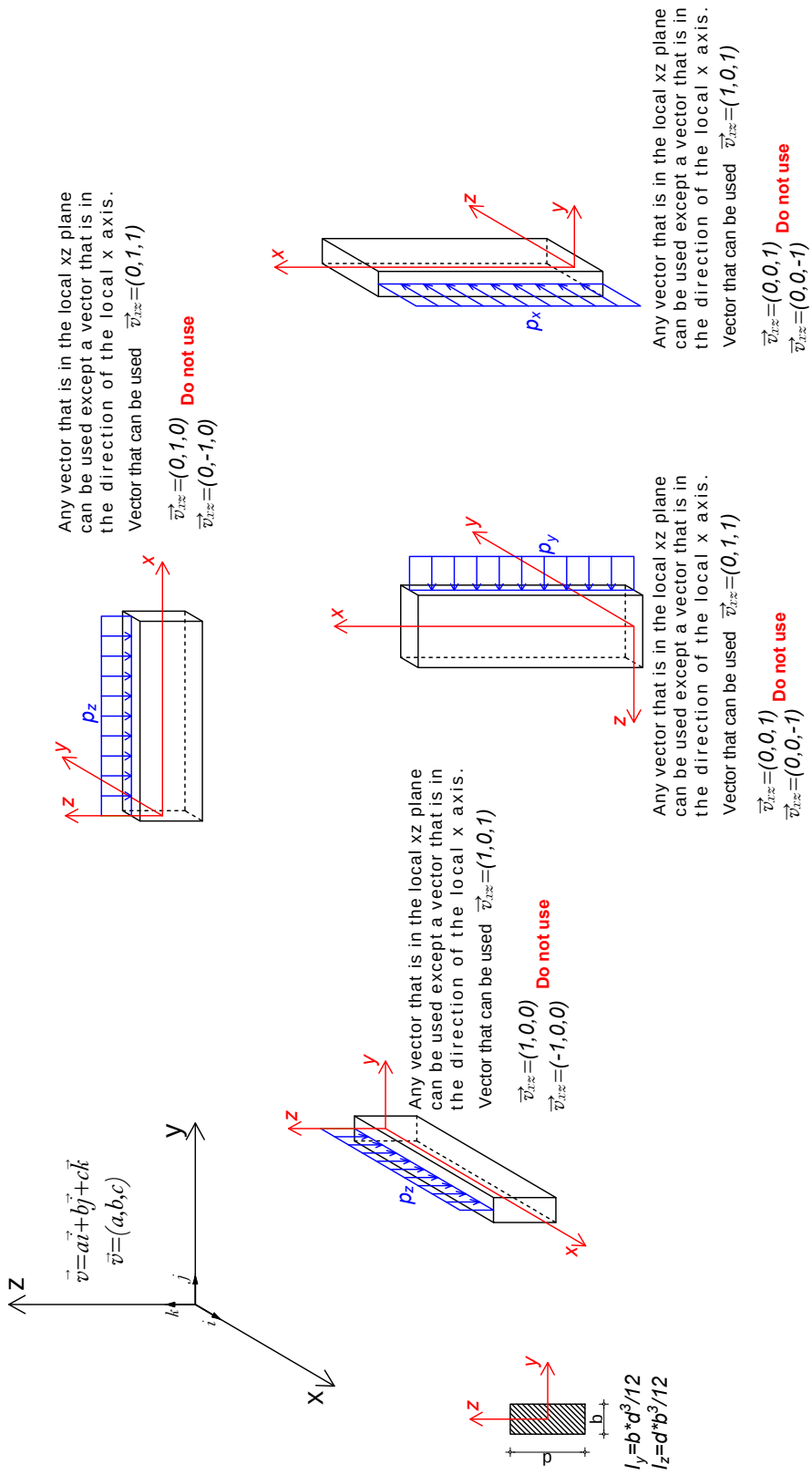


Figure 205.4: Beam Element, details of vector in X-Z plane.

205.3.4.54 Modeling, Finite Element: Large Displacement Elastic Beam–Column Element, with Corotational Transformation

The command is:

```
add element # <.> type beam_elastic_corotational with nodes (<.>, <.>)
    cross_section = <L^2>
    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> );
```

where

- `No (or #)<element_number>` is a unique element integer number (does not have to be sequential, any unique positive integer number can be used)
- `type beam_elastic` is the element type
- `with nodes (n1, n2)` are the 2 nodes (node numbers) defining this element
- `cross_section` is the cross section area, $[L^2]$
- `elastic_modulus` elastic modulus of the material which makes up the beam, $[F/L^2]$
- `shear_modulus` shear modulus of the material which makes up the beam, $[F/L^2]$
- `torsion_Jx` cross section polar (torsional) moment of inertia, $[L^4]$
- `bending_Iy` cross section moment of inertia about local y axis, $[L^4]$
- `bending_Iz` cross section moment of inertia about local z axis, $[L^4]$
- `mass_density` mass per unit volume of the material, $[M/L^3]$
- `xz_plane_vector` a vector which defines the orientation of the local (beam coordinate system) xz plane in global coordinates.
- `joint_1_offset` vector defining the rigid offset between end of beam and connection node 1, $[L]$
- `joint_2_offset` vector defining the rigid offset between end of beam and connection node 2, $[L]$

205.3.4.55 Modeling, Finite Element: Timoshenko Elastic Beam–Column Element

The command is:

```
add element # <.> type beam_elastic_Timoshenko with nodes (<.>, <.>)
    cross_section = <L^2>
    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>
    shear_correction_coefficient = <.>
    xz_plane_vector = (<.>, <.>, <.> )
    joint_1_offset = (<L>, <L>, <L> )
    joint_2_offset = (<L>, <L>, <L> );
```

where

- `No (or #)<element_number>` is a unique element integer number (does not have to be sequential, any unique positive integer number can be used)
- `type beam_elastic` is the element type
- `with nodes (n1, n2)` are the 2 nodes (node numbers) defining this element
- `cross_section` is the cross section area, $[L^2]$
- `elastic_modulus` elastic modulus of the material which makes up the beam, $[F/L^2]$
- `shear_modulus` shear modulus of the material which makes up the beam, $[F/L^2]$
- `torsion_Jx` cross section polar (torsional) moment of inertia, $[L^4]$
- `bending_Iy` cross section moment of inertia about local y axis, $[L^4]$
- `bending_Iz` cross section moment of inertia about local z axis, $[L^4]$
- `mass_density` mass per unit volume of the material, $[M/L^3]$
- `shear_correction_coefficient` a parameter for shear correction. When this parameter becomes very large, the Timoshenko beam element becomes Euler-Bernoulli beam. If not specifically calibrated, can use 1.0 for this parameter.
- `xz_plane_vector` a vector which defines the orientation of the local (beam coordinate system) xz plane in global coordinates.

- `joint_1_offset` vector defining the rigid offset between end of beam and connection node 1, $[L]$
- `joint_2_offset` vector defining the rigid offset between end of beam and connection node 2, $[L]$

Description of output by this element can be found in Section 206.8.4. more on this finite element can be found in Section 102.7 on Page 126 in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).

205.3.4.56 Modeling, Finite Element: Timoshenko Elastic Beam–Column Element with Directional Shear Correction Coefficients

The command is:

```
add element # <.> type beam_elastic_Timoshenko_directional with nodes (<.>, <.>)
    cross_section = <L^2>
    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>
    shear_correction_coefficient_y = <.>
    shear_correction_coefficient_z = <.>
    xz_plane_vector = (<.>, <.>, <.> )
    joint_1_offset = (<L>, <L>, <L> )
    joint_2_offset = (<L>, <L>, <L> );
```

where

- `No (or #)<element_number>` is a unique element integer number (does not have to be sequential, any unique positive integer number can be used)
- `type beam_elastic` is the element type
- `with nodes (n1, n2)` are the 2 nodes (node numbers) defining this element
- `cross_section` is the cross section area, $[L^2]$
- `elastic_modulus` elastic modulus of the material which makes up the beam, $[F/L^2]$
- `shear_modulus` shear modulus of the material which makes up the beam, $[F/L^2]$
- `torsion_Jx` cross section polar (torsional) moment of inertia, $[L^4]$
- `bending_Iy` cross section moment of inertia about local y axis, $[L^4]$
- `bending_Iz` cross section moment of inertia about local z axis, $[L^4]$
- `mass_density` mass per unit volume of the material, $[M/L^3]$
- `shear_correction_coefficient_y` parameter for shear correction about local y axis. When this parameter becomes very large, the Timoshenko beam element becomes Euler-Bernoulli beam. If not specifically calibrated, can use 1.0 for this parameter.

- `shear_correction_coefficient_z` parameter for shear correction about local z axis. When this parameter becomes very large, the Timoshenko beam element becomes Euler-Bernoulli beam. If not specifically calibrated, can use 1.0 for this parameter.
- `xz_plane_vector` a vector which defines the orientation of the local (beam coordinate system) xz plane in global coordinates.
- `joint_1_offset` vector defining the rigid offset between end of beam and connection node 1, $[L]$
- `joint_2_offset` vector defining the rigid offset between end of beam and connection node 2, $[L]$

Description of output by this element can be found in Section 206.8.4. more on this finite element can be found in Section 102.7 on Page 126 in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).

205.3.4.57 Modeling, Finite Element: Adding 1D Fiber to a Beam Cross Section

Fibers can be added to the fiber beam cross section.

The command is:

```
add fiber # <.> using material # <.> to section # <.> fiber_cross_section = <area> fiber_location = (<L>,<L>);
```

For example:

```
add fiber # 1 using material # 1 to section # 1 fiber_cross_section = 5*cm^2 fiber_location = (10*cm,10*cm);
```

adds a fiber number 1 to section number 1 at coordinates $y = 10\text{cm}$, $z = 10\text{cm}$ with cross section area of 5cm^2 using material number 1.

The material for fiber must be a uniaxial material, for example uniaxial_concrete02, uniaxial_elastic, uniaxial_steel01, and uniaxial_steel02.

205.3.4.58 Modeling, Finite Element: Adding Fiber Section to the Finite Element Model

Fiber section can be added to the finite element model.

The command is:

```
1  add section # <.> type FiberSection
2     TorsionConstant_GJ = <F*L^2);
```

where

- `TorsionConstant_GJ` provides a linear torsional stiffness to the element.

Fibers can be added to the section as described in section 205.3.4.57 on page 907.

The command is:

```
1 add fiber # <.> using material # <.> to section # <.> fiber_cross_section = <area>
2   fiber_location = (<L>,<L>);
```

where

- `fiber_cross_section` is the area of the fiber element. (Total cross section are is the sum of all fiber areas) $[L^2]$
- `fiber_location` location of the fiber in the beam local Y-Z plane.

205.3.4.59 Modeling, Finite Element: 3D Displacement Based Fiber Beam-Column Element

```
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> );
```

where

- `No (or #)<element_number>` is the unique element integer number (does not have to be sequential, any unique positive integer number can be used)
- `type BeamColumnDispFiber3d` is the element type
- `with nodes (n1, n2)` are the 2 nodes defining this element
- `number_of_integration_points` is number of integration points to be used along the beam element
- `section_number` is the number of predefined section
- `mass_density` mass per unit volume of the material, $[M/L^3]$
- `xz_plane_vector` unit vector which defines the orientation of the web of the beam in global coordinates.
- `joint_1_offset` vector defining the rigid offset between end of beam and connection node 1, $[L]$
- `joint_2_offset` vector defining the rigid offset between end of beam and connection node 2, $[L]$

Description of output by this element can be found in Section 206.8.6,

205.3.4.60 Modeling, Finite Element: 3D Displacement Based Fiber Beam-Column Element with Corotational Coordinate Transformation

```
add element # <.> type BeamColumnDispFiber3d_Corotational 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> );
```

where

- `No (or #)<element_number>` is the unique element integer number (does not have to be sequential, any unique positive integer number can be used)
- `type BeamColumnDispFiber3d_Corotational` is the element type
- `with nodes (n1, n2)` are the 2 nodes defining this element
- `number_of_integration_points` is number of integration points to be used along the beam element
- `section_number` is the number of predefined section
- `mass_density` mass per unit volume of the material, $[M/L^3]$
- `xz_plane_vector` unit vector which defines the orientation of the web of the beam in global coordinates.
- `joint_1_offset` vector defining the rigid offset between end of beam and connection node 1, $[L]$
- `joint_2_offset` vector defining the rigid offset between end of beam and connection node 2, $[L]$

Description of output by this element can be found in section 206.8.6.

The co-rotational formulation used in this element is based on Crisfield (1990).

205.3.4.61 Modeling, Finite Element: 3DOF+6DOF=9DOF Beam-Column Element

```
add element # <.> type beam_9dof_elastic
    with nodes (<.>, <.>)
    cross_section = <L^2>
    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> );
```

where

- `No (or #)<element_number>` is the unique element integer number (does not have to be sequential, any unique positive integer number can be used)
- `type beam_9dof_elastic` is the element type
- `with nodes (n1, n2)` are the 2 nodes defining this element, where the first node (n1) is the one with 3 DOFs and the second (n2) is the one with 6 DOFs
- `cross_section` is the cross section area, $[L^2]$
- `elastic_modulus` elastic modulus of the material which makes up the beam, $[F/L^2]$
- `shear_modulus` shear modulus of the material which makes up the beam, $[F/L^2]$
- `torsion_Jx` cross section polar (torsional) moment of inertia, $[L^4]$
- `bending_Iy` cross section moment of inertia about local y axis, $[L^4]$
- `bending_Iz` cross section moment of inertia about local z axis, $[L^4]$
- `mass_density` mass per unit volume of the material, $[M/L^3]$
- `xz_plane_vector` unit vector which defines the orientation of the web of the beam in global coordinates.
- `joint_1_offset` vector defining the rigid offset between end of beam and connection node 1, $[L]$
- `joint_2_offset` vector defining the rigid offset between end of beam and connection node 2, $[L]$

This finite element has only 3DOFs (translations) at the first node, and full 6DOFs at the other, second node. Due to missing rotational stiffness on first, 3DOF node, this beam has zero torsional stiffness.

This element is useful for connection of solid (3DOFs per node) and structural (6DOFs per node) elements It his beam element is used on its own, DOF that corresponds to torsion of the second node (DOF number 7), should be fixed as this beam does not provide that stiffness.

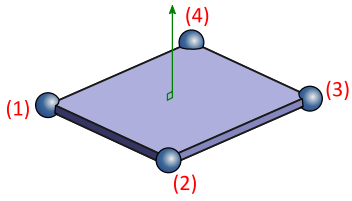
More on this finite element can be found in Section 102.8 on Page 129 in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).

205.3.4.62 Modeling, Finite Element: 4 Node ANDES Shell with Drilling DOFs

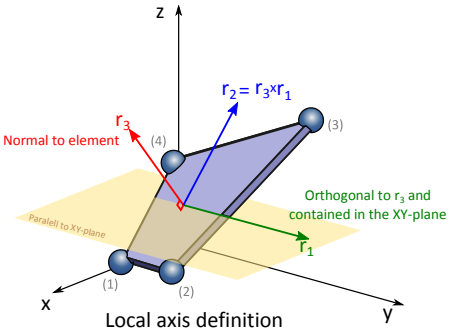
ANDES based 3D shell element including drilling degrees of freedom. Made up by patching together 4 ANDES shell triangle elements (and then averaging two and two squares made up two and two triangles).

The command is:

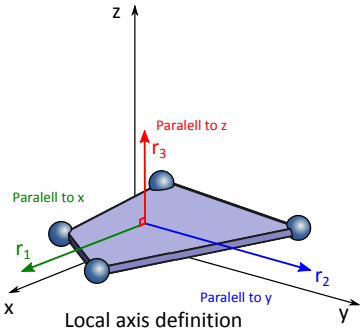
```
add element # <.> type 4NodeShell_ANDES
    with nodes (<.>, <.>, <.>)
    use material # <.>
    thickness = <L> ;
```



Node numbering
is counter-clockwise



Local axis definition
in general



Local axis definition
if r_3 is nearly vertical

- `No (or #)<element_number>` is the unique element integer number (does not have to be sequential, any unique positive integer number can be used)
- `material # <.>` number of a previously defined material. (see `add material ...`)
- `thickness` shell thickness, $[L]$

Description of output by this element can be found in Section 206.8.5.

More on this finite element can be found in Section 102.10 on page 135 in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).

205.3.4.63 Modeling, Finite Element: 3 Node ANDES Shell with Drilling DOFs

```
add element # <.> type 3NodeShell_ANDES
    with nodes (<.>, <.>, <.>)
    use material # <.>
    thickness = <L> ;
```

- No (or #)<element_number> is the unique element integer number (does not have to be sequential, any unique positive integer number can be used)

205.3.4.64 Modeling, Finite Element: 4 Node Shell NLDKGQ, or 4 Node Shell Xin-Zheng-Lu

This is a 3D quadrilateral shell element with membrane and drill DOFs based on the theory of generalized conforming element. This element accounts for the geometric nonlinearity of large deformation using a simplified version of updated Lagrangian formulation, where nodal coordinates are updated in each step, however strains and stresses are still calculated with reference to the original, undeformed system. It can be used together with elastic or inelastic sections. This element was originally developed by Professor Xin-Zheng Lu (Tsinghua University) and his students.

The command is:

```
add element # <.> type 4NodeShell_NLDKGQ
    with nodes (<.>, <.>, <.>, <.>)
    section_number = <.>;
```

It can also be called using the alternative command:

```
add element # <.> type 4NodeShell_XinZhengLu_Tsinghua
    with nodes (<.>, <.>, <.>, <.>)
    section_number = <.>;
```

- `No (or #)<element_number>` is the unique element integer number (does not have to be sequential, any unique positive integer number can be used)
- `with nodes (n1, n2, n3, n4)` are the 4 nodes defining this element
- `section_number` is the number of predefined shell cross section, described on page 916.

205.3.4.65 Modeling, Finite Element: Inelastic Layered Shell Section

This command is used to add a layered shell section. The section is made up of a number of layers with different thicknesses and different material properties (i.e., concrete layers or rebar layers). This type of section is used together with plane stress materials and shell elements.

The command is:

```
add section # <.> type LayeredShellFiber
    number_of_layers = <.>
    thickness_array = "<.>,<.>..."
    with material # "<.>,<.>..."
    thickness_scale_unit = <L>
    outofplane_shear_modulus = <F/L^2>;
```

where

- `number_of_layers` is the number of layers that the section has
- `thickness_array` is the relative thickness of each layer
- `with material #` is the material tag of each layer, only plane stress materials can be used here, see pages 878 and 879.
- `thickness_scale_unit` is the total thickness of the section
- `outofplane_shear_modulus` is the out-of-plane shear modulus of the section

205.3.4.66 Modeling, Finite Element: ElasticMembranePlaneStress Element (to be removed!)

NOTE: this element is being removed, and will not be available after Real ESSI version 19.07 (current). This is a 2D finite element, and we only maintain 3D finite elements. This element is replaced by a 3D 27 node elastic and/or elastic-plastic wall/plate/shell brick element.

The command is:

```
add element No (or #) <element_number>
    type ElasticMembranePlaneStress
    with nodes (n1, n2, n3, n4)
    use material No (or #) <material_number>
    thickness = <L> ;
```

where

- `No (or #)<element_number>` is a unique element integer number (does not have to be sequential, any unique positive integer number can be used).
- `type ElasticMembranePlaneStress` is the element type.
- `with nodes (n1, n2, n3, n4)` are the 4 nodes (node numbers) defining this element.
- `use material No (or #)` is the material number for linear elastic material that makes up the element.
- `thickness` is the thickness of the membrane.

205.3.4.67 Modeling, Finite Element: InelasticMembranePlaneStress Element (to be removed!)

NOTE: this element is being removed, and will not be available after Real ESSI version 19.07 (current). This is a 2D finite element, and we only maintain 3D finite elements. This element is replaced by a 3D 27 node elastic or elastic-plastic wall/plate/shell brick element or elastic-plastic shell element.

The command is:

```
add element No (or #) <element_number>
    type InelasticMembranePlaneStress
    with nodes (n1, n2, n3, n4)
    use material No (or #) <material_number>
    ;
```

where

- `No (or #)<element_number>` is a unique element integer number (does not have to be sequential, any unique positive integer number can be used).
- `type InelasticMembranePlaneStress` is the element type.
- `with nodes (n1, n2, n3, n4)` are the 4 nodes (node numbers) defining this element.
- `use material No (or #)` is the material number for inelastic material that makes up the element. Since this is a plane stress element, material needs to have plane stress constitutive integration algorithm available. In addition, this material should specify thickness of the element. Different layers and their thicknesses for different materials (for example concrete and steel) will be defined within material definition. `PlaneStressLayeredMaterial` is a material of this type.

205.3.4.68 Modeling, Finite Element: SuperElementLinearElasticImport

The command is:

```
add element No (or #) <element_number>
   type SuperElementLinearElasticImport
   with hdf5_file = <string>
   ;
```

where

- `No (or #)<element_number>` is a unique element integer number (does not have to be sequential, any unique positive integer number can be used).
- `type SuperElementLinearElasticImport` is the element type.
- `hdf5_file` specifies the HDF5 filename of the SuperElement with SuperElement data. The HDF5 file should contain the following datasets:
  - `Node` dataset within HDF5 file is organized in a column (a 1D dataset), and it specifies the node tags/numbers of nodes that make up the SuperElement.
  - `DofList` dataset within HDF5 file is a organized in a column, and it specifies the number of DOFs per each `Node`. For example if nodes are representing structural elements, they usually have 6 DOFs per node, while solids will have 3 DOFs per node. `DofList` dataset has to have the same number of entries as `Node` dataset, as each entry in `DofList` corresponds to one node from `Node` dataset.
  - `MassMatrix` is a matrix, that sets masses/numbers for a mass matrix of the SuperElement.
  - `StiffnessMatrix` is a matrix, that sets stiffness/numbers for a stiffness matrix of the SuperElement.
  - `ConnectNode` dataset within HDF5 file is organized in a column (a 1D dataset), and it specifies the node tags/numbers of nodes that are going to be connected to Real-ESSI mesh.
  - `ConnectNodeCoordinate` dataset within HDF5 file is organized in a matrix (a 2D dataset), and it specifies the nodal coordinates for nodes that are going to be connected to Real-ESSI mesh. Since each node has 3 coordinates, the length of `ConnectNodeCoordinate` is the same as the length of `ConnectNode` and each line has three entries, for X, Y and Z coordinates of given node.

In addition to the minimum dataset requirements above, users can get more output from Real-ESSI:

- Results for individual finite elements (internal forces, etc.), can be obtained if node, DofList, mass matrix and stiffness matrix for each finite element within the super element are provided.
- Graphical post-processing can be obtained if coordinates for all nodes and their connectivity into finite elements are provided (a mesh data).

205.3.4.69 Modeling, Finite Element: 8 Node Brick Element

The command is:

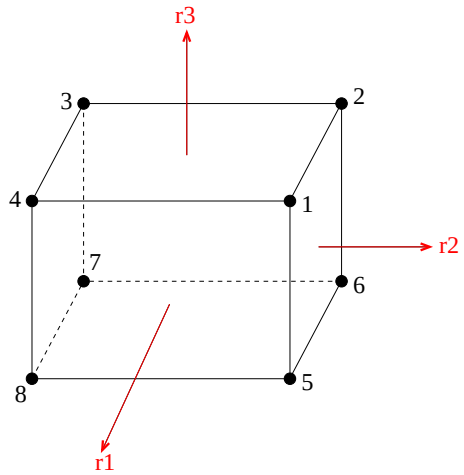
```
add element # <element_number> type 8NodeBrick
    using <.> Gauss points each direction
    with nodes (n1, n2, n3, n4, n5, n6, n7, n8)
    use material No (or #) <material_number>;
```

and/or;

```
add element # <element_number> type 8NodeBrick
    with nodes (n1, n2, n3, n4, n5, n6, n7, n8)
    use material No (or #) <material_number>;
```

where:

- `No (or #)<element_number>` is the unique element integer number (does not have to be sequential, any unique positive integer number can be used)
- `type 8NodeBrick` is the element type.
- `with nodes (n1, n2, n3, n4, n5, n6, n7, n8)` are the 8 nodes for this element, in the order as per figure below



- `using <.> Gauss points each direction` is the number of Gauss points to be used in each direction (r1, r2, and r3) for integration of finite element matrices (mass and stiffness). There can be from 1 to 6 Gauss points used (uniformly) in each direction (r1, r2, and r3). Command for the brick finite element (above) without number of Gauss points control is kept for back compatibility.

For 8 node bricks 2 Gauss points are used in each direction ($2 \times 3 \times 3$), while for 20 nodes, 8-20 node and 8-27 node bricks 3 Gauss points are used in each direction ($3 \times 3 \times 3$),

- `use material No (or #)` is the material number which makes up the element (nonlinear elastic or elastic-plastic material properties for each integration (Gauss) point will evolve independently as the element deforms). Use LT version with LT materials.

Description of output by this element can be found in section 206.8.2.

More on this finite element can be found in Section 102.4.1 on page 114 in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).

205.3.4.70 Modeling, Finite Element: 20 Node Brick Element

The command is:

```
add element No (or #) <element_number> type 20NodeBrick
   using <.> Gauss points each direction
   with nodes (n1, n2, n3, n4, n5, n6, n7, n8,
               n9, n10, n11, n12, n13, n14, n15, n16,
               n17, n18, n19, n20 )
   use material No (or #) <material_number>;
```

and/or

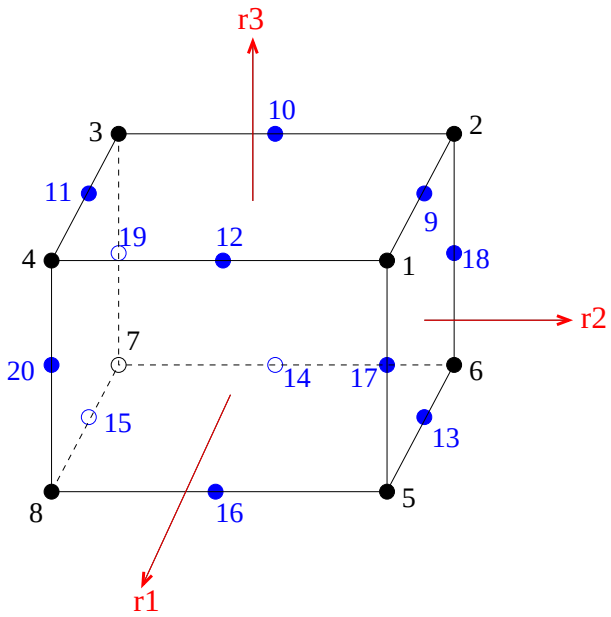
```
add element No (or #) <element_number> type 20NodeBrick
   with nodes (n1, n2, n3, n4, n5, n6, n7, n8,
               n9, n10, n11, n12, n13, n14, n15, n16,
               n17, n18, n19, n20 )
   use material No (or #) <material_number>;
```

where:

- `No (or #)<element_number>` is the unique element integer number (does not have to be sequential, any unique positive integer number can be used)
- `type 20NodeBrick` is the element type. `20NodeBrick_elastic` can be used if elastic material is used. It this case, the stiffness and mass matrices will not be updated at each step.
- `with nodes (n1, n2, n3, n4, n5, n6, n7, n8, n9, n10, n11, n12, n13, n14, n15, n16, n17, n18, n19, n20)` are the 20 nodes for this element, written in the order defined as per figure below
- `using <.> Gauss points each direction` is the number of Gauss points to be used in each direction (r1, r2, and r3) for integration of finite element matrices (mass and stiffness). There can be from 1 to 6 Gauss points used (uniformly) in each direction (r1, r2, and r3). Command for the brick finite element (above) without number of Gauss points control is kept for back compatibility. For 8 node bricks 2 Gauss points are used in each direction ($2 \times 3 \times 3$), while for 20 nodes, 8-20 node and 8-27 node bricks 3 Gauss points are used in each direction ($3 \times 3 \times 3$),
- `use material No (or #)` is the material number which makes up the element (nonlinear elastic or elastic-plastic material properties for each integration (Gauss) point will evolve independently as the element deforms)

Description of output by this element can be found in Section 206.8.2.

More on this finite element can be found in Section 102.4.3 on page 116 in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).



205.3.4.71 Modeling, Finite Element: 27 Node Brick Element

The command is:

```
add element # <element_number>
           type 27NodeBrick
           using <.> Gauss points each direction
           with nodes (n1, n2, n3, n4, n5, n6, n7, n8,
                      n9, n10, n11, n12, n13, n14, n15, n16,
                      n17, n18, n19, n20, n21, n22, n23,
                      n124, n25, n26, n27 )
           use material # <material_number>;
```

and/or

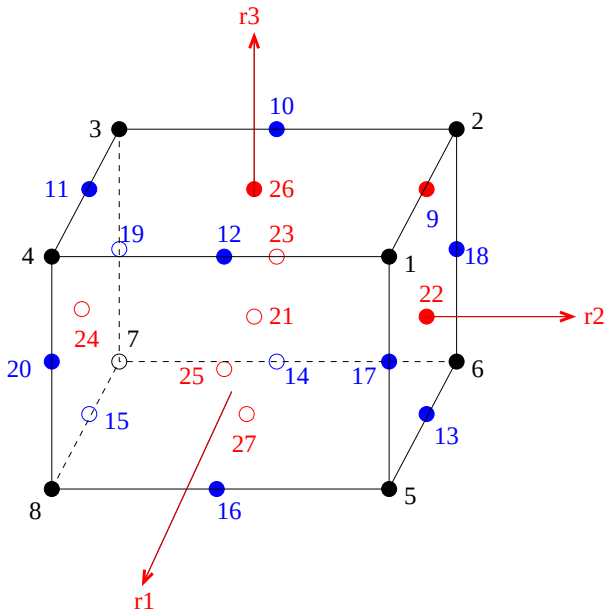
```
add element # <element_number>
           type 27NodeBrick
           with nodes (n1, n2, n3, n4, n5, n6, n7, n8,
                      n9, n10, n11, n12, n13, n14, n15, n16,
                      n17, n18, n19, n20, n21, n22, n23,
                      n124, n25, n26, n27 )
           use material # <material_number>;
```

where:

- `No (or #)<element_number>` is the unique element integer number (does not have to be sequential, any unique positive integer number can be used)
- `type 27NodeBrick` is the element type.
- `with nodes (n1, n2, n3, n4, n5, n6, n7, n8,`
  `n9, n10, n11, n12, n13, n14, n15, n16, n17, n18, n19, n20)`
  `n21, n22, n23, n24, n25, n26, n27)` are the 27 nodes for this element, written in the order defined as per this figure
- `using <.> Gauss points each direction` is the number of Gauss points to be used in each direction (r1, r2, and r3) for integration of finite element matrices (mass and stiffness). There can be from 1 to 6 Gauss points used (uniformly) in each direction (r1, r2, and r3). Command for the brick finite element (above) without number of Gauss points control is kept for back compatibility. For 8 node bricks 2 Gauss points are used in each direction ($2 \times 3 \times 3$), while for 20 nodes, 8-20 node and 8-27 node bricks 3 Gauss points are used in each direction ($3 \times 3 \times 3$),
- `use material No (or #)` is the material number which makes up the element (nonlinear elastic and/or elastic-plastic material properties for each integration (Gauss) point will evolve independently as the element deforms).

Description of output by this element can be found in Section 206.8.2.

More on this finite element can be found in Section 102.4.4 on page 118 in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).



205.3.4.72 Modeling, Finite Element: Variable 8-27 Node Brick Element

The command is:

```
add element No (or #) <element_number> type variable_node_brick_8_to_27
          using <.> Gauss points each direction
          with nodes (n1, n2, n3, n4, n5, n6, n7, n8,
                      n9, n10, n11, n12, n13, n14, n15, n16,
                      n17, n18, n19, n20, n21, n22, n23, n24, n25, n26, n27)
          use material No (or #) <material_number>;
```

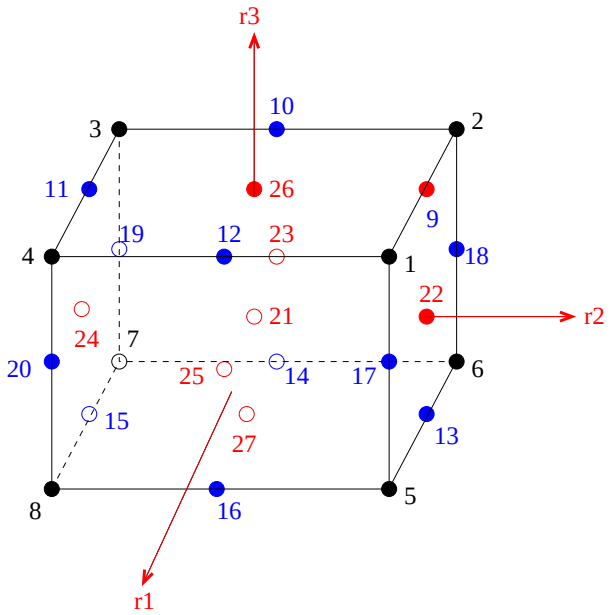
and/or

```
add element No (or #) <element_number> type variable_node_brick_8_to_27
          using <.> Gauss points each direction
          with nodes (n1, n2, n3, n4, n5, n6, n7, n8,
                      n9, n10, n11, n12, n13, n14, n15, n16,
                      n17, n18, n19, n20, n21, n22, n23, n24, n25, n26, n27)
          use material No (or #) <material_number>;
```

where:

- `No (or #)<element_number>` is the unique element integer number (does not have to be sequential, any unique positive integer number can be used)
- `type variable_node_brick_8_to_27` is the element type
- `with nodes (n1, n2, n3, n4, n5, n6, n7, n8,`
  `n9, n10, n11, n12, n13, n14, n15, n16, n17, n18, n19, n20,`
  `n21, n22, n23, n24, n25, n26, n27)`
  are the 8 to 27 nodes for this element, written in the order defined as per this figure. Nodes 1-8 are obligatory, while any other nodes can be used but do not have to, the element will automatically pick proper shape functions. This element is good for transitions in meshing.
- `using <.> Gauss points each direction` is the number of Gauss points to be used in each direction (r1, r2, and r3) for integration of finite element matrices (mass and stiffness). There can be from 1 to 6 Gauss points used (uniformly) in each direction (r1, r2, and r3). Command for the brick finite element (above) without number of Gauss points control is kept for back compatibility. For 8 node bricks 2 Gauss points are used in each direction ($2 \times 3 \times 3$), while for 20 nodes, 8-20 node and 8-27 node bricks 3 Gauss points are used in each direction ($3 \times 3 \times 3$),
- `material No (or #)` is the material number which makes up the element (nonlinear elastic and/or elastic-plastic material properties for each integration (Gauss) point will evolve independently as the element deforms)

Description of output by this element can be found in Section 206.8.2.



205.3.4.73 Modeling, Finite Element: 8 Node Brick u-p Element

The command is:

```
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^3*T/M>
    k_y = <L^3*T/M>
    k_z = <L^3*T/M>
    K_s = <F/L^2>
    K_f = <F/L^2>;
```

and/or

```
add element # <.> type 8NodeBrick_up
    with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>)
    use material # <.>
    porosity = <.>
    alpha = <.>
    rho_s = <M/L^3>
    rho_f = <M/L^3>
    k_x = <L^3*T/M>
    k_y = <L^3*T/M>
    k_z = <L^3*T/M>
    K_s = <F/L^2>
    K_f = <F/L^2>;
```

where:

- `No (or #)<element_number>` is the unique element integer number that does not have to be sequential, any unique positive integer number can be used.
- `type 8NodeBrick_up` is the element type/name.
- `with nodes (n1, n2, n3, n4, n5, n6, n7, n8)` are the 8 nodes for this element, is specified order.
- `using <.> Gauss points each direction` is the number of Gauss points to be used in each direction (r1, r2, and r3) for integration of finite element matrices (mass and stiffness). There can be from 1 to 6 Gauss points used (uniformly) in each direction (r1, r2, and r3). Command for the brick finite element (above) without number of Gauss points control is kept for back compatibility.

For 8 node bricks 2 Gauss points are used in each direction ($2 \times 3 \times 3$), while for 20 nodes, 8-20 node and 8-27 node bricks 3 Gauss points are used in each direction ($3 \times 3 \times 3$).

- `use material No (or #)` is the material number which makes up the element (nonlinear elastic or elastic-plastic material properties for each integration (Gauss) point will evolve independently as the element deforms). Use LT version with LT materials.
- `porosity` is the porosity ($n = V_{voids}/V_{total}$) of material in this element.
- `alpha` is the parameter controlling level of effective stress analysis. For soils, usually $\alpha = 1$ is used, while for other materials (saturated concrete, bone material, etc.) lower values are used ($0 \leq \alpha \leq 1$).
- `rho_s` is the density of particles of the solid phase. It is important to note that this is a density of the actual mineral that makes up solid particles!.
- `rho_f` is the density of pore fluid. It is usually density of water, however, for unsaturated and partially saturated materials, this density will be different, as described in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).
- `k_x` is the permeability in the x direction (global x) of the element. It is also important to note about the units used for permeability, as noted below. With isotropic permeability, usually the case, $k_x = k_y = k_z$.
- `k_y` is the permeability in the y direction (global y) of the element. It is also important to note about the units used for permeability, as noted below. With isotropic permeability, usually the case, $k_x = k_y = k_z$.
- `k_z` is the permeability in the z direction (global z) of the element. It is also important to note about the units used for permeability, as noted below. With isotropic permeability, usually the case, $k_x = k_y = k_z$.
- `K_s` is the bulk modulus of the soil phase particles. It is important to note that this is a bulk modulus of the actual mineral that makes up solid particles!
- `K_f` is the bulk modulus of the fluid phase that is found in porous material pores. It is usually bulk modulus of the fluid (physical value of the bulk modulus of fluid, for example water), however, for unsaturated and partially saturated materials, this density is a density of a mixture, as described in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).

Note: the permeability k_x, k_y, k_z is used with dimensions of $[length]^3[time]/[mass]$, which is different from the usual soil mechanics convention, where the permeability has the dimension of velocity, i.e. $[length]/[time]$. Their values are related by $k = K/\rho_f g$, where g is the gravitational acceleration at which the permeability is measured.

More on theory for this finite element can be found in Section 102.12.3.3 on page 156 in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).

Description of output by this element can be found in Section 206.8.2 in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).

205.3.4.74 Modeling, Finite Element: 20 Node Brick u-p Element

The command is:

```
add element # <.> type 20NodeBrick_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^3*T/M>
    k_y = <L^3*T/M>
    k_z = <L^3*T/M>
    K_s = <F/L^2>
    K_f = <F/L^2>;
```

and/or

```
add element # <.> type 20NodeBrick_up
    with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>,<.>, <.>, <.>, <.>, <.>, ↩
        <.>, <.>, <.>,<.>, <.>, <.>, <.>)
    use material # <.>
    porosity = <.>
    alpha = <.>
    rho_s = <M/L^3>
    rho_f = <M/L^3>
    k_x = <L^3*T/M>
    k_y = <L^3*T/M>
    k_z = <L^3*T/M>
    K_s = <F/L^2>
    K_f = <F/L^2>;
```

where:

- `No (or #)<element_number>` is the unique element integer number that does not have to be sequential, any unique positive integer number can be used.
- `type 8NodeBrick_up` is the element type/name.
- `with nodes (n1, n2, n3, n4, n5, n6, n7, n8, n9, n10, n11, n12, n13, n14, n15, ↩ n16, n17, n18, n19, n20)` are the 20 nodes for this element, is specified order.
- `using <.> Gauss points each direction` is the number of Gauss points to be used in each direction (r1, r2, and r3) for integration of finite element matrices (mass and stiffness). There can be from 1 to 6 Gauss points used (uniformly) in each direction (r1, r2, and r3). Command for the

brick finite element (above) without number of Gauss points control is kept for back compatibility. For 8 node bricks 2 Gauss points are used in each direction ($2 \times 3 \times 3$), while for 20 nodes, 8-20 node and 8-27 node bricks 3 Gauss points are used in each direction ($3 \times 3 \times 3$).

- `use material No (or #)` is the material number which makes up the element (nonlinear elastic or elastic-plastic material properties for each integration (Gauss) point will evolve independently as the element deforms). Use LT version with LT materials.
- `porosity` is the porosity ($n = V_{voids}/V_{total}$) of material in this element.
- `alpha` is the parameter controlling level of effective stress analysis. For soils, usually $\alpha = 1$ is used, while for other materials (saturated concrete, bone material, etc.) lower values are used ($0 \leq \alpha \leq 1$).
- `rho_s` is the density of particles of the solid phase. It is important to note that this is a density of the actual mineral that makes up solid particles!.
- `rho_f` is the density of pore fluid. It is usually density of water, however, for unsaturated and partially saturated materials, this density will be different, as described in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).
- `k_x` is the permeability in the x direction (global x) of the element. It is also important to note about the units used for permeability, as noted below. With isotropic permeability, usually the case, $k_x = k_y = k_z$.
- `k_y` is the permeability in the y direction (global y) of the element. It is also important to note about the units used for permeability, as noted below. With isotropic permeability, usually the case, $k_x = k_y = k_z$.
- `k_z` is the permeability in the z direction (global z) of the element. It is also important to note about the units used for permeability, as noted below. With isotropic permeability, usually the case, $k_x = k_y = k_z$.
- `K_s` is the bulk modulus of the soil phase particles. It is important to note that this is a bulk modulus of the actual mineral that makes up solid particles!
- `K_f` is the bulk modulus of the fluid phase that is found in porous material pores. It is usually bulk modulus of the fluid (physical value of the bulk modulus of fluid, for example water), however, for unsaturated and partially saturated materials, this density is a density of a mixture, as described in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).

Note: the permeability k_x, k_y, k_z is used with dimensions of $[length]^3[time]/[mass]$, which is different from the usual soil mechanics convention, where the permeability has the dimension of velocity, i.e. $[length]/[time]$. Their values are related by $k = K/\rho_f g$, where g is the gravitational acceleration at which the permeability is measured.

More on theory for this finite element can be found in section 102.12.3.3 on page 156 in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).

205.3.4.75 Modeling, Finite Element: 27 Node Brick u-p Element

The command is:

```
add element # <.> type 27NodeBrick_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^3*T/M>
    k_y = <L^3*T/M>
    k_z = <L^3*T/M>
    K_s = <F/L^2>
    K_f = <F/L^2>;
```

and/or

```
add element # <.> type 27NodeBrick_up
    with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>,<.>, <.>, <.>, <.>, <.>, ↩
        <.>, <.>, <.>,<.>, <.>, <.>, <.>, <.>, <.>, <.>,<.>, <.>, <.>, <.>)
    use material # <.>
    porosity = <.>
    alpha = <.>
    rho_s = <M/L^3>
    rho_f = <M/L^3>
    k_x = <L^3*T/M>
    k_y = <L^3*T/M>
    k_z = <L^3*T/M>
    K_s = <F/L^2>
    K_f = <F/L^2>;
```

where:

- `No (or #)<element_number>` is the unique element integer number that does not have to be sequential, any unique positive integer number can be used.
- `type 8NodeBrick_up` is the element type/name.
- `with nodes (n1, n2, n3, n4, n5, n6, n7, n8, n9, n10, n11, n12, n13, n14, n15, ↩ n16, n17, n18, n19, n20, n21, n22, n23, n24, n25, n26, n27)` are the 27 nodes for this element, is specified order.
- `using <.> Gauss points each direction` is the number of Gauss points to be used in each direction (r1, r2, and r3) for integration of finite element matrices (mass and stiffness). There can

be from 1 to 6 Gauss points used (uniformly) in each direction (r1, r2, and r3). Command for the brick finite element (above) without number of Gauss points control is kept for back compatibility. For 8 node bricks 2 Gauss points are used in each direction ($2 \times 3 \times 3$), while for 20 nodes, 8-20 node and 8-27 node bricks 3 Gauss points are used in each direction ($3 \times 3 \times 3$).

- `use material No (or #)` is the material number which makes up the element (nonlinear elastic or elastic-plastic material properties for each integration (Gauss) point will evolve independently as the element deforms). Use LT version with LT materials.
- `porosity` is the porosity ($n = V_{voids}/V_{total}$) of material in this element.
- `alpha` is the parameter controlling level of effective stress analysis. For soils, usually $\alpha = 1$ is used, while for other materials (saturated concrete, bone material, etc.) lower values are used ($0 \leq \alpha \leq 1$).
- `rho_s` is the density of particles of the solid phase. It is important to note that this is a density of the actual mineral that makes up solid particles!.
- `rho_f` is the density of pore fluid. It is usually density of water, however, for unsaturated and partially saturated materials, this density will be different, as described in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).
- `k_x` is the permeability in the x direction (global x) of the element. It is also important to note about the units used for permeability, as noted below. With isotropic permeability, usually the case, $k_x = k_y = k_z$.
- `k_y` is the permeability in the y direction (global y) of the element. It is also important to note about the units used for permeability, as noted below. With isotropic permeability, usually the case, $k_x = k_y = k_z$.
- `k_z` is the permeability in the z direction (global z) of the element. It is also important to note about the units used for permeability, as noted below. With isotropic permeability, usually the case, $k_x = k_y = k_z$.
- `K_s` is the bulk modulus of the soil phase particles. It is important to note that this is a bulk modulus of the actual mineral that makes up solid particles!
- `K_f` is the bulk modulus of the fluid phase that is found in porous material pores. It is usually bulk modulus of the fluid (physical value of the bulk modulus of fluid, for example water), however, for

unsaturated and partially saturated materials, this density is a density of a mixture, as described in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).

Note that, the permeability $\mathbf{k}$ is used with dimensions of $[length]^3[time]/[mass]$, which is different from the usual soil mechanics convention, where the permeability has the dimension of velocity, i.e. $[length]/[time]$. Their values are related by $k = K/\rho_f g$, where g is the gravitational acceleration at which the permeability is measured.

More on theory for this finite element can be found in Section 102.12.3.3 on page 156 of the main document. Description of output by this element can be found in Section 206.8.2.

205.3.4.76 Modeling, Finite Element: 8 Node Brick u-p-U Element

The command is:

```
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^3*T/M>
    k_y = <L^3*T/M>
    k_z = <L^3*T/M>
    K_s = <F/L^2>
    K_f = <F/L^2>;
```

and/or

```
add element # <.> type 8NodeBrick_upU
    with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>)
    use material # <.>
    porosity = <.>
    alpha = <.>
    rho_s = <M/L^3>
    rho_f = <M/L^3>
    k_x = <L^3*T/M>
    k_y = <L^3*T/M>
    k_z = <L^3*T/M>
    K_s = <F/L^2>
    K_f = <F/L^2>;
```

where:

- `No (or #)<element_number>` is the unique element integer number that does not have to be sequential, any unique positive integer number can be used.
- `type 8NodeBrick_up` is the element type/name.
- `with nodes (n1, n2, n3, n4, n5, n6, n7, n8)` are the 8 nodes for this element, is specified order.
- `using <.> Gauss points each direction` is the number of Gauss points to be used in each direction (r1, r2, and r3) for integration of finite element matrices (mass and stiffness). There can be from 1 to 6 Gauss points used (uniformly) in each direction (r1, r2, and r3). Command for the brick finite element (above) without number of Gauss points control is kept for back compatibility.

For 8 node bricks 2 Gauss points are used in each direction ($2 \times 3 \times 3$), while for 20 nodes, 8-20 node and 8-27 node bricks 3 Gauss points are used in each direction ($3 \times 3 \times 3$).

- `use material No (or #)` is the material number which makes up the element (nonlinear elastic or elastic-plastic material properties for each integration (Gauss) point will evolve independently as the element deforms). Use LT version with LT materials.
- `porosity` is the porosity ($n = V_{voids}/V_{total}$) of material in this element.
- `alpha` is the parameter controlling level of effective stress analysis. For soils, usually $\alpha = 1$ is used, while for other materials (saturated concrete, bone material, etc.) lower values are used ($0 \leq \alpha \leq 1$).
- `rho_s` is the density of particles of the solid phase. It is important to note that this is a density of the actual mineral that makes up solid particles!.
- `rho_f` is the density of pore fluid. It is usually density of water, however, for unsaturated and partially saturated materials, this density will be different, as described in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).
- `k_x` is the permeability in the x direction (global x) of the element. It is also important to note about the units used for permeability, as noted below. With isotropic permeability, usually the case, $k_x = k_y = k_z$.
- `k_y` is the permeability in the y direction (global y) of the element. It is also important to note about the units used for permeability, as noted below. With isotropic permeability, usually the case, $k_x = k_y = k_z$.
- `k_z` is the permeability in the z direction (global z) of the element. It is also important to note about the units used for permeability, as noted below. With isotropic permeability, usually the case, $k_x = k_y = k_z$.
- `K_s` is the bulk modulus of the soil phase particles. It is important to note that this is a bulk modulus of the actual mineral that makes up solid particles!
- `K_f` is the bulk modulus of the fluid phase that is found in porous material pores. It is usually bulk modulus of the fluid (physical value of the bulk modulus of fluid, for example water), however, for unsaturated and partially saturated materials, this density is a density of a mixture, as described in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).

Note that, the permeability $\mathbf{k}$ is used with dimensions of $[length]^3[time]/[mass]$, which is different from the usual soil mechanics convention, where the permeability has the dimension of velocity, i.e. $[length]/[time]$. Their values are related by $k = K/\rho_f g$, where g is the gravitational acceleration at which the permeability is measured.

Please note that the $u-p-U$ element, and the $u-p-U$ formulation is a dynamic formulation and is meant to be used with dynamic analysis, and not static analysis, so that all the element matrices, as described in theory section, noted below, are developed and used.

More on theory for this finite element can be found in Section 102.12.1.7 on page 152 in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL). Description of output by this element can be found in Section 206.8.2 in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).

205.3.4.77 Modeling, Finite Element: 20 Node Brick u-p-U Element

The command is:

```
add element # <.> type 20NodeBrick_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^3*T/M>
    k_y = <L^3*T/M>
    k_z = <L^3*T/M>
    K_s = <F/L^2>
    K_f = <F/L^2>;
```

and/or

```
add element # <.> type 20NodeBrick_upU
    with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>,<.>, <.>, <.>, <.>, <.>, ↩
        <.>, <.>, <.>,<.>, <.>, <.>, <.>)
    use material # <.>
    porosity = <.>
    alpha = <.>
    rho_s = <M/L^3>
    rho_f = <M/L^3>
    k_x = <L^3*T/M>
    k_y = <L^3*T/M>
    k_z = <L^3*T/M>
    K_s = <F/L^2>
    K_f = <F/L^2>;
```

where:

- `No (or #)<element_number>` is the unique element integer number that does not have to be sequential, any unique positive integer number can be used.
- `type 8NodeBrick_up` is the element type/name.
- `with nodes (n1, n2, n3, n4, n5, n6, n7, n8, n9, n10, n11, n12, n13, n14, n15, ↩ n16, n17, n18, n19, n20)` are the 20 nodes for this element, is specified order.
- `using <.> Gauss points each direction` is the number of Gauss points to be used in each direction (r1, r2, and r3) for integration of finite element matrices (mass and stiffness). There can be from 1 to 6 Gauss points used (uniformly) in each direction (r1, r2, and r3). Command for the

brick finite element (above) without number of Gauss points control is kept for back compatibility. For 8 node bricks 2 Gauss points are used in each direction ($2 \times 3 \times 3$), while for 20 nodes, 8-20 node and 8-27 node bricks 3 Gauss points are used in each direction ($3 \times 3 \times 3$).

- `use material No (or #)` is the material number which makes up the element (nonlinear elastic or elastic-plastic material properties for each integration (Gauss) point will evolve independently as the element deforms). Use LT version with LT materials.
- `porosity` is the porosity ($n = V_{voids}/V_{total}$) of material in this element.
- `alpha` is the parameter controlling level of effective stress analysis. For soils, usually $\alpha = 1$ is used, while for other materials (saturated concrete, bone material, etc.) lower values are used ($0 \leq \alpha \leq 1$).
- `rho_s` is the density of particles of the solid phase. It is important to note that this is a density of the actual mineral that makes up solid particles!.
- `rho_f` is the density of pore fluid. It is usually density of water, however, for unsaturated and partially saturated materials, this density will be different, as described in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).
- `k_x` is the permeability in the x direction (global x) of the element. It is also important to note about the units used for permeability, as noted below. With isotropic permeability, usually the case, $k_x = k_y = k_z$.
- `k_y` is the permeability in the y direction (global y) of the element. It is also important to note about the units used for permeability, as noted below. With isotropic permeability, usually the case, $k_x = k_y = k_z$.
- `k_z` is the permeability in the z direction (global z) of the element. It is also important to note about the units used for permeability, as noted below. With isotropic permeability, usually the case, $k_x = k_y = k_z$.
- `K_s` is the bulk modulus of the soil phase particles. It is important to note that this is a bulk modulus of the actual mineral that makes up solid particles!
- `K_f` is the bulk modulus of the fluid phase that is found in porous material pores. It is usually bulk modulus of the fluid (physical value of the bulk modulus of fluid, for example water), however, for unsaturated and partially saturated materials, this density is a density of a mixture, as described in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).

Note that, the permeability $\mathbf{k}$ is used with dimensions of $[length]^3[time]/[mass]$, which is different from the usual soil mechanics convention, where the permeability has the dimension of velocity, i.e. $[length]/[time]$. Their values are related by $k = K/\rho_f g$, where g is the gravitational acceleration at which the permeability is measured.

Please note that the $u-p-U$ element, and the $u-p-U$ formulation is a dynamic formulation and is meant to be used with dynamic analysis, and not static analysis, so that all the element matrices, as described in theory section, noted below, are developed and used.

More on theory for this finite element can be found in section 102.12.1.8 on page 152 in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL). Description of output by this element can be found in Section 206.8.2.

205.3.4.78 Modeling, Finite Element: 27 Node Brick u-p-U Element

The command is:

```
add element # <.> type 27NodeBrick_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^3*T/M>
    k_y = <L^3*T/M>
    k_z = <L^3*T/M>
    K_s = <F/L^2>
    K_f = <F/L^2>;
```

and/or

```
add element # <.> type 27NodeBrick_upU
    with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>,<.>, <.>, <.>, <.>, <.>, ↩
        <.>, <.>, <.>,<.>, <.>, <.>, <.>, <.>, <.>, <.>,<.>, <.>, <.>, <.>)
    use material # <.>
    porosity = <.>
    alpha = <.>
    rho_s = <M/L^3>
    rho_f = <M/L^3>
    k_x = <L^3*T/M>
    k_y = <L^3*T/M>
    k_z = <L^3*T/M>
    K_s = <F/L^2>
    K_f = <F/L^2>;
```

where:

- `No (or #)<element_number>` is the unique element integer number that does not have to be sequential, any unique positive integer number can be used.
- `type 8NodeBrick_up` is the element type/name.
- `with nodes (n1, n2, n3, n4, n5, n6, n7, n8, n9, n10, n11, n12, n13, n14, n15, ↩ n16, n17, n18, n19, n20, n21, n22, n23, n24, n25, n26, n27)` are the 27 nodes for this element, is specified order.
- `using <.> Gauss points each direction` is the number of Gauss points to be used in each direction (r1, r2, and r3) for integration of finite element matrices (mass and stiffness). There can

be from 1 to 6 Gauss points used (uniformly) in each direction (r1, r2, and r3). Command for the brick finite element (above) without number of Gauss points control is kept for back compatibility. For 8 node bricks 2 Gauss points are used in each direction ($2 \times 3 \times 3$), while for 20 nodes, 8-20 node and 8-27 node bricks 3 Gauss points are used in each direction ($3 \times 3 \times 3$).

- `use material No (or #)` is the material number which makes up the element (nonlinear elastic or elastic-plastic material properties for each integration (Gauss) point will evolve independently as the element deforms). Use LT version with LT materials.
- `porosity` is the porosity ($n = V_{voids}/V_{total}$) of material in this element.
- `alpha` is the parameter controlling level of effective stress analysis. For soils, usually $\alpha = 1$ is used, while for other materials (saturated concrete, bone material, etc.) lower values are used ($0 \leq \alpha \leq 1$).
- `rho_s` is the density of particles of the solid phase. It is important to note that this is a density of the actual mineral that makes up solid particles!.
- `rho_f` is the density of pore fluid. It is usually density of water, however, for unsaturated and partially saturated materials, this density will be different, as described in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).
- `k_x` is the permeability in the x direction (global x) of the element. It is also important to note about the units used for permeability, as noted below. With isotropic permeability, usually the case, $k_x = k_y = k_z$.
- `k_y` is the permeability in the y direction (global y) of the element. It is also important to note about the units used for permeability, as noted below. With isotropic permeability, usually the case, $k_x = k_y = k_z$.
- `k_z` is the permeability in the z direction (global z) of the element. It is also important to note about the units used for permeability, as noted below. With isotropic permeability, usually the case, $k_x = k_y = k_z$.
- `K_s` is the bulk modulus of the soil phase particles. It is important to note that this is a bulk modulus of the actual mineral that makes up solid particles!
- `K_f` is the bulk modulus of the fluid phase that is found in porous material pores. It is usually bulk modulus of the fluid (physical value of the bulk modulus of fluid, for example water), however, for

unsaturated and partially saturated materials, this density is a density of a mixture, as described in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).

Note that, the permeability $\mathbf{k}$ is used with dimensions of $[length]^3[time]/[mass]$, which is different from the usual soil mechanics convention, where the permeability has the dimension of velocity, i.e. $[length]/[time]$. Their values are related by $k = K/\rho_f g$, where g is the gravitational acceleration at which the permeability is measured.

Please note that the $u-p-U$ element, and the $u-p-U$ formulation is a dynamic formulation and is meant to be used with dynamic analysis, and not static analysis, so that all the element matrices, as described in theory section, noted below, are developed and used.

More on theory for this finite element can be found in Section 102.12.1.9 on page 152 of the main document. Description of output by this element can be found in Section 206.8.2.

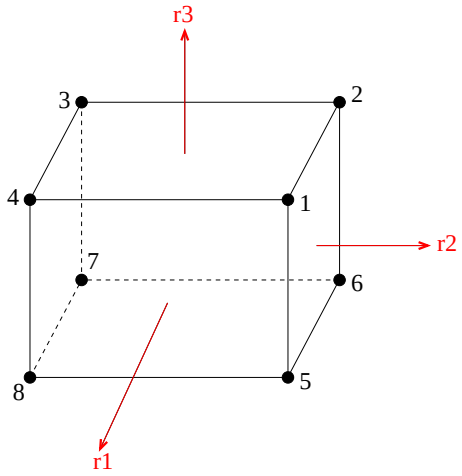
205.3.4.79 Modeling, Finite Element: 8 Node Cosserat Brick Element

The command is:

```
add element # <element_number> type Cosserat8NodeBrick
  with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>)
  use material # <.>;
```

where:

- `<element_number>` is the unique element integer number (does not have to be sequential, any unique positive integer number can be used)
- `type Cosserat8NodeBrick` is the element type.
- `with nodes (n1, n2, n3, n4, n5, n6, n7, n8)` are the 8 nodes for this element. Each node should have 6 DOFs for this element. The element should be in the order as per figure below



- `use material No (or #)` is the material number which makes up the element. The element can use materials `Cosserat_linear_elastic_isotropic_3d` and `Cosserat_von_Mises`.

205.3.4.80 Modeling, Finite Element: Bonded Contact/Interface/Joint Element

The command is:

```
add element # <.> type BondedContact
    with nodes (<.>, <.>)
    penalty_stiffness = <F/L>
```

where

- `penalty_stiffness` represents the penalty stiffness in the three orthogonal x, y and z directions, that connects two nodes of this element.

205.3.4.81 Modeling, Finite Element: Coupled Bonded Contact/Interface/Joint Element

The command is:

```
add element # <.> type CoupledBondedContact
    with nodes (<.>, <.>)
    penalty_stiffness = <F/L>
    axial_viscous_damping = <F/L>
    shear_viscous_damping = <F/L>
    contact_plane_vector = (<.>, <.>, <.> );
```

where

- `penalty_stiffness` represents the penalty stiffness in the three orthogonal x, y and z directions, that connects two nodes of this element. The penalty stiffness is used for both solid and fluid DOFs.
- `axial_viscous_damping` is the viscous damping in axial.
- `shear_viscous_damping` is the viscous damping in shear.
- `contact_plane_vector` defines the normal to the contact/interface/joint plane.

IMPORTANT NOTE No. 1: `contact_plane_vector` defines a direction from Node I to Node J, that is, from the first to the second node. If this normal vector is reversed, the contact/interface/joint element behaves as a hook and is likely to create convergence issues.

IMPORTANT NOTE No. 2: Two nodes that form the Contact/Interface/Joint Element, need to be placed at the same physical location, coordinates in order to prevent convergence issues when nodes are separated and element tries to close the gap in the very first step.

205.3.4.82 Modeling, Finite Element: Force Based Dry Hard Contact/Interface/Joint Element

The command is:

```
add element # <.> type ForceBasedHardContact
    with nodes (<.>, <.>)
    axial_stiffness = <F/L>
    shear_stiffness = <F/L>
    axial_viscous_damping = <F/L>
    shear_viscous_damping = <F/L>
    friction_ratio = <.>
    contact_plane_vector = (<.>, <.>, <.> );
```

The axial force F_a and axial stiffness E_a in defined as

$$F_a = E_a * \delta_a \tag{205.2}$$

where

δ_a refers to the axial relative displacement in axial contact/interface/joint direction,

E_a refers to the axial stiffness in axial contact direction, and

- `axial_stiffness (b)` represents the stiffness in the axial/axial direction (local x axis).
- `shear_stiffness` Is the stiffness in the tangential (shear, local y or z axis) directions.
- `axial_viscous_damping` Is the viscous damping in axial/axial.
- `shear_viscous_damping` Is the viscous damping in shear.
- `friction_ratio` Coulomb friction ratio.
- `contact_plane_vector` Vector defining the normal to the contact/interface/joint plane.

IMPORTANT NOTE No. 1: `contact_plane_vector` defines a direction from Node I to Node J, that is, from the first to the second node. If this normal vector is reversed, the contact/interface/joint element behaves as a hook and is likely to create convergence issues.

IMPORTANT NOTE No. 2: Two nodes that form the Contact/Interface/Joint Element, need to be placed at the same physical location, coordinates in order to prevent convergence issues when nodes are separated and element tries to close the gap in the very first step.

Description of output by this element can be found in Section 206.8.8.

205.3.4.83 Modeling, Finite Element: Force Based Dry Soft Contact/Interface/Joint Element

The command is:

```
add element # <.> type ForceBasedSoftContact
    with nodes (<.>, <.>)
    initial_axial_stiffness = <F/L>
    stiffening_rate = <1/m>
    max_axial_stiffness = <F/L>
    shear_stiffness = <F/L>
    axial_viscous_damping = <F/L>
    shear_viscous_damping = <F/L>
    friction_ratio = <.>
    contact_plane_vector = (<.>, <.>, <.> );
```

The axial force F_a and axial stiffness E_a in defined as

$$F_a = b * exp(a * \delta_a) * \delta_a$$
$$E_a = max(b * exp(a * \delta_a) * (1 + a * \delta_a), E_{max}) \tag{205.3}$$

where

δ_a refers to the axial relative displacement in axial contact/interface/joint direction,

b refers to the axial stiffness in axial contact/interface/joint direction,

a refers to the stiffening rate in axial contact/interface/joint direction,

E_{max} refers to the maximum axial stiffness, and

- `initial_axial_stiffness (b)` represents the stiffness in the axial direction (local x axis).
- `stiffening_rate (a)` Represents exponential stiffening rate $exp(a * \delta_a)$ in axial direction.
- `max_axial_stiffness` (E_{max}) Defines the maximum stiffness in the axial direction (local x axis).
- `shear_stiffness` Is the stiffness in the tangential (shear, local y or z axis) directions.
- `axial_viscous_damping` Is the viscous damping in axial.
- `shear_viscous_damping` Is the viscous damping in shear.
- `friction_ratio` Coulomb friction ratio.
- `contact_plane_vector` Vector defining the normal to the contact/interface/joint plane.

IMPORTANT NOTE No. 1: `contact_plane_vector` defines a direction from Node I to Node J, that is, from the first to the second node. If this normal vector is reversed, the contact/interface/joint element behaves as a hook and is likely to create convergence issues.

IMPORTANT NOTE No. 2: Two nodes that form the Contact/Interface/Joint Element, need to be placed at the same physical location, coordinates in order to prevent convergence issues when nodes are separated and element tries to close the gap in the very first step.

Description of output by this element can be found in Section 206.8.8.

205.3.4.84 Modeling, Finite Element: Force Based Coupled Hard Contact/Interface/Joint Element

The command is:

```
add element # <.> type ForceBasedCoupledHardContact
    with nodes (<.>, <.>)
    axial_stiffness = <F/L>
    axial_penalty_stiffness = <F/L>
    shear_stiffness = <F/L>
    axial_viscous_damping = <F/L>
    shear_viscous_damping = <F/L>
    friction_ratio = <.>
    contact_plane_vector = (<.>, <.>, <.> );
```

The axial force F_a and axial stiffness E_a in defined as

$$\begin{aligned} F_a &= b * \delta_a \\ E_a &= b \end{aligned} \tag{205.4}$$

where

δ_a refers to the axial relative displacement in axial contact/interface/joint direction,

b refers to the axial stiffness in axial contact/interface/joint direction, and

- `axial_stiffness (b)` represents the axial stiffness in the axial direction (local x axis).
- `axial_penalty_stiffness` (E_p) defines the penalty stiffness between U_i degree of freedom (DoF) (saturated, coupled u-p-U element) and u_i DoF (dry u element) to enforce movement of fluid in u-p-U element with solid in u element in contact/interface/joint axial direction. This is useful for pumping action, gap opens and draws the fluid from u-p-U element and then gap closes and pumps, pushes fluid into u-p-U element.
- `shear_stiffness` Is the stiffness in the tangential (shear, local y or z axis) directions.
- `axial_viscous_damping` Is the viscous damping in axial.
- `shear_viscous_damping` Is the viscous damping in shear.
- `friction_ratio` Coulomb friction ratio.
- `contact_plane_vector` Vector defining the normal to the contact/interface/joint plane.

IMPORTANT NOTE No. 1: `contact_plane_vector` defines a direction from Node I to Node J, that is, from the first to the second node. If this normal vector is reversed, the contact/interface/joint element behaves as a hook and is likely to create convergence issues.

IMPORTANT NOTE No. 2: Two nodes that form the Contact/Interface/Joint Element, need to be placed at the same physical location, coordinates in order to prevent convergence issues when nodes are separated and element tries to close the gap in the very first step.

Description of output by this element can be found in Section 206.8.8.

205.3.4.85 Modeling, Finite Element: Force Based Coupled Soft Contact/Interface/Joint Element

The command is:

```
add element # <.> type ForceBasedCoupledSoftContact
    with nodes (<.>, <.>)
    initial_axial_stiffness = <F/L>
    stiffening_rate = <1/m>
    max_axial_stiffness = <F/L>
    axial_penalty_stiffness = <F/L>
    shear_stiffness = <F/L>
    axial_viscous_damping = <F/L>
    shear_viscous_damping = <F/L>
    friction_ratio = <.>
    contact_plane_vector = (<.>, <.>, <.> );
```

The axial force F_a and axial stiffness E_a in defined as

$$\begin{aligned} F_a &= b * exp(a * \delta_a) * \delta_a \\ E_a &= max(b * exp(a * \delta_a) * (1 + a * \delta_a), E_{max}) \end{aligned} \tag{205.5}$$

where

δ_a refers to the axial relative displacement in axial contact/interface/joint direction,

b refers to the axial stiffness in axial contact/interface/joint direction,

a refers to the stiffening rate in axial contact/interface/joint direction,

E_{max} refers to the maximum axial stiffness, and

- `initial_axial_stiffness (b)` represents the stiffness in the axial direction (local x axis).
- `stiffening_rate (a)` Represents exponential stiffening rate $exp(a * \delta_a)$ in axial direction.
- `max_axial_stiffness` (E_{max}) Defines the maximum stiffness in the axial direction (local x axis).
- `axial_penalty_stiffness` (E_p) defines the penalty stiffness between U_i degree of freedom (DoF) (saturated, coupled u-p-U element) and u_i DoF (dry u element) to enforce movement of fluid in u-p-U element with solid in u element in contact/interface/joint axial direction. This is useful for pumping action, gap opens and draws the fluid from u-p-U element and then gap closes and pumps, pushes fluid into u-p-U element.
- `shear_stiffness` Is the stiffness in the tangential (shear, local y or z axis) directions.
- `axial_viscous_damping` Is the viscous damping in axial.
- `shear_viscous_damping` Is the viscous damping in shear.

- `friction_ratio` Coulomb friction ratio.
- `contact_plane_vector` Vector defining the normal to the contact/interface/joint plane.

IMPORTANT NOTE No. 1: `contact_plane_vector` defines a direction from Node I to Node J, that is, from the first to the second node. If this normal vector is reversed, the contact/interface/joint element behaves as a hook and is likely to create convergence issues.

IMPORTANT NOTE No. 2: Two nodes that form the Contact/Interface/Joint Element, need to be placed at the same physical location, coordinates in order to prevent convergence issues when nodes are separated and element tries to close the gap in the very first step.

Description of output by this element can be found in Section 206.8.8.

205.3.4.86 Modeling, Finite Element: Stress Based Dry Hard Contact/Interface/Joint Element with Elastic Perfectly Plastic Shear Behavior

The command is:

```
add element # <.> type StressBasedHardContact_ElPPlShear
    with nodes (<.>, <.>)
    axial_stiffness = <Pa>
    initial_shear_stiffness = <Pa>
    axial_viscous_damping = <Pa*s>
    shear_viscous_damping = <Pa*s>
    residual_friction_coefficient = <.>
    shear_zone_thickness = <m>
    contact_plane_vector = (<.>, <.>, <.> );
```

and/or;

```
add element # <.> type StressBasedHardContact_ElPPlShear
    with nodes (<.>, <.>)
    axial_stiffness = <Pa>
    initial_shear_stiffness = <Pa>
    axial_viscous_damping = <Pa*s>
    shear_viscous_damping = <Pa*s>
    residual_friction_coefficient = <.>
    shear_zone_thickness = <m>
    surface_vector_relative_tolerance = <.>;
```

The axial stress σ_a and axial stiffness E_a in defined as

$$\begin{aligned} \sigma_a &= b * \epsilon_a \\ E_a &= b \end{aligned} \tag{205.6}$$

where

$E_a = b$ refers to the axial stiffness in axial contact/interface/joint direction,

ϵ_a refers to the axial strain in axial contact/interface/joint direction $\epsilon_a = \delta_a/h$,

δ_a is the relative axial penetration in contact axial direction,

h is the shear zone thickness, and

- `axial_stiffness` (b) represents the stiffness in the axial direction (local x axis).
- `initial_shear_stiffness` (E_s) Is the stiffness in the tangential (shear, local y or z axis) directions at $101kPa$ axial stress described in Section 104.7.3.2
- `axial_viscous_damping` Is the viscous damping in axial.

- `shear_viscous_damping` Is the viscous damping in shear.
- `residual_friction_coefficient` (μ_r) Is the residual friction coefficient described in Section 104.7.3.2.
- `shear_zone_thickness` h Is the shear zone thickness.
- `contact_plane_vector` Vector defining the normal to the contact/interface/joint plane.
- `surface_vector_relative_tolerance` defines the relative tolerance to find all the contact/interface/joint normals and create multiple contact elements for a given contact node pairs for a conforming surface-to-surface mesh.

IMPORTANT NOTE No. 1: `contact_plane_vector` defines a direction from Node I to Node J, that is, from the first to the second node. If this normal vector is reversed, the contact/interface/joint element behaves as a hook and is likely to create convergence issues.

IMPORTANT NOTE No. 2: Two nodes that form the Contact/Interface/Joint Element, need to be placed at the same physical location, coordinates in order to prevent convergence issues when nodes are separated and element tries to close the gap in the very first step.

Description of output by this element can be found in Section 206.8.8.

205.3.4.87 Modeling, Finite Element: Stress Based Dry Hard Contact/Interface/Joint Element with Nonlinear Hardening Shear Behavior

The command is:

```
add element # <.> type StressBasedHardContact_NonLinHardShear
    with nodes (<.>, <.>)
    axial_stiffness = <Pa>
    initial_shear_stiffness = <Pa>
    axial_viscous_damping = <Pa*s>
    shear_viscous_damping = <Pa*s>
    residual_friction_coefficient = <.>
    shear_zone_thickness = <m>
    contact_plane_vector = (<.>, <.>, <.> );
```

and/or;

```
add element # <.> type StressBasedHardContact_NonLinHardShear
    with nodes (<.>, <.>)
    axial_stiffness = <Pa>
    initial_shear_stiffness = <Pa>
    axial_viscous_damping = <Pa*s>
    shear_viscous_damping = <Pa*s>
    residual_friction_coefficient = <.>
    shear_zone_thickness = <m>
    surface_vector_relative_tolerance = <.>;
```

The axial stress σ_a and axial stiffness E_a in defined as

$$\begin{aligned} \sigma_a &= b * \epsilon_a \\ E_a &= b \end{aligned} \tag{205.7}$$

where

$E_a = b$ refers to the axial stiffness in axial contact/interface/joint direction,

ϵ_a refers to the axial strain in axial contact direction $\epsilon_a = \delta_a / h$,

δ_a is the relative axial penetration in contact/interface/joint axial direction,

h is the shear zone thickness, and

- `axial_stiffness (b)` represents the stiffness in the axial direction (local x axis).
- `initial_shear_stiffness` (E_s) Is the stiffness in the tangential (shear, local y or z axis) directions at $101kPa$ axial stress, described in Section 104.7.3.4
- `axial_viscous_damping` Is the viscous damping in axial.

- `shear_viscous_damping` Is the viscous damping in shear.
- `residual_friction_coefficient` (μ_r) Is the residual frictional parameter as described in Section 104.7.3.3
- `shear_zone_thickness` h Is the shear zone thickness.
- `contact_plane_vector` Vector defining the normal to the contact/interface/joint plane.
- `surface_vector_relative_tolerance` defines the relative tolerance to find all the contact/interface/joint normals and create multiple contact elements for a given contact node pairs for a conforming surface-to-surface mesh.

IMPORTANT NOTE No. 1: `contact_plane_vector` defines a direction from Node I to Node J, that is, from the first to the second node. If this normal vector is reversed, the contact/interface/joint element behaves as a hook and is likely to create convergence issues.

IMPORTANT NOTE No. 2: Two nodes that form the Contact/Interface/Joint Element, need to be placed at the same physical location, coordinates in order to prevent convergence issues when nodes are separated and element tries to close the gap in the very first step.

Description of output by this element can be found in Section 206.8.8.

205.3.4.88 Modeling, Finite Element: Stress Based Dry Hard Contact/Interface/Joint Element with Nonlinear Hardening and Softening Shear Behavior

The command is:

```
add element # <.> type StressBasedHardContact_NonLinHardSoftShear
    with nodes (<.>, <.>)
    axial_stiffness = <Pa>
    initial_shear_stiffness = <Pa>
    rate_of_softening = <>
    size_of_peak_plateau = <>
    axial_viscous_damping = <Pa*s>
    shear_viscous_damping = <Pa*s>
    peak_friction_coefficient_limit = <>
    peak_friction_coefficient_rate_of_decrease = <.>
    residual_friction_coefficient = <.>
    shear_zone_thickness = <m>
    contact_plane_vector = (<.>, <.>, <.> );
```

and/or;

```
add element # <.> type StressBasedHardContact_NonLinHardSoftShear
    with nodes (<.>, <.>)
    axial_stiffness = <Pa>
    initial_shear_stiffness = <Pa>
    rate_of_softening = <>
    size_of_peak_plateau = <>
    axial_viscous_damping = <Pa*s>
    shear_viscous_damping = <Pa*s>
    peak_friction_coefficient_limit = <>
    peak_friction_coefficient_rate_of_decrease = <.>
    residual_friction_coefficient = <.>
    shear_zone_thickness = <m>
    surface_vector_relative_tolerance = <.>;
```

The axial stress σ_a and axial stiffness E_a in defined as

$$\begin{aligned} \sigma_a &= b * \epsilon_a \\ E_a &= b \end{aligned} \tag{205.8}$$

where

$E_a = b$ refers to the axial stiffness in axial contact/interface/joint direction,

ϵ_a refers to the axial strain in axial contact direction $\epsilon_a = \delta_a/h$,

δ_a is the relative axial penetration in contact/interface/joint axial direction,

h is the shear zone thickness, and

- `axial_stiffness (b)` represents the stiffness in the axial direction (local x axis).

- `initial_shear_stiffness` (E_s) Is the stiffness in the tangential (shear, local y or z axis) directions at $101kPa$ axial stress, described in Section 104.7.3.4

- `rate_of_softening` (R_s) Is the parameter to control the rate of frictional softening described in Section 104.7.3.4. The frictional softening function is an inverse tangent function raised to power n with incremental form as

$$\Delta\mu = -\frac{n * R_s(\mu_p - \mu_r)}{(\pi/2)^n \theta^{1/n-1}} * cos^2\theta\Delta\gamma^p \tag{205.9}$$

$$\theta = \frac{\mu_p - \mu}{\mu_p - \mu_r}(\pi/2)^n \tag{205.10}$$

  where, R_s is the frictional softening rate parameter, $\Delta\gamma^p$ is the plastic shear strain and n represents the size of the peak plateau.

$$\Delta\gamma^p = \sqrt{\Delta\epsilon^p_{ij}\Delta\epsilon^p_{ij}} \tag{205.11}$$

- `size_of_peak_plateau` (n) Is the frictional softening parameter to control the size of plateau as described in Section 104.7.3.4. The frictional softening function is an inverse tangent function raised to power n with incremental form as shown in Equation 205.17.

- `axial_viscous_damping` Is the viscous damping in axial.

- `shear_viscous_damping` Is the viscous damping in shear.

- `peak_friction_coefficient_limit` (μ_{p0}) Is the limit to the peak frictional hardening parameter μ_p.

- `peak_friction_coefficient_rate_of_decrease` (k) Is the rate of decrease of peak frictional hardening parameter μ_p with axial stress, described in Section 104.7.3.4

$$\mu_p = max(\mu_{p0}, \mu_{p0} - k * log(\sigma_a/P_0)) \tag{205.12}$$

  where μ_{p0} is the peak frictional hardening limit, k is the peak frictional parameter rate of decrease and P_0 is the reference stress of $P_0 = 101kPa$.

- `residual_friction_coefficient` (μ_r) Is the residual frictional parameter as described in Section 104.7.3.4

- `shear_zone_thickness` h Is the shear zone thickness.

- `contact_plane_vector` Vector defining the normal to the contact/interface/joint plane.

- `surface_vector_relative_tolerance` defines the relative tolerance to find all the contact/interface/joint normals and create multiple contact elements for a given contact node pairs for a conforming surface-to-surface mesh.

IMPORTANT NOTE No. 1: `contact_plane_vector` defines a direction from Node I to Node J, that is, from the first to the second node. If this normal vector is reversed, the contact/interface/joint element behaves as a hook and is likely to create convergence issues.

IMPORTANT NOTE No. 2: Two nodes that form the Contact/Interface/Joint Element, need to be placed at the same physical location, coordinates in order to prevent convergence issues when nodes are separated and element tries to close the gap in the very first step.

Description of output by this element can be found in Section 206.8.8.

205.3.4.89 Modeling, Finite Element: Stress Based Dry Soft Contact/Interface/Joint Element with Elastic Perfectly Plastic Shear Behavior

The command is:

```
add element # <.> type StressBasedSoftContact_ElPPlShear
    with nodes (<.>, <.>)
    initial_axial_stiffness = <Pa>
    stiffening_rate = <>
    max_axial_stiffness = <Pa>
    initial_shear_stiffness = <Pa>
    axial_viscous_damping = <Pa*s>
    shear_viscous_damping = <Pa*s>
    residual_friction_coefficient = <.>
    shear_zone_thickness = <m>
    contact_plane_vector = (<.>, <.>, <.> );
```

and/or;

```
add element # <.> type StressBasedSoftContact_ElPPlShear
    with nodes (<.>, <.>)
    initial_axial_stiffness = <Pa>
    stiffening_rate = <>
    max_axial_stiffness = <Pa>
    initial_shear_stiffness = <Pa>
    axial_viscous_damping = <Pa*s>
    shear_viscous_damping = <Pa*s>
    residual_friction_coefficient = <.>
    shear_zone_thickness = <m>
    surface_vector_relative_tolerance = <.>;
```

The axial stress σ_a and axial stiffness E_a in defined as

$$\begin{aligned} \sigma_a &= b * exp(a * \epsilon_a) * \epsilon_a \\ E_a &= max(b * exp(a * \epsilon_a) * (1 + a * \epsilon_a), E_{max}) \end{aligned} \tag{205.13}$$

where

b refers to the initial axial stiffness in axial contact/interface/joint direction,

a refers to the stiffening rate in axial contact direction,

E_{max} refers to the maximum axial stiffness,

E_a refers to the axial stiffness,

ϵ_a refers to the axial strain in axial contact/interface/joint direction $\epsilon_a = \delta_a/h$,

δ_a is the relative axial penetration in contact axial direction,

h is the shear zone thickness, and

- `initial_axial_stiffness (b)` represents the stiffness in the axial direction (local x axis).
- `stiffening_rate (a)` Represents exponential stiffening rate $exp(a * \epsilon_a)$ in axial direction.
- `max_axial_stiffness`(E_{max}) Defines the maximum stiffness in the axial direction (local x axis) for the contact/interface/joint element.
- `initial_shear_stiffness` (E_s) Is the stiffness in the tangential (shear, local y or z axis) directions at $101kPa$ axial stress described in Section 104.7.3.2
- `axial_viscous_damping` Is the viscous damping in axial.
- `shear_viscous_damping` Is the viscous damping in shear.
- `residual_friction_coefficient` (μ_r) Is the residual friction coefficient described in Section 104.7.3.2
- `shear_zone_thickness` h Is the shear zone thickness
- `contact_plane_vector` Vector defining the normal to the contact/interface/joint plane.
- `surface_vector_relative_tolerance` defines the relative tolerance to find all the contact/interface/joint normals and create multiple contact elements for a given contact node pairs for a conforming surface-to-surface mesh.

IMPORTANT NOTE No. 1: `contact_plane_vector` defines a direction from Node I to Node J, that is, from the first to the second node. If this normal vector is reversed, the contact/interface/joint element behaves as a hook and is likely to create convergence issues.

IMPORTANT NOTE No. 2: Two nodes that form the Contact/Interface/Joint Element, need to be placed at the same physical location, coordinates in order to prevent convergence issues when nodes are separated and element tries to close the gap in the very first step.

Description of output by this element can be found in Section 206.8.8.

205.3.4.90 Modeling, Finite Element: Stress Based Dry Soft Contact/Interface/Joint Element with Nonlinear Hardening Shear Behavior

The command is:

```
add element # <.> type StressBasedSoftContact_NonLinHardShear
    with nodes (<.>, <.>)
    initial_axial_stiffness = <Pa>
    stiffening_rate = <>
    max_axial_stiffness = <Pa>
    initial_shear_stiffness = <Pa>
    axial_viscous_damping = <Pa*s>
    shear_viscous_damping = <Pa*s>
    residual_friction_coefficient = <.>
    shear_zone_thickness = <m>
    contact_plane_vector = (<.>, <.>, <.> );
```

and/or;

```
add element # <.> type StressBasedSoftContact_NonLinHardShear
    with nodes (<.>, <.>)
    initial_axial_stiffness = <Pa>
    stiffening_rate = <>
    max_axial_stiffness = <Pa>
    initial_shear_stiffness = <Pa>
    axial_viscous_damping = <Pa*s>
    shear_viscous_damping = <Pa*s>
    residual_friction_coefficient = <.>
    shear_zone_thickness = <m>
    surface_vector_relative_tolerance = <.>;
```

The axial stress σ_a and axial stiffness E_a in defined as

$$\begin{aligned} \sigma_a &= b * exp(a * \epsilon_a) * \epsilon_a \\ E_a &= max(b * exp(a * \epsilon_a) * (1 + a * \epsilon_a), E_{max}) \end{aligned} \tag{205.14}$$

where

b refers to the initial axial stiffness in axial contact/interface/joint direction,

a refers to the stiffening rate in axial contact direction,

E_{max} refers to the maximum axial stiffness,

E_a refers to the axial stiffness,

ϵ_a refers to the axial strain in axial contact/interface/joint direction $\epsilon_a = \delta_a/h$,

δ_a is the relative axial penetration in contact axial direction,

h is the shear zone thickness, and

- `initial_axial_stiffness` (b) represents the stiffness in the axial direction (local x axis) for 1m penetration.
- `stiffening_rate` (a) Represents exponential stiffening rate $exp(sr * \epsilon_a)$ in axial direction.
- `max_axial_stiffness`(E_{max}) Defines the maximum stiffness in the axial direction (local x axis) for the contact/interface/joint element.
- `initial_shear_stiffness` (E_s) Is the stiffness in the tangential (shear, local y or z axis) directions at $101kPa$ axial stress described in Section 104.7.3.3
- `axial_viscous_damping` Is the viscous damping in axial.
- `shear_viscous_damping` Is the viscous damping in shear.
- `residual_friction_coefficient` (μ_r) Is the residual frictional parameter as described in Section 104.7.3.3
- `shear_zone_thickness` h Is the shear zone thickness
- `contact_plane_vector` Vector defining the normal to the contact/interface/joint plane.
- `surface_vector_relative_tolerance` defines the relative tolerance to find all the contact/interface/joint normals and create multiple contact elements for a given contact node pairs for a conforming surface-to-surface mesh.

IMPORTANT NOTE No. 1: `contact_plane_vector` defines a direction from Node I to Node J, that is, from the first to the second node. If this normal vector is reversed, the contact/interface/joint element behaves as a hook and is likely to create convergence issues.

IMPORTANT NOTE No. 2: Two nodes that form the Contact/Interface/Joint Element, need to be placed at the same physical location, coordinates in order to prevent convergence issues when nodes are separated and element tries to close the gap in the very first step.

Description of output by this element can be found in Section 206.8.8.

205.3.4.91 Modeling, Finite Element: Stress Based Dry Soft Contact/Interface/Joint Element with Nonlinear Hardening and Softening Shear Behavior

The command is:

```
add element # <.> type StressBasedSoftContact_NonLinHardSoftShear
    with nodes (<.>, <.>)
    initial_axial_stiffness = <Pa>
    stiffening_rate = <>
    max_axial_stiffness = <Pa>
    initial_shear_stiffness = <Pa>
    rate_of_softening = <>
    size_of_peak_plateau = <>
    axial_viscous_damping = <Pa*s>
    shear_viscous_damping = <Pa*s>
    peak_friction_coefficient_limit = <>
    peak_friction_coefficient_rate_of_decrease = <.>
    residual_friction_coefficient = <.>
    shear_zone_thickness = <m>
    contact_plane_vector = (<.>, <.>, <.> );
```

and/or;

```
add element # <.> type StressBasedSoftContact_NonLinHardSoftShear
    with nodes (<.>, <.>)
    initial_axial_stiffness = <Pa>
    stiffening_rate = <>
    max_axial_stiffness = <Pa>
    initial_shear_stiffness = <Pa>
    rate_of_softening = <>
    size_of_peak_plateau = <>
    axial_viscous_damping = <Pa*s>
    shear_viscous_damping = <Pa*s>
    peak_friction_coefficient_limit = <>
    peak_friction_coefficient_rate_of_decrease = <.>
    residual_friction_coefficient = <.>
    shear_zone_thickness = <m>
    surface_vector_relative_tolerance = <.>;
```

The axial stress σ_a and axial stiffness E_a in defined as

$$\begin{aligned} \sigma_a &= b * exp(a * \epsilon_a) * \epsilon_a \\ E_a &= max(b * exp(a * \epsilon_a) * (1 + a * \epsilon_a), E_{max}) \end{aligned} \tag{205.15}$$

where

b refers to the initial axial stiffness in axial contact/interface/joint direction,

a refers to the stiffening rate in axial contact direction,

E_{max} refers to the maximum axial stiffness,

E_a refers to the axial stiffness,
ϵ_a refers to the axial strain in axial contact/interface/joint direction $\epsilon_a = \delta_a/h$,
δ_a is the relative axial penetration in contact axial direction,
h is the shear zone thickness, and

- `initial_axial_stiffness` (b) represents the stiffness in the axial direction (local x axis) for 1m penetration.
- `stiffening_rate` (a) Represents exponential stiffening rate $exp(sr * \epsilon_n)$ in axial direction.
- `max_axial_stiffness`(E_{max}) Defines the maximum stiffness in the axial direction (local x axis) for the contact/interface/joint element.
- `initial_shear_stiffness` (E_s) Is the stiffness in the tangential (shear, local y or z axis) directions at $101kPa$ axial stress, described in Section 104.7.3.4
- `rate_of_softening` (R_s) Is the parameter to control the rate of frictional softening described in Section 104.7.3.4. The frictional softening function is an inverse tangent function raised to power n with incremental form as

$$\Delta\mu = -\frac{n * R_s(\mu_p - \mu_r)}{(\pi/2)^n \theta^{1/n-1}} * cos^2\theta\Delta\gamma^p \tag{205.16}$$

$$\theta = \frac{\mu_p - \mu}{\mu_p - \mu_r}(\pi/2)^n \tag{205.17}$$

  where, R_s is the frictional softening rate parameter, $\Delta\gamma^p$ is the plastic shear strain and n represents the size of the peak plateau.

$$\Delta\gamma^p = \sqrt{\Delta\epsilon^p_{ij}\Delta\epsilon^p_{ij}} \tag{205.18}$$

- `size_of_peak_plateau` (n) Is the frictional softening parameter to control the size of plateau as described in Section 104.7.3.4. The frictional softening function is an inverse tangent function raised to power n with incremental form as shown in Equation 205.17.
- `axial_viscous_damping` Is the viscous damping in axial.
- `shear_viscous_damping` Is the viscous damping in shear.
- `peak_friction_coefficient_limit` (μ_{p0}) Is the limit to the peak frictional hardening parameter μ_p.

- `peak_friction_coefficient_rate_of_decrease` (k) Is the rate of decrease of peak frictional hardening parameter μ_p with axial stress, described in Section 104.7.3.4

$$\mu_p = max(\mu_{p0}, \mu_{p0} - k * log(\sigma_a / P_0)) \tag{205.19}$$

  where μ_{p0} is the peak frictional hardening limit, k is the peak frictional parameter rate of decrease and P_0 is the reference stress of $P_0 = 101kPa$.

- `residual_friction_coefficient` (μ_r) Is the residual frictional parameter as described in Section 104.7.3.4

- `shear_zone_thickness` h Is the shear zone thickness

- `contact_plane_vector` Vector defining the normal to the contact/interface/joint plane.

- `surface_vector_relative_tolerance` defines the relative tolerance to find all the contact/interface/joint normals and create multiple contact elements for a given contact node pairs for a conforming surface-to-surface mesh.

IMPORTANT NOTE No. 1: `contact_plane_vector` defines a direction from Node I to Node J, that is, from the first to the second node. If this normal vector is reversed, the contact/interface/joint element behaves as a hook and is likely to create convergence issues.

IMPORTANT NOTE No. 2: Two nodes that form the Contact/Interface/Joint Element, need to be placed at the same physical location, coordinates in order to prevent convergence issues when nodes are separated and element tries to close the gap in the very first step.

Description of output by this element can be found in Section 206.8.8.

205.3.4.92 Modeling, Finite Element: Stress Based Coupled Hard Contact/Interface/Joint Element with Elastic Perfectly Plastic Shear Behavior

The command is:

```
add element # <.> type StressBasedCoupledHardContact_ElPPlShear
    with nodes (<.>, <.>)
    axial_stiffness = <Pa>
    axial_penalty_stiffness = <Pa>
    initial_shear_stiffness = <Pa>
    axial_viscous_damping = <Pa*s>
    shear_viscous_damping = <Pa*s>
    residual_friction_coefficient = <.>
    shear_zone_thickness = <m>
    contact_plane_vector = (<.>, <.>, <.> );
```

and/or;

```
add element # <.> type StressBasedCoupledHardContact_ElPPlShear
    with nodes (<.>, <.>)
    axial_stiffness = <Pa>
    axial_penalty_stiffness = <Pa>
    initial_shear_stiffness = <Pa>
    axial_viscous_damping = <Pa*s>
    shear_viscous_damping = <Pa*s>
    residual_friction_coefficient = <.>
    shear_zone_thickness = <m>
    surface_vector_relative_tolerance = <.>;
```

The axial stress σ_a and axial stiffness E_a in defined as

$$\begin{aligned} \sigma_a &= b * \epsilon_a \\ E_a &= b \end{aligned} \tag{205.20}$$

where

$E_a = b$ refers to the axial stiffness in axial contact/interface/joint direction,

ϵ_a refers to the axial strain in axial contact direction $\epsilon_a = \delta_a / h$,

δ_a is the relative axial penetration in contact/interface/joint axial direction,

h is the shear zone thickness, and

- `axial_stiffness (b)` represents the stiffness in the axial direction (local x axis).
- `axial_penalty_stiffness` (E_p) defines the penalty stiffness between U_i degree of freedom (DoF) (saturated, coupled u-p-U element) and u_i DoF (dry u element) to enforce movement of fluid in u-p-U element with solid in u element in contact/interface/joint axial direction. This is

useful for pumping action, gap opens and draws the fluid from u-p-U element and then gap closes and pumps, pushes fluid into u-p-U element.

- `initial_shear_stiffness` (E_s) Is the stiffness in the tangential (shear, local y or z axis) directions at $101kPa$ axial stress described in Section 104.7.3.2
- `axial_viscous_damping` Is the viscous damping in axial.
- `shear_viscous_damping` Is the viscous damping in shear.
- `residual_friction_coefficient` (μ_r) Is the residual friction coefficient described in Section 104.7.3.2
- `shear_zone_thickness` h Is the shear zone thickness
- `contact_plane_vector` Vector defining the normal to the contact/interface/joint plane.
- `surface_vector_relative_tolerance` defines the relative tolerance to find all the contact/interface/joint normals and create multiple contact elements for a given contact node pairs for a conforming surface-to-surface mesh.

IMPORTANT NOTE No. 1: `contact_plane_vector` defines a direction from Node I to Node J, that is, from the first to the second node. If this normal vector is reversed, the contact/interface/joint element behaves as a hook and is likely to create convergence issues.

IMPORTANT NOTE No. 2: Two nodes that form the Contact/Interface/Joint Element, need to be placed at the same physical location, coordinates in order to prevent convergence issues when nodes are separated and element tries to close the gap in the very first step.

Description of output by this element can be found in Section 206.8.8.

205.3.4.93 Modeling, Finite Element: Stress Based Coupled Hard Contact/Interface/Joint Element with Nonlinear Hardening Shear Behavior

The command is:

```
add element # <.> type StressBasedCoupledHardContact_NonLinHardShear
    with nodes (<.>, <.>)
    axial_stiffness = <Pa>
    axial_penalty_stiffness = <Pa>
    initial_shear_stiffness = <Pa>
    axial_viscous_damping = <Pa*s>
    shear_viscous_damping = <Pa*s>
    residual_friction_coefficient = <.>
    shear_zone_thickness = <m>
    contact_plane_vector = (<.>, <.>, <.> );
```

and/or;

```
add element # <.> type StressBasedCoupledHardContact_NonLinHardShear
    with nodes (<.>, <.>)
    axial_stiffness = <Pa>
    axial_penalty_stiffness = <Pa>
    initial_shear_stiffness = <Pa>
    axial_viscous_damping = <Pa*s>
    shear_viscous_damping = <Pa*s>
    residual_friction_coefficient = <.>
    shear_zone_thickness = <m>
    surface_vector_relative_tolerance = <.>;
```

The axial stress σ_a and axial stiffness E_a in defined as

$$\begin{aligned} \sigma_a &= b * \epsilon_a \\ E_a &= b \end{aligned} \tag{205.21}$$

where

$E_a = b$ refers to the axial stiffness in axial contact/interface/joint direction,

ϵ_a refers to the axial strain in axial contact direction $\epsilon_a = \delta_a / h$,

δ_a is the relative axial penetration in contact/interface/joint axial direction,

h is the shear zone thickness, and

- `axial_stiffness (b)` represents the stiffness in the axial direction (local x axis) for 1m penetration.
- `axial_penalty_stiffness` (E_p) defines the penalty stiffness between U_i degree of freedom (DoF) (saturated, coupled u-p-U element) and u_i DoF (dry u element) to enforce movement of

fluid in u-p-U element with solid in u element in contact/interface/joint axial direction. This is useful for pumping action, gap opens and draws the fluid from u-p-U element and then gap closes and pumps, pushes fluid into u-p-U element.

- `initial_shear_stiffness` (E_s) Is the stiffness in the tangential (shear, local y or z axis) directions at $101kPa$ axial stress described in Section 104.7.3.3
- `axial_viscous_damping` Is the viscous damping in axial.
- `shear_viscous_damping` Is the viscous damping in shear.
- `residual_friction_coefficient` (μ_r) Is the residual frictional parameter as described in Section 104.7.3.3
- `shear_zone_thickness` h Is the shear zone thickness
- `contact_plane_vector` Vector defining the normal to the contact/interface/joint plane.
- `surface_vector_relative_tolerance` defines the relative tolerance to find all the contact/interface/joint normals and create multiple contact elements for a given contact node pairs for a conforming surface-to-surface mesh.

IMPORTANT NOTE No. 1: `contact_plane_vector` defines a direction from Node I to Node J, that is, from the first to the second node. If this normal vector is reversed, the contact/interface/joint element behaves as a hook and is likely to create convergence issues.

IMPORTANT NOTE No. 2: Two nodes that form the Contact/Interface/Joint Element, need to be placed at the same physical location, coordinates in order to prevent convergence issues when nodes are separated and element tries to close the gap in the very first step.

Description of output by this element can be found in Section 206.8.8.

205.3.4.94 Modeling, Finite Element: Stress Based Coupled Hard Contact/Interface/Joint Element with Nonlinear Hardening and Softening Shear Behavior

The command is:

```
add element # <.> type StressBasedCoupledHardContact_NonLinHardSoftShear
    with nodes (<.>, <.>)
    axial_stiffness = <Pa>
    axial_penalty_stiffness =<Pa>
    initial_shear_stiffness = <Pa>
    rate_of_softening = <>
    size_of_peak_plateau = <>
    axial_viscous_damping = <Pa*s>
    shear_viscous_damping = <Pa*s>
    peak_friction_coefficient_limit = <>
    peak_friction_coefficient_rate_of_decrease = <.>
    residual_friction_coefficient = <.>
    shear_zone_thickness = <m>
    contact_plane_vector = (<.>, <.>, <.> );
```

and/or;

```
add element # <.> type StressBasedCoupledHardContact_NonLinHardSoftShear
    with nodes (<.>, <.>)
    axial_stiffness = <Pa>
    axial_penalty_stiffness =<Pa>
    initial_shear_stiffness = <Pa>
    rate_of_softening = <>
    size_of_peak_plateau = <>
    axial_viscous_damping = <Pa*s>
    shear_viscous_damping = <Pa*s>
    peak_friction_coefficient_limit = <>
    peak_friction_coefficient_rate_of_decrease = <.>
    residual_friction_coefficient = <.>
    shear_zone_thickness = <m>
    surface_vector_relative_tolerance = <.>;
```

The axial stress σ_a and axial stiffness E_a in defined as

$$\begin{aligned} \sigma_a &= b * \epsilon_a \\ E_a &= b \end{aligned} \tag{205.22}$$

where

$E_a = b$ refers to the axial stiffness in axial contact/interface/joint direction,

ϵ_a refers to the axial strain in axial contact direction $\epsilon_a = \delta_a / h$,

δ_a is the relative axial penetration in contact/interface/joint axial direction,

h is the shear zone thickness, and

- `axial_stiffness` (b) represents the stiffness in the axial direction (local x axis) for 1m penetration.

- `axial_penalty_stiffness` (E_p) defines the penalty stiffness between U_i degree of freedom (DoF) (saturated, coupled u-p-U element) and u_i DoF (dry u element) to enforce movement of fluid in u-p-U element with solid in u element in contact/interface/joint axial direction. This is useful for pumping action, gap opens and draws the fluid from u-p-U element and then gap closes and pumps, pushes fluid into u-p-U element.

- `initial_shear_stiffness` (E_s) Is the stiffness in the tangential (shear, local y or z axis) directions at $101kPa$ axial stress, described in Section 104.7.3.4

- `rate_of_softening` (R_s) Is the parameter to control the rate of frictional softening described in Section 104.7.3.4. The frictional softening function is an inverse tangent function raised to power n with incremental form as

$$\Delta\mu = -\frac{n * R_s(\mu_p - \mu_r)}{(\pi/2)^n \theta^{1/n-1}} * cos^2\theta\Delta\gamma^p \tag{205.23}$$

$$\theta = \frac{\mu_p - \mu}{\mu_p - \mu_r}(\pi/2)^n \tag{205.24}$$

where, R_s is the frictional softening rate parameter, $\Delta\gamma^p$ is the plastic shear strain and n represents the size of the peak plateau.

$$\Delta\gamma^p = \sqrt{\Delta\epsilon^p_{ij}\Delta\epsilon^p_{ij}} \tag{205.25}$$

- `size_of_peak_plateau` (n) Is the frictional softening parameter to control the size of plateau as described in Section 104.7.3.4. The frictional softening function is an inverse tangent function raised to power n with incremental form as shown in Equation 205.24.

- `axial_viscous_damping` Is the viscous damping in axial.

- `shear_viscous_damping` Is the viscous damping in shear.

- `peak_friction_coefficient_limit` (μ_{p0}) Is the limit to the peak frictional hardening parameter μ_p.

- `peak_friction_coefficient_rate_of_decrease` (k) Is the rate of decrease of peak frictional hardening parameter μ_p with axial stress, described in Section 104.7.3.4

$$\mu_p = max(\mu_{p0}, \mu_{p0} - k * log(\sigma_a/P_0)) \tag{205.26}$$

where μ_{p0} is the peak frictional hardening limit, k is the peak frictional parameter rate of decrease and P_0 is the reference stress of $P_0 = 101kPa$.

- `residual_friction_coefficient` (μ_r) Is the residual frictional parameter as described in Section 104.7.3.4
- `shear_zone_thickness` h Is the shear zone thickness.
- `contact_plane_vector` Vector defining the normal to the contact/interface/joint plane.
- `surface_vector_relative_tolerance` defines the relative tolerance to find all the contact/interface/joint normals and create multiple contact elements for a given contact node pairs for a conforming surface-to-surface mesh.

IMPORTANT NOTE No. 1: `contact_plane_vector` defines a direction from Node I to Node J, that is, from the first to the second node. If this normal vector is reversed, the contact/interface/joint element behaves as a hook and is likely to create convergence issues.

IMPORTANT NOTE No. 2: Two nodes that form the Contact/Interface/Joint Element, need to be placed at the same physical location, coordinates in order to prevent convergence issues when nodes are separated and element tries to close the gap in the very first step.

Description of output by this element can be found in Section 206.8.8.

205.3.4.95 Modeling, Finite Element: Stress Based Coupled Soft Contact/Interface/Joint Element with Elastic Perfectly Plastic Shear Behavior

The command is:

```
add element # <.> type StressBasedCoupledSoftContact_ElPPlShear
    with nodes (<.>, <.>)
    initial_axial_stiffness = <Pa>
    stiffening_rate = <>
    max_axial_stiffness = <Pa>
    axial_penalty_stiffness = <Pa>
    initial_shear_stiffness = <Pa>
    axial_viscous_damping = <Pa*s>
    shear_viscous_damping = <Pa*s>
    residual_friction_coefficient = <.>
    shear_zone_thickness = <m>
    contact_plane_vector = (<.>, <.>, <.> );
```

and/or;

```
add element # <.> type StressBasedCoupledSoftContact_ElPPlShear
    with nodes (<.>, <.>)
    initial_axial_stiffness = <Pa>
    stiffening_rate = <>
    max_axial_stiffness = <Pa>
    axial_penalty_stiffness = <Pa>
    initial_shear_stiffness = <Pa>
    axial_viscous_damping = <Pa*s>
    shear_viscous_damping = <Pa*s>
    residual_friction_coefficient = <.>
    shear_zone_thickness = <m>
    surface_vector_relative_tolerance = <.>;
```

The axial stress σ_a and axial stiffness E_a in defined as

$$\begin{aligned} \sigma_a &= b * exp(a * \epsilon_a) * \epsilon_a \\ E_a &= max(b * exp(a * \epsilon_a) * (1 + a * \epsilon_a), E_{max}) \end{aligned} \tag{205.27}$$

where

b refers to the initial axial stiffness in axial contact/interface/joint direction,

a refers to the stiffening rate in axial contact direction,

E_{max} refers to the maximum axial stiffness,

E_a refers to the axial stiffness,

ϵ_a refers to the axial strain in axial contact/interface/joint direction $\epsilon_a = \delta_a/h$,

δ_a is the relative axial penetration in contact axial direction,

h is the shear zone thickness, and

- `initial_axial_stiffness` (b) represents the stiffness in the axial direction (local x axis) for 1m penetration.
- `stiffening_rate` (a) Represents exponential stiffening rate $exp(sr * \epsilon_n)$ in axial direction.
- `max_axial_stiffness`(E_{max}) Defines the maximum stiffness in the axial direction (local x axis) for the contact/interface/joint element.
- `axial_penalty_stiffness` (E_p) defines the penalty stiffness between U_i degree of freedom (DoF) (saturated, coupled u-p-U element) and u_i DoF (dry u element) to enforce movement of fluid in u-p-U element with solid in u element in contact/interface/joint axial direction. This is useful for pumping action, gap opens and draws the fluid from u-p-U element and then gap closes and pumps, pushes fluid into u-p-U element.
- `initial_shear_stiffness` (E_s) Is the stiffness in the tangential (shear, local y or z axis) directions at $101kPa$ axial stress described in Section 104.7.3.2
- `axial_viscous_damping` Is the viscous damping in axial.
- `shear_viscous_damping` Is the viscous damping in shear.
- `residual_friction_coefficient` (μ_r) Is the residual friction coefficient described in Section 104.7.3.2
- `shear_zone_thickness` h Is the shear zone thickness
- `contact_plane_vector` Vector defining the normal to the contact/interface/joint plane.
- `surface_vector_relative_tolerance` defines the relative tolerance to find all the contact/interface/joint normals and create multiple contact elements for a given contact node pairs for a conforming surface-to-surface mesh.

IMPORTANT NOTE No 1: `contact_plane_vector` defines a direction from Node I to Node J, that is, from the first to the second node. If this normal vector is reversed, the contact/interface/joint element behaves as a hook and is likely to create convergence issues.

IMPORTANT NOTE No. 2: Two nodes that form the Contact/Interface/Joint Element, need to be placed at the same physical location, coordinates in order to prevent convergence issues when nodes are separated and element tries to close the gap in the very first step.

Description of output by this element can be found in Section ??.

205.3.4.96 Modeling, Finite Element: Stress Based Coupled Soft Contact/Interface/Joint Element with Nonlinear Hardening Shear Behavior

The command is:

```
add element # <.> type StressBasedCoupledSoftContact_NonLinHardShear
    with nodes (<.>, <.>)
    initial_axial_stiffness = <Pa>
    stiffening_rate = <>
    max_axial_stiffness = <Pa>
    axial_penalty_stiffness = <Pa>
    initial_shear_stiffness = <Pa>
    axial_viscous_damping = <Pa*s>
    shear_viscous_damping = <Pa*s>
    residual_friction_coefficient = <.>
    shear_zone_thickness = <m>
    contact_plane_vector = (<.>, <.>, <.> );
```

and/or;

```
add element # <.> type StressBasedCoupledSoftContact_NonLinHardShear
    with nodes (<.>, <.>)
    initial_axial_stiffness = <Pa>
    stiffening_rate = <>
    max_axial_stiffness = <Pa>
    axial_penalty_stiffness = <Pa>
    initial_shear_stiffness = <Pa>
    axial_viscous_damping = <Pa*s>
    shear_viscous_damping = <Pa*s>
    residual_friction_coefficient = <.>
    shear_zone_thickness = <m>
    surface_vector_relative_tolerance = <.>;
```

The axial stress σ_a and axial stiffness E_a in defined as

$$\begin{aligned} \sigma_a &= b * exp(a * \epsilon_a) * \epsilon_a \\ E_a &= max(b * exp(a * \epsilon_a) * (1 + a * \epsilon_a), E_{max}) \end{aligned} \tag{205.28}$$

where

b refers to the initial axial stiffness in axial contact/interface/joint direction,

a refers to the stiffening rate in axial contact direction,

E_{max} refers to the maximum axial stiffness,

E_a refers to the axial stiffness,

ϵ_a refers to the axial strain in axial contact/interface/joint direction $\epsilon_a = \delta_a/h$,

δ_a is the relative axial penetration in contact axial direction,

h is the shear zone thickness, and

- `initial_axial_stiffness (b)` represents the stiffness in the axial direction (local x axis) for 1m penetration.
- `stiffening_rate (a)` Represents exponential stiffening rate $exp(sr * \epsilon_n)$ in axial direction.
- `max_axial_stiffness`(E_{max}) Defines the maximum stiffness in the axial direction (local x axis) for the contact/interface/joint element.
- `axial_penalty_stiffness` (E_p) defines the penalty stiffness between U_i degree of freedom (DoF) (saturated, coupled u-p-U element) and u_i DoF (dry u element) to enforce movement of fluid in u-p-U element with solid in u element in contact/interface/joint axial direction. This is useful for pumping action, gap opens and draws the fluid from u-p-U element and then gap closes and pumps, pushes fluid into u-p-U element.
- `initial_shear_stiffness` (E_s) Is the stiffness in the tangential (shear, local y or z axis) directions at $101kPa$ axial stress described in Section 104.7.3.3
- `axial_viscous_damping` Is the viscous damping in axial.
- `shear_viscous_damping` Is the viscous damping in shear.
- `residual_friction_coefficient` (μ_r) Is the residual frictional parameter as described in Section 104.7.3.3
- `shear_zone_thickness` h Is the shear zone thickness
- `contact_plane_vector` Vector defining the normal to the contact/interface/joint plane.
- `surface_vector_relative_tolerance` defines the relative tolerance to find all the contact/interface/joint normals and create multiple contact elements for a given contact node pairs for a conforming surface-to-surface mesh.

IMPORTANT NOTE No. 1: `contact_plane_vector` defines a direction from Node I to Node J, that is, from the first to the second node. If this normal vector is reversed, the contact/interface/joint element behaves as a hook and is likely to create convergence issues.

IMPORTANT NOTE No. 2: Two nodes that form the Contact/Interface/Joint Element, need to be placed at the same physical location, coordinates in order to prevent convergence issues when nodes are separated and element tries to close the gap in the very first step.

Description of output by this element can be found in Section 206.8.8.

205.3.4.97 Modeling, Finite Element: Stress Based Coupled Soft Contact/Interface/Joint Element with Nonlinear Hardening and Softening Shear Behavior

The command is:

```
add element # <.> type StressBasedCoupledSoftContact_NonLinHardSoftShear
    with nodes (<.>, <.>)
    initial_axial_stiffness = <Pa>
    stiffening_rate = <>
    max_axial_stiffness = <Pa>
    axial_penalty_stiffness =<Pa>
    initial_shear_stiffness = <Pa>
    rate_of_softening = <>
    size_of_peak_plateau = <>
    axial_viscous_damping = <Pa*s>
    shear_viscous_damping = <Pa*s>
    peak_friction_coefficient_limit = <>
    peak_friction_coefficient_rate_of_decrease = <.>
    residual_friction_coefficient = <.>
    shear_zone_thickness = <m>
    contact_plane_vector = (<.>, <.>, <.> );
```

and/or;

```
add element # <.> type StressBasedCoupledSoftContact_NonLinHardSoftShear
    with nodes (<.>, <.>)
    initial_axial_stiffness = <Pa>
    stiffening_rate = <>
    max_axial_stiffness = <Pa>
    axial_penalty_stiffness =<Pa>
    initial_shear_stiffness = <Pa>
    rate_of_softening = <>
    size_of_peak_plateau = <>
    axial_viscous_damping = <Pa*s>
    shear_viscous_damping = <Pa*s>
    peak_friction_coefficient_limit = <>
    peak_friction_coefficient_rate_of_decrease = <.>
    residual_friction_coefficient = <.>
    shear_zone_thickness = <m>
    surface_vector_relative_tolerance = <.>;
```

The axial stress σ_a and axial stiffness E_a in defined as

$$\begin{aligned} \sigma_a &= b * exp(a * \epsilon_a) * \epsilon_a \\ E_a &= max(b * exp(a * \epsilon_a) * (1 + a * \epsilon_a), E_{max}) \end{aligned} \tag{205.29}$$

where

b refers to the initial axial stiffness in axial contact/interface/joint direction,

a refers to the stiffening rate in axial contact direction,

E_{max} refers to the maximum axial stiffness,
E_a refers to the axial stiffness,
ϵ_a refers to the axial strain in axial contact/interface/joint direction $\epsilon_a = \delta_a/h$,
δ_a is the relative axial penetration in contact axial direction,
h is the shear zone thickness, and

- `initial_axial_stiffness` (b) represents the stiffness in the axial direction (local x axis) for 1m penetration.
- `stiffening_rate` (a) Represents exponential stiffening rate $exp(sr * \epsilon_n)$ in axial direction.
- `max_axial_stiffness`(E_{max}) Defines the maximum stiffness in the axial direction (local x axis) for the contact/interface/joint element.
- `axial_penalty_stiffness` (E_p) defines the penalty stiffness between U_i degree of freedom (DoF) (saturated, coupled u-p-U element) and u_i DoF (dry u element) to enforce movement of fluid in u-p-U element with solid in u element in contact/interface/joint axial direction. This is useful for pumping action, gap opens and draws the fluid from u-p-U element and then gap closes and pumps, pushes fluid into u-p-U element.
- `initial_shear_stiffness` (E_s) Is the stiffness in the tangential (shear, local y or z axis) directions at $101kPa$ axial stress, described in Section 104.7.3.4
- `rate_of_softening` (R_s) Is the parameter to control the rate of frictional softening described in Section 104.7.3.4. The frictional softening function is an inverse tangent function raised to power n with incremental form as

$$\Delta\mu = -\frac{n * R_s(\mu_p - \mu_r)}{(\pi/2)^n \theta^{1/n-1}} * cos^2\theta\Delta\gamma^p \tag{205.30}$$

$$\theta = \frac{\mu_p - \mu}{\mu_p - \mu_r}(\pi/2)^n \tag{205.31}$$

  where, R_s is the frictional softening rate parameter, $\Delta\gamma^p$ is the plastic shear strain and n represents the size of the peak plateau.

$$\Delta\gamma^p = \sqrt{\Delta\epsilon_{ij}^p \Delta\epsilon_{ij}^p} \tag{205.32}$$

- `size_of_peak_plateau` (n) Is the frictional softening parameter to control the size of plateau as described in Section 104.7.3.4. The frictional softening function is an inverse tangent function raised to power n with incremental form as shown in Equation 205.31.

- `axial_viscous_damping` Is the viscous damping in axial.
- `shear_viscous_damping` Is the viscous damping in shear.
- `peak_friction_coefficient_limit` (μ_{p0}) Is the limit to the peak frictional hardening parameter μ_p.
- `peak_friction_coefficient_rate_of_decrease` (k) Is the rate of decrease of peak frictional hardening parameter μ_p with axial stress, described in Section 104.7.3.4

$$\mu_p = max(\mu_{p0}, \mu_{p0} - k * log(\sigma_a / P_0)) \tag{205.33}$$

  where μ_{p0} is the peak frictional hardening limit, k is the peak frictional parameter rate of decrease and P_0 is the reference stress of $P_0 = 101kPa$.

- `residual_friction_coefficient` (μ_r) Is the residual frictional parameter as described in Section 104.7.3.4
- `shear_zone_thickness` h Is the shear zone thickness.
- `contact_plane_vector` Vector defining the normal to the contact/interface/joint plane.
- `surface_vector_relative_tolerance` defines the relative tolerance to find all the contact/interface/joint normals and create multiple contact elements for a given contact node pairs for a conforming surface-to-surface mesh.

IMPORTANT NOTE No. 1: `contact_plane_vector` defines a direction from Node I to Node J, that is, from the first to the second node. If this normal vector is reversed, the contact/interface/joint element behaves as a hook and is likely to create convergence issues.

IMPORTANT NOTE No. 2: Two nodes that form the Contact/Interface/Joint Element, need to be placed at the same physical location, coordinates in order to prevent convergence issues when nodes are separated and element tries to close the gap in the very first step.

Description of output by this element can be found in Section 206.8.8.

205.3.4.98 Modeling, Finite Element: Neoprene Isolator Finite Element

(command syntax is in development),

. . .

more on this finite element can be found in Section 102.11 on Page 136 in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).

205.3.4.99 Modeling, Finite Element: Lead Core Rubber Isolator/Dissipator Element

(command syntax is in development),

. . .

more on this finite element can be found in Section 102.11 on Page 136 in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).

205.3.4.100 Modeling, Finite Element: Frictional Pendulum Isolator/Dissipator Finite Element version01

(command syntax is in development),

. . .

more on this finite element can be found in Section 102.11 on Page 136 in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).

205.3.4.101 Modeling, Finite Element: Frictional Pendulum Isolator/Dissipator Finite Element version03

(command syntax is in development),

. . .

more on this finite element can be found in Section 102.11 on Page 136 in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).

205.3.4.102 Modeling, Damping: Adding Rayleigh Damping

First, define the Rayleigh damping.

```
add damping # <.> type Rayleigh with a0 = <1/T> a1 = <T> stiffness_to_use = ↩
   <Initial_Stiffness|Current_Stiffness|Last_Committed_Stiffness>;
```

then apply it to element or node.

```
add damping # <.> to element # <.>;
```

```
add damping # <.> to node # <.>;
```

NOTE:

- If the simulation model is a distributed mass system (e.g. using solid brick elements with nonzero density), users should add damping to elements. In other words, if no additional mass were added to nodes, the command add damping to nodes won't have any effect in ESSI.
- If the simulation model is a lumped mass model (e.g. using the massless beam/truss with lumped mass at nodes), users should add damping to nodes.

205.3.4.103 Modeling, Damping: Adding 3rd Order Caughey Damping

First, define the 3rd-order Caughey damping

```
add damping # <.> type Caughey3rd
with a0 = <T> a1 = <1/T> a2 = <T^3> stiffness_to_use = ↩
   <Initial_Stiffness|Current_Stiffness|Last_Committed_Stiffness>;
```

then apply it to element or node.

```
add damping # <.> to element # <.>;
```

```
add damping # <.> to node # <.>;
```

NOTE:

- If the simulation model is a distributed mass system (e.g. using solid brick elements with nonzero density), users should add damping to elements. In other words, if no additional mass were added to nodes, the command add damping to nodes won't have any effect in ESSI.
- If the simulation model is a lumped mass model (e.g. using the massless beam/truss with lumped mass at nodes), users should add damping to nodes.

205.3.4.104 Modeling, Damping: Adding 4th Caughey Damping

First, define the 4th-order Caughey damping

```
add damping # <.> type Caughey4th
with a0 = <1/T> a1 = <T> a2 = <T^3> a3 = <T^5> stiffness_to_use = ↩
   <Initial_Stiffness|Current_Stiffness|Last_Committed_Stiffness>;
```

then apply it to element or node.

```
add damping # <.> to element # <.>;
```

```
add damping # <.> to node # <.>;
```

NOTE:

- If the simulation model is a distributed mass system (e.g. using solid brick elements with nonzero density), users should add damping to elements. In other words, if no additional mass were added to nodes, the command add damping to nodes won't have any effect in ESSI.
- If the simulation model is a lumped mass system (e.g. using the massless beam/truss with lumped mass at nodes), users should add damping to nodes.

205.3.4.105 Modeling, Constraints and Supports: Adding Constraints or Supports

```
fix node # <.> dofs [ux uy uz p Ux Uy Uz rx ry rz all];
```

where at least one of the DOF fixity codes (`ux uy uz p Ux Uy Uz rx ry rz all`) has to be invoked. These codes are

- `ux`, translation in x direction, for structures and solids (solid phase only in $u-p-U$ and $u-p$ elements)
- `uy`, translation in y direction, for structures and solids (solid phase only in $u-p-U$ and $u-p$ elements)
- `uz`, translation in z direction, for structures and solids (solid phase only in $u-p-U$ and $u-p$ elements)
- `p`, pore fluid pressure (for fluid phase in $u-p-U$ and $u-p$ elements))
- `Ux`, translation of pore fluid phase in x direction (for $u-p-U$ elements)
- `Uy`, translation of pore fluid phase in y direction (for $u-p-U$ elements)
- `Uz`, translation of pore fluid phase in z direction (for $u-p-U$ elements)
- `rx`, rotation around x axes (for structural elements)
- `ry`, rotation around y axes (for structural elements)
- `rz`, rotation around z axes (for structural elements)
- `all`, all applicable DOFs for a given node

Example fix translation x and y for node #3 `fix node # 3 dofs ux uy;`
Example fix all appropriate DOFs for node #7. `fix node # 7 dofs all;`

205.3.4.106 Modeling, Constraints and Supports: Adding Stochastic Constraints or Supports

Define fixities, boundary conditions for a stochastic node. The command would fix all the polynomial chaos expanded dofs associated with the specified physical dof.

```
fix node # <.> stochastic dofs [ux uy uz all];
```

where at least one of the DOF fixity codes (`ux uy uz all`) has to be invoked. These codes are

- `ux`, translation in x direction, including all the associated polynomial chaos expanded dofs.
- `uy`, translation in y direction, including all the associated polynomial chaos expanded dofs.
- `uz`, translation in z direction, including all the associated polynomial chaos expanded dofs.
- `all`, all applicable DOFs for a given node, including all the associated polynomial chaos expanded dofs.

For example,

```
fix node # 3 stochastic dofs ux uy;
```

Fix translation dofs ux and uy, including all the associated polynomial chaos expanded dofs, for stochastic node # 3.

205.3.4.107 Modeling, Constraints and Supports: Free Constraint or Support

Free the specified DOFs on a designated node.

```
free node # <.> dofs [ux uy ux p Ux Uy Uz rx ry rz];
```

205.3.4.108 Modeling, Constraints and Supports: Add Tied/Connected Main-Fillower Nodes for the Same DOFs

Add the equal dof for tied/connected nodes for the same degree of freedom.

```
add constraint equal_dof with
master node # <.> and
slave node # <.>
dof to constrain <.>;
```

205.3.4.109 Modeling, Constraints and Supports: Adding Tied/Connected, Main-Follower Nodes for Different DOFs

Add the equal dof for tied/connected nodes for different degree of freedom.

```
add constraint equal_dof with node # <.> dof <.> master and node # <.> dof <.> ↩
    slave;
```

205.3.4.110 Modeling, Constraints and Supports: Remove Tied/Connected Main-Follower equal DOFs

Remove the tied/connected nodes equal_dofs.

```
remove constraint equal_dof node # <.>
```

205.3.4.111 Modeling, Constraints and Supports: Adding Single Point Constraint to Nodes

Define the single point constraint to nodes on a particular degree of freedom for a specified value.

```
add single point constraint to node # <.>
dof to constrain <dof_type>
constraint value of <.>
```

205.3.4.112 Modeling, Acceleration Field: Adding Acceleration/Inertia Field

```
add acceleration field # <.>
    ax = <acceleration in x direction>*[L/T^2]
    ay = <acceleration in y direction>*[L/T^2]
    az = <acceleration in z direction>*[L/T^2];
```

Example adding acceleration induced loading field for (some) elements

```
add acceleration field # 1
ax = 0*m/s^2
ay = 0*m/s^2
az = -9.81*m/s^2;
```

NOTE: see note on page 1004 for command

```
add load # <.> to element # <.> type self_weight use acceleration field # <.>;
```

205.3.4.113 Modeling, Loads: Nodal Loads

The general signature to add loads is

```
add load # <.> to node # <.>
   type <load type> <direction> = <force_amplitude>
   {more parameters};
```

The load # is a unique number assigned to each load. The node # is the number of a node which has already been defined. The load type refers to the functional form in time or pseudo-time (for static analysis) and can be any of the list

- `linear` Constant rate time dependence.
- `path` Use an arbitrary function defined in an external file.

Each force type except linear have additional parameters which will be explained later.

The force direction refers to the degree of freedom the force will be added to. These force directions are the conjugate in energy of the DOFs defined earlier. These are,

- `Fx`, force in x direction [1]
- `Fy`, force in y direction [1]
- `Fz`, force in z direction [1]
- `F_fluid_x`, force to the pore fluid phase in x direction [2]
- `F_fluid_y`, force to the pore fluid phase in y direction [2]
- `F_fluid_z`, force to the pore fluid phase in z direction [2]
- `Mx`, moment about x axes [3]
- `My`, moment about y axes [3]
- `Mz`, moment about z axes [3]

Example command for adding three linear forces ($fx = -10 * kN, fy = -10 * kN, fz = -10 * kN$) to node # 1:

[1] Applies to solid phase only when connected to coupled elements

[1] Applies to fluid phase when connected to coupled elements. HOWEVER, please note that these are NOT pore fluid pressures, see section 102.12.1.5 on page 148, in Lecture Notes (Jeremić et al., 1989-2025) (Lecture Notes URL).

[1] For elements with rotational DOFs, i.e. beams, shells

```
add load # 1 to node #1 type linear Fx = -10*kN;
add load # 2 to node #1 type linear Fy = -10*kN;
add load # 3 to node #1 type linear Fz = -10*kN;
```

The force `type` refers to the functional dependence in time (or pseudo-time) that the force will have. The possible functional forms have been listed before. Listed are additional parameters which define these forces.

1. `linear`

   Receives no extra parameters. In this case the magnitude of the force is interpreted as the magnitude of the force after one second of time (or pseudo-time) has passed.

2. `path`

   How to add path loads is in the next page.

205.3.4.114 Modeling, Loads: Nodal Path Loads

To add forces which follow a path other than linear, we have the `path_series`, for equally spaced time series data, and `path_time_series` for variable spaced time series data.

The commands are:

```
add load # <.> to node # <.> type path_series
  FORCETYPE = <force or moment scale factor>
  time_step = <T>
  series_file = "STRING";
```

```
add load # <.> to node # <.> type path_time_series
  FORCETYPE = <force or moment scale factor>
  series_file = "STRING";
```

As before, `FORCETYPE` can be `Fx`, `Fy`, `Fz`, `Mx`, `My`, `Mz`, `F_fluidx`, `F_fluidy`, `F_fluidz`.

The format of the series_file is one column of text for the equally spaced case (`path_series`) and double column, one for time and second one for data values, for (`path_time_series`).

205.3.4.115 Modeling, Loads: Nodal Loads From Reactions

Loads can be added from reactions.

The command is:

```
add load # <.> to node # <.> type from_reactions;
add load # <.> to all nodes type from_reactions;
```

The load # is a unique number assigned to each load. The node # is the number of a node which has already been defined. This DSL applies an external load equal to the reaction calculated at that node. It is useful, for stage loading where a constrained dof gets relaxed. The first command add load for a specified node whereas, the second command applies load for all nodes.

For example:

```
add load # 3 to node #5 type from_reactions;
```

Adds an external load to node 5 from its reaction force calculated in previous stage.

```
add load # 3 to all nodes type from_reactions;
```

Adds an external load to all nodes from their reaction force calculated in previous stage.

205.3.4.116 Modeling, Loads: Selfweight Element Load

```
add load # <.>
to element # <.>
type self_weight
use acceleration field # <.>;
```

NOTE: since the gravity acceleration field is $g = 9.81\text{m/s}^2$, meaning that there is an increment of 9.81m/s of velocity each second (please note that this defines a rate of increase in velocity), gravity is then applied in 1 second! This is sometimes (most of the time) too harsh numerically! It helps if one defines an acceleration field of say 0.0981m/s^2 and then apply it in 100 seconds. This is to be done in command `add acceleration field ...` on page 999.

205.3.4.117 Modeling, Loads: Selfweight Nodal Load

```
1 add load # <.> to node # <.> type self_weight use acceleration field # <.>;
```

NOTE: For this command to take effect, there should be concentrated mass defined at the node.

205.3.4.118 Modeling, Loads: 8 Node Brick Surface Load with the Constant Pressure

Surface of 8 node brick element with same pressure magnitudes at all nodes:

```
add load # <.> to element # <.> type surface at nodes (<.> , <.> , <.> , <.>) ↩
    with magnitude <Pa>;
```

Note: This command works for the dry `8NodeBrick` element and the coupled `8NodeBrick_upU` element. For the coupled upU element, this command applies all surface load on the solid phase, simulating a drained surface loading condition.

A new command for undrained surface loading on upU element will be added soon...

205.3.4.119 Modeling, Loads: 8 Node Brick Surface Load with Variable Pressure

Surface of 8 node brick element with variable pressure magnitudes at all nodes:

```
1 add load # <.> to element # <.> type surface at nodes (<.> , <.> , <.> , <.>) ↩
     with magnitudes ( <Pa> , <Pa> , <Pa> , <Pa>);
```

205.3.4.120 Modeling, Loads: 20Node Brick Surface Load with the Constant Pressure

Surface of 20 node brick element with same pressure magnitudes at all nodes:

```
add load # <.> to element # <.> type surface at nodes (<.> , <.> , <.> , <.>, <.>, <.>, <.>, <.>) with magnitude <Pa>;
```

205.3.4.121 Modeling, Loads: 20 Node Brick, Surface Load with Variable Pressure

Surface of 20 node brick element with variable pressure magnitudes at all nodes:

```
add load # <.> to element # <.> type surface at nodes (<.> , <.> , <.> , <.>, <.>, <.>, <.>, <.>) with magnitudes ( <Pa> , <Pa> , <Pa> , <Pa>, <Pa>, <Pa>, <Pa>, <Pa>);
```

205.3.4.122 Modeling, Loads: 27 Node Brick Surface Load with the Constant Pressure

Surface of 27 node brick element with same pressure magnitudes at all nodes:

```
1 add load # <.> to element # <.> type surface at nodes (<.> , <.> , <.> , <.>, ↩
     <.>, <.>, <.>, <.>, <.>) with magnitude <Pa>;
```

205.3.4.123 Modeling, Loads: 27 Node Brick Surface Load with Variable Pressure

Surface of 27 node brick element with variable pressure magnitudes at all nodes:

```
1 add load # <.> to element # <.> type surface at nodes (<.> , <.> , <.> , <.>, ↩
     <.>, <.>, <.>, <.>, <.>) with magnitudes ( <Pa> , <Pa> , <Pa> , <Pa>, <Pa>, ↩
     <Pa>, <Pa>, <Pa>, <Pa>);
```

205.3.4.124 Modeling, Loads: Removing Loads

Loads can be removed using:

```
remove load # <.>
```

205.3.4.125 Modeling, Loads: Domain Reduction Method, DRM

```
add load # <.> type domain reduction method hdf5_file = <string>;
add load # <.> type domain reduction method hdf5_file = <string> scale_factor = <.>;
```

- `hdf5_file` HDF5 file with information for the DRM specification. See section....
- `scale_factor` Factor to linearly scale the motion.

Creating DRM input in HDF5 format. As shown in Fig.(205.5), eight components are required for the DRM input.

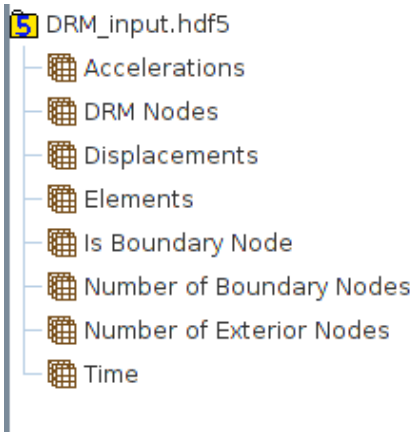


Figure 205.5: Components of DRM input in HDF5 format.

The name of the sub-folders must be exactly the same as shown here.

1. Elements: element numbers for DRM elements, a single layer of elements used to add the earthquake motion.
2. DRM Nodes: Node numbers for DRM elements.
3. Is Boundary Node: used to describe whether each of nodes in "DRM Nodes" is a boundary node or an exterior node.
   - If this value is "1", the corresponding node in "DRM Nodes" is a boundary node.
   - If this value is "0", the corresponding node in "DRM Nodes" is an exterior node in the DRM element.
4. Number of Boundary Nodes: the number of boundary nodes.

5. Number of Exterior Nodes: the number of exterior nodes.

6. Displacements: the displacement components of input earthquake motion on corresponding DRM Nodes. Displacements are a 2D array,

   - column number represents a time-step,
   - rows represents the displacement in one direction.

     If every node has 3 degrees of freedom (DOFs, say ux, uy, and uz), the first three rows represent the input displacements on first DRM node in directions of ux, uy, and uz. Then, next three rows represent the input displacements for the next node. So the total row number should be three times of the number of DRM Nodes.

7. Accelerations: same data structure as displacements, above.

8. Time: real time for each time-step in the input earthquake motion.

Python example script to generate the DRM HDF5-based input is given below. Please note that script only generates simplest possible rigid body motion, and that for any realistic motions, those will have to be created using 1C or 3C seismic motions codes, for example, SW4, SynAcc, fk, Hisada, EDT, MS ESSI, or even SHAKE for 1C motions.

```
# Created by Jose Antonio Abell Mena
# This file reads old-format DRM input files and translates them into new
# HDF5-based format.
#

# This file produces a rigid body input to the DRM layer. That is, all DRM
# nodes have same X-direction displacement and acceleration. In this case a
# sine wave is used. This is not realistic, its just for demonstration
# purposes. DRM won't work in this case but can be used to verify input if a
# pseudo-static analysis is done (zero density on all elements and apply loads
# with transient analysis.)
# For real input motions, produced using some other means, say SW4, fk,
# Hisada or MS ESSI program, this simple program can be used as an example,
# where DRM input file format used here can be (re) used, while those real
# motions are read form output of above mentioned programs.

import scipy as sp
import h5py
import time

# Write elements and nodes data
elements = sp.loadtxt("DRMelements.txt",dtype=sp.int32)
exterior_nodes = sp.loadtxt("DRMexterior.txt",dtype=sp.int32)
boundary_nodes = sp.loadtxt("DRMbound.txt",dtype=sp.int32)
```

```
25
26 Ne = sp.array(exterior_nodes.size)
27 Nb = sp.array(boundary_nodes.size)
28
29 Nt = Ne+Nb
30
31 all_nodes = sp.hstack((boundary_nodes, exterior_nodes))
32 is_boundary_node = sp.zeros(Nt, dtype=sp.int32)
33 is_boundary_node[0:Nb] = 1
34
35 h5file = h5py.File("small.h5.drminput","w")
36
37 h5file.create_dataset("Elements", data=elements)
38 h5file.create_dataset("DRM Nodes", data=all_nodes)
39
40 # This array has 1 if the node at the corresponding position in "DRM nodes"
41 # array is a boundary node and zero if not
42 h5file.create_dataset("Is Boundary Node", data=is_boundary_node)
43
44 h5file.create_dataset("Number of Exterior Nodes", data=Ne)
45 h5file.create_dataset("Number of Boundary Nodes", data=Nb)
46
47 # Write timestamp (time format used is that of c "asctime" Www Mmm dd
48 # hh:mm:ss yyyy example: Tue Jan 13 10:17:09 2009)
49 localtime = time.asctime( time.localtime(time.time()) )
50 h5file.create_dataset("Created",data=str(localtime))
51
52 # Generate motions
53 t = sp.linspace(0,10,1001)
54 w = 2*sp.pi/0.5
55 d = sp.sin(w*t)
56 a = -w**2*sp.sin(w*t)
57
58 # Output accelerations, displacements and time-vector
59
60 # Format is:
61 #
62 # Array/matrix for Accelerations and Displacements has the following shape
63 # [3*(N_boundary_nodes + N_exterior_nodes) , Ntimesteps]
64 #
65 # where component
66 # A[3*n], A[3*n+1], A[3*n+2]
67 # correspond to accelerations/displacements in X, Y, and Z directions
68 # at node n.
69 # The location corresponding to node n is that of the n-th component of array
70 # "DRM Nodes"
71
72 # Time vector
73
74 h5file.create_dataset("Time", data=t)
75
```

```
acc = h5file.create_dataset("Accelerations", (3*Nt,len(t)), dtype=sp.double)
dis = h5file.create_dataset("Displacements", (3*Nt,len(t)), dtype=sp.double)

for node_index in range(Nt):
 acc[3*node_index,:] = a
 acc[3*node_index+1,:] = 0*a #Zero acceleration in y and z
 acc[3*node_index+2,:] = 0*a
 dis[3*node_index,:] = d
 dis[3*node_index+1,:] = 0*d #Zero displacement in y and z
 dis[3*node_index+2,:] = 0*d

h5file.close()
```

205.3.4.126 Modeling, Wave Field for Creating DRM Loads: Add Wave Field

```
add wave field # <.> with
    acceleration_filename = <string>
    unit_of_acceleration = <L/T^2>
    displacement_filename = <string>
    unit_of_displacement = <L>
    add_compensation_time = <T>
    motion_depth = <L>
    monitoring_location = <within_soil_layer|equivalent_rock_outcropping>
    soil_profile_filename = <string>
    unit_of_Vs = <L/T>
    unit_of_rho = <M/L^3>
    unit_of_damping = <absolute|percent>
    unit_of_thickness = <L>
    ;
```

Example adding a wave field

```
add wave field # 1 with
    acceleration_filename = "acc.txt"
    unit_of_acceleration = 1 * m/s^2
    displacement_filename = "dis.txt"
    unit_of_displacement = 1 * m
    add_compensation_time = 0.5 * s
    motion_depth = 0 * m
    monitoring_location = within_soil_layer
    soil_profile_filename = "soil_profile.txt"
    unit_of_Vs = 1 * m/s
    unit_of_rho = 1 * kg/m^3
    unit_of_damping = absolute
    unit_of_thickness = 1*m
    ;
```

where:

- `No (or #)<.>` is the unique wave field ID. The wave field ID does not have to be sequential, any unique positive integer number can be used. Each wave field is a 1C wave field. The wave field does not have a direction. Later, if users want to add load with the wave field, users should specify the direction with each wave field.

- `acceleration_filename` is the filename of a plain text file, which contains the acceleration of the input motion. The file should have two columns, where the first column is the accumulated time and the second column is the corresponding acceleration. For the DRM loading from a wave field, if the simulation time is longer than the earthquake motion, the remaining simulation will continue with zero motions. The wave field does NOT conduct any base correction on the input

motion, so the simulation results may have permanent deformation after the earthquake. If users want to have base corrections, users should pre-process the earthquake motion by themselves.

- `displacement_filename` is the filename of a plain text file, which contains the displacement of the input motion. The file should have two columns, where the first column is the accumulated time and the second column is the corresponding displacement. For the DRM loading from a wave field, if the simulation time is longer than the earthquake motion, the remaining simulation will continue with zero motions. The wave field does NOT conduct any base correction on the input motion, so the simulation results may have permanent deformation after the earthquake. If users want to have base corrections, users should pre-process the earthquake motion by themselves.

- `add_compensation_time` is a feature to add zero-motion in the beginning and at the end of the earthquake motion. Since the wave propagation theory is solving the wave equation in frequency domain (steady state), without additional zeros, the beginning and the end of wave may be mixed up. If the user does not want to add the additional zeros, user can specify 0*s.

- `motion_depth` is the depth of the input motion. Usually, the `motion_depth` is at the surface (namely, 0*m). Later, users can specify the request depth for deconvolution. However, users can also specify a specific depth of the input motion. In this case, users can request both convolution and deconvolution.

  - If the request depth is deeper than this input acceleration depth, the wave propagation will generate the deconvolution results.
  - If the request depth is shallower than this input acceleration depth, the wave propagation will generate the convolution results. It is recommended to add a small damping for wave convolution.

  The acceleration depth is the relative depth to the soil surface, so both negative and positive depth are acceptable and result in the same results.

- `monitoring_location` is the location of the earthquake monitoring station. When the monitoring location is within soil layer, the wave propagation is conducted inside the soil layer directly. When the monitoring location is equivalent rock outcropping, wave deconvolution is conducted back to the bedrock first and then propagate into the soil layers.

- `soil_profile_filename` contains the soil properties for each layer. The soil profile file should have four columns, which are the shear wave velocity, density, damping ratio and thickness of each

layer respectively. The soil layers should be from the soil surface to the bedrock. The last layer is bedrock, which has three columns only. User should NOT give the thickness of the last layer.

One Example of soil profile file is given below.

```
// Vs rho damp thickness
200 2000 0.03 35
250 2000 0.04 35
2000 2400 0.05
```

205.3.4.127 Modeling, Wave Field for Creating DRM Loads: Deconvolution

This command performs deconvolution or convolution of given motions at surface or certain depth and writes accelerations, velocities and displacements in 3 directions, in files, that can then be used to create DRM loads.

```
generate wave propagation results of wave field
  # <.> at depth <L> to file <output_filename_prefix>
```

One example of deconvolution is

```
generate wave propagation results of wave field
  # 1 at depth -60*m to file "Northridge_record" ;
```

where

- `wave field # <.>` specifies the wave field number which will be used for wave propagation.
- `depth <L>` is the request depth of the output motion.
  - If the request depth is deeper than the input motion that defined in the wave field, this command will generate the deconvolution results.
  - If the request depth is shallower than the input motion that defined in the wave field, this command will generate the convolution results. It is recommended to add a small damping for wave convolution.

  The depth specifies the *relative* location between the soil surface and the request depth, which means both positive and negative depth are acceptable and will result in the same results.
- `output_filename_prefix` specified the prefix of the output filenames. This command will generate 3 output files, whose suffix are `at_str(depth)_acc.txt` , `at_str(depth)_vel.txt` , `at_str(depth)_dis.txt` .

205.3.4.128 Modeling, Wave Field for Creating DRM Input: Motions

This command performs deconvolution or convolution of given motions at surface or certain depth and directly generates DRM motions in 1, 2, or 3 directions.

```
generate DRM motion file from wave field
  # <.> in direction <ux|uy|uz>
  soil_surface at z = <L>
  hdf5_file = <string> ;
```

```
generate DRM motion file from wave field
  # <.> in direction <ux|uy|uz>
  # <.> in direction <ux|uy|uz>
  soil_surface at z = <L>
  hdf5_file = <string> ;
```

```
generate DRM motion file from wave field
  # <.> in direction <ux|uy|uz>
  # <.> in direction <ux|uy|uz>
  # <.> in direction <ux|uy|uz>
  soil_surface at z = <L>
  hdf5_file = <string> ;
```

One example of deconvolution to DRM is

```
generate DRM motion file from wave field
  # 1 in direction ux
  soil_surface at z = 0*m
  hdf5_file = "input.hdf5" ;
```

where:

- `in direction <ux,uy,uz>` specifies the direction of the wave field. Each wave field is a 1C wave field. At most 3 wave fields can be associated with the load.
- `soil_surface` specifies the relation between the FEM coordinate systems and the soil profile depths inside the wave field. The soil surface should always be above the DRM nodes. Namely, soil surface is generally the surface between the soil and the structure, NOT the bedrock surface.
- `hdf5_file` specifies the HDF5 file which contain the information about the DRM elements and DRM nodes.

205.3.4.129 Modeling, Wave Field for Creating DRM Input: Forces

This command performs deconvolution or convolution of given motions at surface or certain depth and directly generates DRM motions and forces in 1, 2, or 3 directions.

```
generate DRM force file from wave field
  # <.> in direction <ux|uy|uz>
  soil_surface at z = <L>
  hdf5_file = <string> ;
```

```
generate DRM force file from wave field
  # <.> in direction <ux|uy|uz>
  # <.> in direction <ux|uy|uz>
  soil_surface at z = <L>
  hdf5_file = <string> ;
```

```
generate DRM force file from wave field
  # <.> in direction <ux|uy|uz>
  # <.> in direction <ux|uy|uz>
  # <.> in direction <ux|uy|uz>
  soil_surface at z = <L>
  hdf5_file = <string> ;
```

One example of deconvolution to DRM is

```
generate DRM force file from wave field
  # 1 in direction ux
  soil_surface at z = 0*m
  hdf5_file = "input.hdf5" ;
```

where:

- `in direction <ux,uy,uz>` specifies the direction of the wave field. Each wave field is a 1C wave field. At most 3 wave fields can be associated with the load.
- `soil_surface` specifies the relation between the FEM coordinate systems and the soil profile depths inside the wave field. The soil surface should always be above the DRM nodes. Namely, soil surface is generally the surface between the soil and the structure, NOT the bedrock surface.
- `hdf5_file` specifies the HDF5 file which contain the information about the DRM elements and DRM nodes.

205.3.4.130 Modeling, Wave Field for Creating DRM Loads: Add Inclined Plane Wave Field from Incident SV Wave Potential Magnitude

```
add wave field # <.> type inclined_plane_wave with
    anticlockwise_angle_of_SV_wave_plane_from <x|y|z> = <degrees>
    SV_incident_magnitude = <L^2>
    SV_incident_angle = <degrees>
    SV_incident_frequency = <1/T>
    motion_time_step = <T>
    number_of_time_steps = <.>
    soil_profile_filename = <string>
    soil_surface at <x|y|z> = <L>
    unit_of_vs_and_vp = <L/T>
    unit_of_rho = <M/L^3>
    unit_of_damping = <absolute|percent>
    unit_of_thickness = <L>
    ;
```

Example of adding an inclined plane wave field

```
add wave field # 1 type inclined_plane_wave with
  anticlockwise_angle_of_SV_wave_plane_from x= 30
  SV_incident_magnitude = 2*m^2
  SV_incident_angle = 60
  SV_incident_frequency = 5/s
  motion_time_step = 0.01*s
  number_of_time_steps = 600
  soil_profile_filename = "soil.txt"
  soil_surface at z = 0*m
  unit_of_vs_and_vp = 1*m/s
  unit_of_rho = 1*kg/m^3
  unit_of_damping = absolute
  unit_of_thickness = 1*m;
```

where:

- `No (or #)<.>` is the unique wave field ID/number. The wave field ID does not have to be sequential, any unique positive integer number can be used. Each wave field is an inclined plane SV wave field.
- `anticlockwise_angle_of_SV_wave_plane_from <x|y|z>` specifies the orientation of the inclined wave field propagation plane. User should give the anticlockwise angle in degrees between the wave propagation plane and the specified reference axis. The reference axis could be x or y or z. As shown in figure 205.7, the `anticlockwise_angle_of_SV_wave_plane_from` x axis is α.
- `SV_incident_magnitude` specifies the incident SV wave potential magnitude. The displacement magnitude of incident SV wave is related to the potential magnitude as follows: $|u| = \phi\omega/V_s$, where

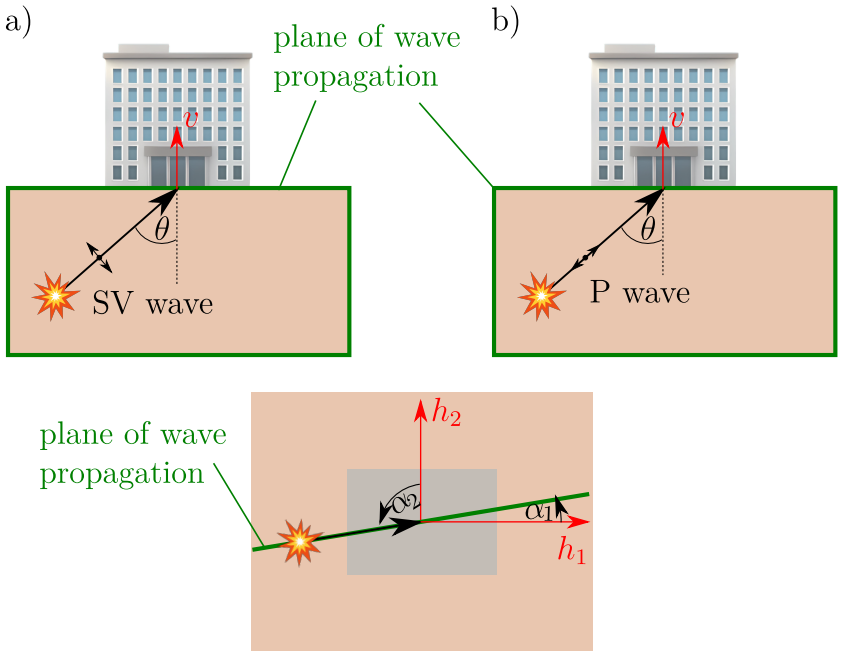


Figure 205.6: Orientation of an inclined plane wave with respect to the vertical, v, and horizontal, h_1, h_2, directions: a) a SV wave and b) a P-wave. The user defines v, h_1, h_2 directions. Either α_1 or α_2 can be input into the DSL command.

$|u|$ is the displacement magnitude, ϕ is potential magnitude, ω is incident angular frequency and V_s is the shear wave velocity of the incident soil/rock layer.

- `SV_incident_angle` specifies the inclination angle of incident SV wave, measured from vertical axis of wave plane to the wave propagation axis. In figure 205.7, the incident angle of SV wave is θ.
- `motion_time_step` is the time step/interval used for discretizing the harmonic motion into time domain.
- `number_of_time_steps` is the number of total time steps for the discretized harmonic motion.
- `soil_profile_filename` is a file name for a file that contains the soil properties for each layer. The soil profile file should have fives columns: (i) shear wave velocity, (ii) compressional wave velocity, (iii) density, (iv) damping ratio and (v) thickness of each layer respectively. The soil layers count from the soil surface to the bedrock. The last layer is bedrock, which has four columns only. User should NOT give the thickness of the last layer, as it extends into halfspace.

  One Example of soil profile file is given below.

```
// Vs Vp rho damp thickness
  200 333 2000 0.02 100
  250 408 2000 0.02 200
 2000 3400 2400 0.02
```

- `soil_surface at <x|y|z>` defines the location of soil surface in the global coordinate system of Real-ESSI.

205.3.4.131 Modeling, Wave Field for Creating DRM Loads: Add Inclined Plane Wave Field from Incident SV Wave Time Series Signal

```
add wave field # <.> type inclined_plane_wave with
    anticlockwise_angle_of_SV_wave_plane_from <x|y|z> = <degrees>
    SV_incident_acceleration_filename = <string>
    unit_of_acceleration = <L/T^2>
    SV_incident_displacement_filename = <string>
    unit_of_displacement = <L>
    SV_incident_angle = <degrees>
    add_compensation_time = <T>
    source_location = (<T>, <T>, <T>)
    soil_profile_filename = <string>
    soil_surface at <x|y|z> = <L>
    unit_of_vs_and_vp = <L/T>
    unit_of_rho = <M/L^3>
    unit_of_damping = <absolute|percent>
    unit_of_thickness = <L>
    ;
```

Example of adding an inclined plane wave field

```
add wave field # 1 type inclined_plane_wave with
  anticlockwise_angle_of_SV_wave_plane_from x = 0
  SV_incident_acceleration_filename = "Kobe_acc.txt"
  unit_of_acceleration = 1*m/s^2
  SV_incident_displacement_filename = "Kobe_disp.txt"
  unit_of_displacement = 1*m
  SV_incident_angle = 15
  add_compensation_time = 0.5*s
  source_location = (-150*m, 0*m, -100*m)
  soil_profile_filename = "soil_profile.txt"
  soil_surface at z = 0*m
  unit_of_vs_and_vp = 1*m/s
  unit_of_rho = 1*kg/m^3
  unit_of_damping = absolute
  unit_of_thickness = 1*m;
```

where:

- `No (or #)<.>` is the unique wave field ID/number. The wave field ID does not have to be sequential, any unique positive integer number can be used. Each wave field is an inclined plane SV wave field.
- `anticlockwise_angle_of_SV_wave_plane_from <x|y|z>` specifies the orientation of the inclined wave field propagation plane. User should give the anticlockwise angle in degrees between the wave propagation plane and the specified reference axis. The reference axis could be x or y or

z. As shown in figure 205.7, the `anticlockwise_angle_of_SV_wave_plane_from` x axis is α.

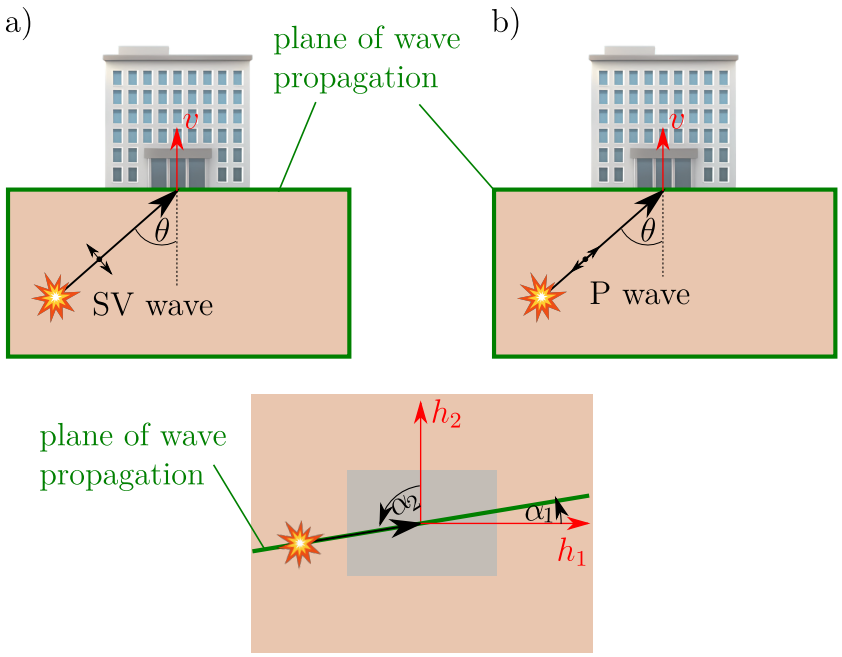


Figure 205.7: Orientation of an inclined plane wave with respect to the vertical, v, and horizontal, h_1, h_2, directions: a) a SV wave and b) a P-wave. The user defines v, h_1, h_2 directions. Either α_1 or α_2 can be input into the DSL command.

- `SV_incident_acceleration_filename` is the filename of a plain text file, which contains the acceleration of the input motion. The file should have two columns, where the first column is the accumulated time and the second column is the corresponding acceleration. For the DRM loading from a wave field, if the simulation time is longer than the earthquake motion, the remaining simulation will continue with zero motions. The wave field implementation in Real-ESSI does NOT conduct any baseline correction on the input motion. The users should perform baseline correction for earthquake motions by themselves.

- `SV_incident_displacement_filename` is the filename of a plain text file, which contains the displacement of the input motion. The file should have two columns, where the first column is the accumulated time and the second column is the corresponding displacement. For the DRM loading from a wave field, if the simulation time is longer than the earthquake motion, the remaining simulation will continue with zero motions. The wave field implementation in Real-ESSI does NOT conduct any baseline correction on the input motion. The users should perform baseline correction for earthquake motions by themselves.

- `SV_incident_angle` specifies the inclination angle of incident SV wave, measured from vertical axis of wave plane to the wave propagation axis. In figure 205.7, the incident angle of SV wave is θ.

- `add_compensation_time` is a feature to add zero-motion in the beginning and at the end of the earthquake motion. If the user does not want to add the additional zeros, user can specify compensation time as 0*s.

- `source_location` specify the location of seismic source, where the seismic motion is input as in `SV_incident_acceleration_filename` and `SV_incident_displacement_filename`, it is used for determining phase of the wave, and the source location can be inside or outside of the model.

- `soil_profile_filename` is a file name for a file that contains the soil properties for each layer. The soil profile file should have fives columns: (i) shear wave velocity, (ii) compressional wave velocity, (iii) density, (iv) damping ratio and (v) thickness of each layer respectively. The soil layers count from the soil surface to the bedrock. The last layer is bedrock, which has four columns only. User should NOT give the thickness of the last layer, as it extends into half-space.

  One Example of soil profile file is given below.

```
// Vs Vp rho damp thickness
  200 333 2000 0.02 100
  250 408 2000 0.02 200
 2000 3400 2400 0.02
```

- `soil_surface at <x|y|z>` defines the location of soil surface in the global coordinate system of Real-ESSI.

205.3.4.132 Modeling, Wave Field for Creating DRM Loads: Add Inclined Plane Wave Field from Incident P Wave Potential Magnitude

```
add wave field # <.> type inclined_plane_wave with
    anticlockwise_angle_of_P_wave_plane_from <x|y|z> = <degrees>
    P_incident_magnitude = <L^2>
    P_incident_angle = <degrees>
    P_incident_frequency = <1/T>
    motion_time_step = <T>
    number_of_time_steps = <.>
    soil_profile_filename = <string>
    soil_surface at <x|y|z> = <L>
    unit_of_vs_and_vp = <L/T>
    unit_of_rho = <M/L^3>
    unit_of_damping = <absolute|percent>
    unit_of_thickness = <L>
    ;
```

Example adding an inclined plane wave field

```
add wave field # 1 type inclined_plane_wave with
  anticlockwise_angle_of_P_wave_plane_from x= 30
  P_incident_magnitude = 2*m^2
  P_incident_angle = 60
  P_incident_frequency = 5/s
  motion_time_step = 0.01*s
  number_of_time_steps = 600
  soil_profile_filename = "soil.txt"
  soil_surface at z = 0*m
  unit_of_vs_and_vp = 1*m/s
  unit_of_rho = 1*kg/m^3
  unit_of_damping = absolute
  unit_of_thickness = 1*m;
```

where:

- `No (or #)<.>` is the unique wave field ID/number. The wave field ID does not have to be sequential, any unique positive integer number can be used. Each wave field is an inclined plane P wave field.
- `anticlockwise_angle_of_P_wave_plane_from <x|y|z>` specifies the orientation of the inclined wave propagation plane. User should give the anticlockwise angle in degrees between the wave propagation plane and the specified reference axis. The reference axis could be x or y or z. As shown in figure 205.7, the `anticlockwise_angle_of_P_wave_plane_from` x axis is α.
- `P_incident_magnitude` specifies the incident P wave potential magnitude. The displacement magnitude of incident P wave is related to the potential magnitude as following: $|u| = \phi\omega/V_p$,

where $|u|$ is the displacement magnitude, ϕ is potential magnitude, ω is incident angular frequency and V_p is the compressional wave velocity of the incident layer.

- `P_incident_angle` specifies the inclination of incident P wave. The angle is measured from vertical axis of wave plane to the wave propagation axis. In figure 205.7, the incident angle of P wave is θ.

- `motion_time_step` is the time interval when discretized the harmonic motion into time domain.

- `number_of_time_steps` is the number of total time steps for the discretized harmonic motion.

- `soil_profile_filename` is a file name for a file that contains the soil properties for each layer. The soil profile file should have fives columns: (i) shear wave velocity, (ii) compressional wave velocity, (iii) density, (iv) damping ratio and (v) thickness of each layer respectively. The soil layers count from the soil surface to the bedrock. The last layer is bedrock, which has four columns only. User should NOT give the thickness of the last layer, as it extends into halfspace.

  One Example of soil profile file is given below.

```
// Vs Vp rho damp thickness
  200 333 2000 0.02 100
  250 408 2000 0.02 200
 2000 3400 2400 0.02
```

- `soil_surface at <x|y|z>` defines the location of soil surface in the global coordinate system of Real-ESSI.

205.3.4.133 Modeling, Wave Field for Creating DRM Loads: Add Inclined Plane Wave Field from Incident P Wave Time Series Signal

```
add wave field # <.> type inclined_plane_wave with
    anticlockwise_angle_of_P_wave_plane_from <x|y|z> = <degrees>
    P_incident_acceleration_filename = <string>
    unit_of_acceleration = <L/T^2>
    P_incident_displacement_filename = <string>
    unit_of_displacement = <L>
    P_incident_angle = <degrees>
    add_compensation_time = <T>
    source_location = (<T>, <T>, <T>)
    soil_profile_filename = <string>
    soil_surface at <x|y|z> = <L>
    unit_of_vs_and_vp = <L/T>
    unit_of_rho = <M/L^3>
    unit_of_damping = <absolute|percent>
    unit_of_thickness = <L>
    ;
```

Example of adding an inclined plane wave field

```
add wave field # 1 type inclined_plane_wave with
  anticlockwise_angle_of_P_wave_plane_from x = 0
  P_incident_acceleration_filename = "Kobe_acc.txt"
  unit_of_acceleration = 1*m/s^2
  P_incident_displacement_filename = "Kobe_disp.txt"
  unit_of_displacement = 1*m
  P_incident_angle = 15
  add_compensation_time = 0.5*s
  source_location = (-150*m, 0*m, -100*m)
  soil_profile_filename = "soil_profile.txt"
  soil_surface at z = 0*m
  unit_of_vs_and_vp = 1*m/s
  unit_of_rho = 1*kg/m^3
  unit_of_damping = absolute
  unit_of_thickness = 1*m;
```

where:

- `No (or #)<.>` is the unique wave field ID/number. The wave field ID does not have to be sequential, any unique positive integer number can be used. Each wave field is an inclined plane P wave field.
- `anticlockwise_angle_of_P_wave_plane_from <x|y|z>` specifies the orientation of the inclined wave field propagation plane. User should give the anticlockwise angle in degrees between the wave propagation plane and the specified reference axis. The reference axis could be x or y or

z. As shown in figure 205.7, the `anticlockwise_angle_of_P_wave_plane_from` x axis is α.

- `P_incident_acceleration_filename` is the filename of a plain text file, which contains the acceleration of the input motion. The file should have two columns, where the first column is the accumulated time and the second column is the corresponding acceleration. For the DRM loading from a wave field, if the simulation time is longer than the earthquake motion, the remaining simulation will continue with zero motions. The wave field implementation in Real-ESSI does NOT conduct any baseline correction on the input motion. The users should perform baseline correction for earthquake motions by themselves.

- `P_incident_displacement_filename` is the filename of a plain text file, which contains the displacement of the input motion. The file should have two columns, where the first column is the accumulated time and the second column is the corresponding displacement. For the DRM loading from a wave field, if the simulation time is longer than the earthquake motion, the remaining simulation will continue with zero motions. The wave field implementation in Real-ESSI does NOT conduct any baseline correction on the input motion. The users should perform baseline correction for earthquake motions by themselves.

- `P_incident_angle` specifies the inclination angle of incident P wave, measured from vertical axis of wave plane to the wave propagation axis. In figure 205.7, the incident angle of P wave is θ.

- `add_compensation_time` is a feature to add zero-motion in the beginning and at the end of the earthquake motion. If the user does not want to add the additional zeros, user can specify compensation time as 0*s.

- `source_location` specify the location of seismic source, where the seismic motion is input as in `P_incident_acceleration_filename` and `P_incident_displacement_filename`, it is used for determining phase of the wave, and the source location can be inside or outside of the model.

- `soil_profile_filename` is a file name for a file that contains the soil properties for each layer. The soil profile file should have fives columns: (i) shear wave velocity, (ii) compressional wave velocity, (iii) density, (iv) damping ratio and (v) thickness of each layer respectively. The soil layers count from the soil surface to the bedrock. The last layer is bedrock, which has four columns only. User should NOT give the thickness of the last layer, as it extends into half-space.

  One Example of soil profile file is given below.

```
1  // Vs Vp rho damp thickness
2    200 333 2000 0.02 100
3    250 408 2000 0.02 200
```

```
4     2000 3400 2400 0.02
```

- `soil_surface at <x|y|z>` defines the location of soil surface in the global coordinate system of Real-ESSI.

205.3.4.134 Modeling, Wave Field for Creating DRM Loads: DRM Inclined Motion

This command generates inclined DRM motion with pre-defined inclined wave field.

```
generate DRM motion file from wave field # <.> hdf5_file = <string>;
```

One example of generating inclined DRM motion is:

```
generate DRM motion file from wave field # 1 hdf5_file = "DRMinput.hdf5";
```

where

- `wave field # <.>` specifies the inclined plane wave field number which will be used for wave propagation.
- `hdf5_file` specifies the HDF5 file which contains the geometric information about the DRM elements and DRM nodes.

205.3.4.135 Modeling, Imposed Motions: through Loads, Motion Time History, Constant Time Step

Impose motions (displacements, velocities and accelerations) through loads. This one is used if time increment is constant during the analysis. Input files have one column only, corresponding file for displacements, velocities, and accelerations.

```
add load # <.> type imposed motion to node # <.> dof DOFTYPE
time_step = <T>
displacement_scale_unit = <L>
displacement_file = "filename"
velocity_scale_unit = <L/T>
velocity_file = "filename"
acceleration_scale_unit = <L/L^2>
acceleration_file = "filename";
```

The above command generates load to the corresponding node to get the applied imposed motion.

205.3.4.136 Modeling, Imposed Motions: through Loads, Stochastic Motion Time History, Constant Time Step

Impose stochastic motions (uncertain displacements, velocities and accelerations) through stochastic loads. This one is used if time increment is constant during the analysis.

```
add load # <.> type imposed random motions to node # <.> dof DOFTYPE
time_step = <T>
displacement_scale_unit = <L>
displacement_file = "filename"
velocity_scale_unit = <L/T>
velocity_file = "filename"
acceleration_scale_unit = <L/L^2>
acceleration_file = "filename"
penalty_stiffness = <N/L>
using double product # <.>;
```

where

- `DOFTYPE` specify the dof to impose the uncertain motion. It can be either `ux`, `uy` or `uz`.
- `time_step` specify the time step of the imposed uncertain motion.
- `displacement_scale_unit` specify the scale unit of the imposed uncertain displacement polynomial chaos (PC) coefficients.
- `displacement_file` specify the filename of a text file containing PC coefficients of uncertain displacement random process. The number of rows of the file content should be equal to total number of polynomial chaos basis of the displacement random process. The number of columns of the file content should be equal to total number of time steps of the displacement random process. The value at the i^{th} row and j^{th} column of the file gives the PC coefficient of the i^{th} PC basis of the displacement random process at the j^{th} time step.
- `velocity_scale_unit` specify the scale unit of the imposed uncertain velocity polynomial chaos (PC) coefficients.
- `velocity_file` specify the filename of a text file containing PC coefficients of uncertain velocity random process. The number of rows of the file content should be equal to total number of polynomial chaos basis of the velocity random process. The number of columns of the file content should be equal to total number of time steps of the velocity random process. The value at the i^{th} row and j^{th} column of the file gives the PC coefficient of the i^{th} PC basis of the velocity random process at the j^{th} time step.

- `acceleration_scale_unit` specify the scale unit of the imposed uncertain acceleration polynomial chaos (PC) coefficients.

- `acceleration_file` specify the filename of a text file containing PC coefficients of uncertain acceleration random process. The number of rows of the file content should be equal to total number of polynomial chaos basis of the acceleration random process. The number of columns of the file content should be equal to total number of time steps of the acceleration random process. The value at the i^{th} row and j^{th} column of the file gives the PC coefficient of the i^{th} PC basis of the acceleration random process at the j^{th} time step.

- `penalty_stiffness` specify the penalty stiffness for the input of uncertain motion using penalty method. The penalty stiffness is expected to be several magnitudes, e.g., $10^3 \sim 10^6$, larger than the elemental stiffness.

- `double product #` specify the ID of the double product that would be used in the formation of stochastic force. In stochastic finite element method (FEM), the first PC basis for this double product should come from the PC representation of uncertain FEM system response, e.g., uncertain structural displacement. The second PC basis for this double product should come from the uncertain imposed motion representation.

205.3.4.137 Modeling, Imposed Motions: through Loads, Stochastic Random Process Motions, Constant Time Step

Impose a defined random process motions.

This one is used if time increment is constant during the analysis.

```
add load # <.> type imposed random motions to node # <.> dof DOFTYPE
time_step = <T>
uncertain_displacement = random field # <.>
displacement_scale_unit = <L>
penalty_stiffness = <N/L>
using double product # <.>;
```

where

- `DOFTYPE` specify the dof to impose the uncertain motion. It can be either `ux`, `uy` or `uz`.
- `time_step` specify the time step of the imposed uncertain motion.
- `uncertain_displacement` specify the imposed uncertain motion through a defined random field-/process.
- `displacement_scale_unit` specify the scale unit of the imposed uncertain displacement.
- `penalty_stiffness` specify the penalty stiffness for the input of uncertain motion using penalty method. The penalty stiffness is expected to be several magnitudes, e.g., $10^3 \sim 10^6$, larger than the elemental stiffness.
- `double product #` specify the ID of the double product that would be used in the formation of stochastic force. In stochastic finite element method (FEM), the first PC basis for this double product should come from the PC representation of uncertain FEM system response, e.g., uncertain structural displacement. The second PC basis for this double product should come from the uncertain imposed motion representation.

205.3.4.138 Modeling, Imposed Motions: through Loads, Motion Time History, Variable Time Step

Impose motions (displacements, velocities and accelerations) through loads. This one is used if time increment is variable during the analysis. Input files have two columns, first column is time and the second column in corresponding file for displacements, velocities, and accelerations. Time steps have to be the same in each file.

```
add load # <.> type imposed motion to node # <.> dof DOFTYPE
displacement_scale_unit = <displacement>
displacement_file = "filename"
velocity_scale_unit = <velocity>
velocity_file = "filename"
acceleration_scale_unit = <acceleration>
acceleration_file = "filename";
```

The above command generates load to the corresponding node to get the applied imposed motion.

205.3.4.139 Modeling, Imposed Motions: Adding Load for Uniform Acceleration Time History

Defines a non-inertial reference frame from which all displacements are measured. This reference frame (fixed to the base of the model) accelerates according to a given acceleration record. All output quantities are derived from this relative coordinate system (not-inertial). To get total displacements, the twice-integrated acceleration record must be added to the results.

The command is:

```
add load # <.> type uniform acceleration to all nodes dof <.>
  time_step = <T>
  scale_factor = <L/T^2>
  initial_velocity = <L/T>
  acceleration_file = <string>;
```

Where

- `time_step` Is the time step of the record in time units.
- `scale_factor` Is a dimensionless factor with which the record is scaled before it's applied.
- `initial_velocity` Initial velocity for all translational DOFs of the system.
- `acceleration_file` String containing the path (relative or absolute) to the record text file.

File format is a single value of the record in acceleration units (m/s/s) per line for each time step. If a time-step different from the record is used for analysis, then the record is interpolated linearly.

205.3.4.140 Modeling, Imposed Motions: Remove Imposed Motions

Motions can be removed using:

```
1 remove imposed motion # <.>
```

205.3.4.141 Modeling, Random Variable: Adding Gaussian Random Variables

Gaussian random variable can be added for probabilistic analysis.

The command is:

```
add random variable # <.> with Gaussian distribution mean = <.> ↩
    standard_deviation = <.>;
```

where:

- `mean` is the mean of the Gaussian random variable
- `standard_deviation` is the standard deviation of the Gaussian random variable

For example:

```
add random variable # 1 with Gaussian distribution mean = 3.0 ↩
    standard_deviation = 1.0;
```

Adds a Gaussian random variable 1 with mean equal to 3.0 and standard deviation equal to 1.0.

205.3.4.142 Modeling, Random Variable: Adding Gaussian Random Variables with Location

Gaussian random variable with spatial location can be added for probabilistic analysis. The location information of the defined random variable is important to calculate the correlation structure of random field, that can consist of many random variables.

The command is:

```
add random variable # <.> with Gaussian distribution mean = <.> ↩
    standard_deviation = <.> at (<L>, <L>, <L>);
```

where:

- `mean` is the mean of the Gaussian random variable
- `standard_deviation` is the standard deviation of the Gaussian random variable

For example:

```
add random variable # 1 with Gaussian distribution mean = 3.0 ↩
    standard_deviation = 1.0 at (3*m, 0*m, 0*m);
```

Adds a Gaussian random variable 1 with mean equal to 3.0 and standard deviation equal to 1.0 at location $x = 3$m, $y = 0$m and $z = 0$m.

205.3.4.143 Modeling, Random Variable: Adding Lognormal Random Variables

Lognormal random variable can be added for probabilistic analysis.

The command is:

```
add random variable # <.> with Lognormal distribution mean = <.> ↩
    standard_deviation = <.>;
```

where:

- `mean` is the mean of the lognormal random variable
- `standard_deviation` is the standard deviation of the lognormal random variable

For example:

```
add random variable # 1 with Lognormal distribution mean = 3.0 ↩
    standard_deviation = 1.0;
```

Adds a Lognormal random variable 1 with mean equal to 3.0 and standard deviation equal to 1.0.

205.3.4.144 Modeling, Random Variable: Adding Lognormal Random Variables with Location

Lognormal random variable with spatial location can be added for probabilistic analysis. The location information of the defined random variable is important to calculate the correlation structure of random field, that can consist of many random variables.

The command is:

```
add random variable # <.> with Lognormal distribution mean = <.> ↩
    standard_deviation = <.> at (<L>, <L>, <L>);
```

where:

- `mean` is the mean of the lognormal random variable
- `standard_deviation` is the standard deviation of the lognormal random variable

For example:

```
add random variable # 1 with Lognormal distribution mean = 3.0 ↩
    standard_deviation = 1.0 at (3*m, 0*m, 0*m);
```

Adds a Lognormal random variable 1 with mean equal to 3.0 and standard deviation equal to 1.0 at location $x = 3$m, $y = 0$m and $z = 0$m.

205.3.4.145 Modeling, Random Variable: Adding Lognormal Random Variables using Logarithmic Input

Lognormal random variable can be added for probabilistic analysis.

The command is:

```
add random variable # <.> with Lognormal distribution lognormal_mean = <.> ↩
    lognormal_standard_deviation = <.>;
```

where:

- `lognormal_mean`: μ is the mean of the natural logarithm of the lognormal random variable X
- `lognormal_standard_deviation`: σ is the standard deviation of the natural logarithm of the lognormal random variable X

In other words, for lognormal distributed random variable X with parameters μ and σ, we have:

$$ln(X) \sim N(\mu, \sigma) \tag{205.34}$$

It is noted that the mean m and variance v of lognormal random variable X is related to parameters μ and σ as follows:

$$m = e^{\mu+\sigma^2/2} \tag{205.35}$$

$$v = e^{2\mu+\sigma^2}(e^{\sigma^2} - 1) \tag{205.36}$$

For example:

```
add random variable # 1 with Lognormal distribution lognormal_mean = 3.0 ↩
    lognormal_standard_deviation = 1.0;
```

adds a Lognormal random variable 1. The natural logarithm of such random variable follows Gaussian distribution with mean equal to 3.0 and standard deviation equal to 1.0.

205.3.4.146 Modeling, Random Variable: Adding Lognormal Random Variables using Logarithmic Input with Location

Lognormal random variable with spatial location can be added for probabilistic analysis. The location information of the defined random variable is important to calculate the correlation structure of random field, which can consist of many random variables.

The command is:

```
add random variable # <.> with Lognormal distribution lognormal_mean = <.> ↩
    lognormal_standard_deviation = <.> at (<L>, <L>, <L>);
```

where:

- `lognormal_mean`: μ is the mean of the natural logarithm of the lognormal random variable X
- `lognormal_standard_deviation`: σ is the standard deviation of the natural logarithm of the lognormal random variable X

In other words, for lognormal distributed random variable X with parameters μ and σ, we have:

$$ln(X) \sim N(\mu, \sigma) \tag{205.37}$$

It is noted that the mean m and variance v of lognormal random variable X is related to parameters μ and σ as follows:

$$m = e^{\mu+\sigma^2/2} \tag{205.38}$$

$$v = e^{2\mu+\sigma^2}(e^{\sigma^2} - 1) \tag{205.39}$$

For example:

```
add random variable # 1 with Lognormal distribution lognormal_mean = 3.0 ↩
    lognormal_standard_deviation = 1.0 at (3*m, 0*m, 0*m);
```

Adds a Lognormal random variable 1 at location $x = 3$m, $y = 0$m and $z = 0$m. The natural logarithm of such random variable follows Gaussian distribution with mean equal to 3.0 and standard deviation equal to 1.0.

205.3.4.147 Modeling, Random Variable: Adding Gamma Random Variables using Shape and Scale Parameters

Random variable following Gamma distribution can be added for probabilistic analysis.

The command is:

```
add random variable # <.> with Gamma distribution shape_parameter = <.> ↩
    scale_parameter = <.>;
```

where:

- `shape_parameter` is the shape parameter of the Gamma random variable
- `scale_parameter` is the scale parameter of the Gamma random variable

For example:

```
add random variable # 1 with Gamma distribution shape_parameter = 5.0 ↩
    scale_parameter = 2.0;
```

Adds a Gamma random variable 1 with the shape parameter equal to 5.0 and scale parameter equal to 2.0.

205.3.4.148 Modeling, Random Variable: Adding Gamma Random Variables using Shape and Scale Parameters with Location

Random variable following Gamma distribution can be added for probabilistic analysis.

The location information of the defined random variable is important to calculate the correlation structure of random field, that can consist of many random variables.

The command is:

```
add random variable # <.> with Gamma distribution shape_parameter = <.> ↩
    scale_parameter = <.> at (<L>, <L>, <L>);
```

where:

- `shape_parameter` is the shape parameter of the Gamma random variable
- `scale_parameter` is the scale parameter of the Gamma random variable

For example:

```
add random variable # 1 with Gamma distribution shape_parameter = 5.0 ↩
    scale_parameter = 2.0 at (3*m, 0*m, 0*m);
```

Adds a Gamma random variable 1 with the shape parameter equal to 5.0 and scale parameter equal to 2.0 at location $x = 3$m, $y = 0$m and $z = 0$m.

205.3.4.149 Modeling, Random Variable: Adding Gamma Random Variables using Mean and Standard Deviation Parameters

Random variable for given mean and standard deviation parameter following Gamma distribution can be added for probabilistic analysis.

The command is:

```
add random variable # <.> with Gamma distribution mean = <.> standard_deviation = <.>;
```

where:

- `mean` is the mean of the Gamma random variable
- `standard_deviation` is the standard deviation of the Gamma random variable

For example:

```
add random variable # 1 with Gamma distribution mean = 5.0 standard_deviation = 2.0;
```

Adds a Gamma random variable 1 with mean equal to 5.0 and standard deviation equal to 2.0.

205.3.4.150 Modeling, Random Variable: Adding Gamma Random Variables using Mean and Standard Deviation Parameters with Location

Random variable for given mean and standard deviation parameter following Gamma distribution can be added for probabilistic analysis.

The location information of the defined random variable is important to calculate the correlation structure of random field, that can consist of many random variables.

The command is:

```
add random variable # <.> with Gamma distribution mean = <.> standard_deviation ↩
    = <.> at (<L>, <L>, <L>);
```

where:

- `mean` is the mean of the Gamma random variable
- `standard_deviation` is the standard deviation of the Gamma random variable

For example:

```
add random variable # 1 with Gamma distribution mean = 5.0 standard_deviation = ↩
    2.0 at (3*m, 0*m, 0*m);
```

Adds a Gamma random variable 1 with mean equal to 5.0 and standard deviation equal to 2.0 at location $x = 3\text{m}$, $y = 0\text{m}$ and $z = 0\text{m}$.

205.3.4.151 Modeling, Random Variable: Adding Weibull Random Variables using Shape and Scale Parameters

Random variable following Weibull distribution can be added for probabilistic analysis.

The command is:

```
add random variable # <.> with Weibull distribution shape_parameter = <.> ↩
    scale_parameter = <.>;
```

where:

- `shape_parameter` is the shape parameter of the Weibull random variable
- `scale_parameter` is the scale parameter of the Weibull random variable

For example:

```
add random variable # 1 with Weibull distribution shape_parameter = 5.0 ↩
    scale_parameter = 2.0;
```

Adds a Weibull random variable 1 with the shape parameter equal to 5.0 and scale parameter equal to 2.0.

205.3.4.152 Modeling, Random Variable: Adding Weibull Random Variables using Shape and Scale Parameters with Location

Random variable following Weibull distribution can be added for probabilistic analysis.

The location information of the defined random variable is important to calculate the correlation structure of random field, which can consist of many random variables.

The command is:

```
add random variable # <.> with Weibull distribution shape_parameter = <.> ↩
    scale_parameter = <.> at (<L>, <L>, <L>);
```

where:

- `shape_parameter` is the shape parameter of the Weibull random variable
- `scale_parameter` is the scale parameter of the Weibull random variable

For example:

```
add random variable # 1 with Weibull distribution shape_parameter = 5.0 ↩
    scale_parameter = 2.0 at (3*m, 0*m, 0*m);
```

adds a Weibull random variable 1 with the shape parameter equal to 5.0 and scale parameter equal to 2.0 at location $x = 3$m, $y = 0$m and $z = 0$m.

205.3.4.153 Modeling, Random Variable: Remove Random Variables

Remove random variables.

The command is:

```
remove random variable # <.> ;
```

For example:

```
remove random variable # 2;
```

Remove random variable 2 from the analysis.

205.3.4.154 Modeling, Random Variable: Hermite Polynomial Chaos Expansion

Hermite polynomial chaos expansion Xiu (2010) can be performed for random variable with any type of distribution.

The command is:

```
Hermite polynomial chaos expansion to random variable # <.> with order <.>;
```

where:

- `order` specifies the order of Hermite polynomial chaos expansion

For example:

```
Hermite polynomial chaos expansion to random variable # 1 with order 6;
```

Performs Hermite polynomial chaos expansion to random variable 1 using Hermite polynomial chaos up to order 6.

205.3.4.155 Modeling, Random Variable: Output Hermite Polynomial Chaos Expansion Result

A HDF5 (.hdf5) file contains computed polynomial chaos coefficients of Hermite polynomial chaos expansion for random variable can be generated.

The command is:

```
generate Hermite polynomial chaos expansion file from random variable # <.> ↩
    hdf5_file = "file_name";
```

where:

- `file_name` is a string that specifies the name of the output hdf5 file.

The generated hdf5 file contains two datasets:

- Dataset PC is a 2D array that describes the multi-dimensional Hermite polynomial chaos (PC) basis. $\boldsymbol{PC_{ij}}$ denotes the order of polynomial chaos dimension ξ_j that contributes to the i^{th} multi-dimensional Hermite PC basis.
- Dataset PC Coefficients is a column vector. The i^{th} component of PC Coefficients is the polynomial chaos coefficient corresponding to the i^{th} PC base as described by PC.

For example:

```
generate Hermite polynomial chaos expansion file from random variable # 2 ↩
    hdf5_file = "PC_RV1.hdf5";
```

Generate HDF5 file named "PC_RV2.hdf5" that contains the computed polynomial chaos coefficients of Hermite polynomial chaos expansion of random variable 2.

205.3.4.156 Modeling, Random Variable: Hermite Polynomial Chaos Expansion & Output Results

Hermite polynomial chaos expansion Xiu (2010) can be performed for random variable with any type of distribution. A HDF5 (.hdf5) file contains computed polynomial chaos coefficients of Hermite polynomial chaos expansion for random variable can be generated.

The command is:

```
generate Hermite polynomial chaos expansion file from random variable # <.> ↩
    with order <.> hdf5_file = "file_name";
```

where:

- `order` specifies the order of Hermite polynomial chaos expansion
- `file_name` is a string that specifies the name of the output hdf5 file.

The generated hdf5 file contains two datasets:

- Dataset PC is a 2D array that describes the multi-dimensional Hermite polynomial chaos (PC) basis. $\boldsymbol{PC_{ij}}$ denotes the order of polynomial chaos dimension ξ_j that contributes to the i^{th} multi-dimensional Hermite PC basis.
- Dataset PC Coefficients is a column vector. The i^{th} component of PC Coefficients is the polynomial chaos coefficient corresponding to the i^{th} PC base as described by PC.

For example:

```
generate Hermite polynomial chaos expansion file from random variable # 1 with ↩
    order 6 hdf5_file = "PC_RV1.hdf5";
```

Perform Hermite polynomial chaos expansion to random variable 1 using Hermite polynomial chaos up to order 6 and generate HDF5 file named "PC_RV1.hdf5" that contains the computed polynomial chaos coefficients.

205.3.4.157 Modeling, Random Field: Adding Random Field with Dimension and Order

Random field with specific Hermite polynomial chaos dimension and order can be added for probabilistic analysis.

The command is:

```
add random field # <.> with Hermite polynomial chaos dimension <.> order <.>;
```

where:

- `dimension` defines the dimension of Hermite polynomial chaos expansion of the random field
- `order` defines the order of Hermite polynomial chaos expansion of the random field

For example:

```
add random field # 1 with Hermite polynomial chaos dimension 4 order 3;
```

adds random field 1 with Hermite polynomial chaos expansion of dimension 4 and order 3.

205.3.4.158 Modeling, Random Field: Define Global Dimension Index of Random Field

Define the global dimension index of local Hermite polynomial chaos (PC) dimension for the uncertainty characterization of the random field.

If the global dimension index is not specified, by default Real-ESSI takes the ID of local Hermite polynomial chaos (PC) dimension as the global dimension index.

Please note that correctly specifying global dimension index for local Hermite PC dimensions of the random field is very important, especially when there are multiple random fields exist in the system and need to compute the triple products of Hermite PC basis of these random fields.

The command is:

```
define random field # <> Hermite polynomial chaos dimension # <> as global ↩
    dimension # <>;
```

where:

- `Hermite polynomial chaos dimension` defines the local dimension ID for Hermite polynomial chaos (PC) basis of the random field. It should be an integer no more than the total number of dimensions adopted in the Hermite polynomial chaos (PC) Karhunen Loève expansion of the random field.
- `global dimension` defines the corresponding global dimension index for the local Hermite PC dimension

For example:

```
define random field # 1 Hermite polynomial chaos dimension # 1 as global ↩
    dimension # 10;
```

defines the global dimension index of local Hermite PC dimension 1 of random field 1 is 10.

205.3.4.159 Modeling, Random Field: Define Global Dimension Index of Random Field from File Input

Define the global dimension index of local Hermite polynomial chaos (PC) dimension for the uncertainty characterization of the random field using inputs from a text file.

If the global dimension index is not specified, by default Real-ESSI takes the ID of local Hermite polynomial chaos (PC) dimension as the global dimension index.

Please note that correctly specifying global dimension index for local Hermite PC dimensions of the random field is very important, especially when there are multiple random fields exist in the system and need to compute the triple products of Hermite PC basis of these random fields.

The command is:

```
define random field # <.> Hermite polynomial chaos dimension from ↩
    dimension_file = "file_name";
```

where:

- `dimension_file` specifies the name of a text file that contains two columns: The first column is local dimension ID of Hermite PC basis; The second column is corresponding global dimension ID for the local dimension of Hermite PC basis. Comments lines starts with "//"

For example:

```
define random field # 1 Hermite polynomial chaos dimension from dimension_file ↩
    = "dimension_info_RF1.txt";
```

defines the global dimension index of local Hermite PC dimensions for random field 1 with input from a text file "dimension_info_RF1.txt".

An example file of "dimension_info_RF1.txt" is provided below:

```
//================================================================
// This file specify the global dimension index of local dimensions
// of Hermite PC basis
// File should have two columns separated by spaces:
// The first column is local KL dimension ID
// The second column is the global KL dimension ID
//================================================================
1 10
2 11
3 12
4 13
```

205.3.4.160 Modeling, Random Field: Set Number of Polynomial Chaos Terms of Random Field

Specify the number of polynomial chaos terms of a random field involved in the stochastic finite element analysis. By default, the full polynomial chaos basis of a random field would be used for uncertainty propagation. The specified number of polynomial chaos terms in this command is used for truncation of polynomial chaos basis.

The command is:

```
set random field # <.> polynomial_chaos_terms = <.>;
```

where:

- `polynomial_chaos_terms` defines the number of truncated Hermite polynomial chaos basis of the random field

For example:

```
set random field # 1 polynomial_chaos_terms = 100;
```

Set the number of truncated Hermite polynomial chaos basis of random field 1 to be 100.

205.3.4.161 Modeling, Random Field: Adding Random Field with Zero Correlation

Random field with uncorrelated random variables can be added for probabilistic analysis.

The command is:

```
add random field # <.> with zero correlation;
```

205.3.4.162 Modeling, Random Field: Adding Random Field with Exponential Correlation

Random field with exponential correlation can be added for probabilistic analysis.

The command is:

```
add random field # <.> with exponential correlation correlation_length = <L> ;
```

where:

- `correlation_length` l_c defines the correlation length of random field such that the correlation $\rho(RV_i, RV_j)$ of any two random variables RV_i and RV_j is given as:

$$\rho(RV_i, RV_j) = exp(-d/l_c) \tag{205.40}$$

Variable d is the Euclidean distance between RV_i and RV_j, that is calculated from the spatial locations of random variables within the random field.

For example:

```
add random field # 1 with exponential correlation correlation_length = 10*m;
```

adds an exponentially correlated random field number 1 with correlation length 10m.

205.3.4.163 Modeling, Random Field: Adding Random Field with Triangular Correlation

Random field with triangular correlation can be added for probabilistic analysis.

The command is:

```
add random field # <.> with triangular correlation correlation_length = <L> ;
```

where:

- `correlation_length` l_c defines the correlation length of random field such that the correlation $\rho(RV_i, RV_j)$ of any two random variables RV_i and RV_j is given as:

$$\rho(RV_i, RV_j) = \max\{1 - \mathrm{d}/\mathrm{l_c},\, 0\} \tag{205.41}$$

Variable d is the Euclidean distance between RV_i and RV_j, that is calculated from the spatial locations of random variables within the random field.

For example:

```
add random field # 1 with triangular correlation correlation_length = 10*m;
```

Adds an triangular correlated random field number 1 with correlation length $10\mathrm{m}$.

205.3.4.164 Modeling, Random Field: Adding Random Field with Exponentially Damped Cosine Correlation

Random field with exponentially damped cosine correlation can be added for probabilistic analysis.

The command is:

```
add random field # <.> with exponentially damped cosine correlation ↩
    correlation_length = <L> ;
```

where:

- `correlation_length` l_c defines the correlation length of random field such that the correlation $\rho(RV_i, RV_j)$ of any two random variables RV_i and RV_j is given as:

$$\rho(RV_i, RV_j) = exp(-d/l_c) * cos(d/l_c) \tag{205.42}$$

Variable d is the Euclidean distance between RV_i and RV_j, that is calculated from the spatial locations of random variables within the random field.

For example:

```
add random field # 1 with exponentially damped cosine correlation ↩
    correlation_length = 10*m;
```

adds an exponentially damped cosine correlated random field number 1 with correlation length 10m.

205.3.4.165 Modeling, Random Field: Adding Random Field with Gaussian Correlation

Random field with Gaussian correlation can be added for probabilistic analysis.

The command is:

```
add random field # <.> with Gaussian correlation correlation_length = <L> ;
```

where:

- `correlation_length` l_c defines the correlation length of random field such that the correlation $\rho(RV_i, RV_j)$ of any two random variables RV_i and RV_j is given as:

$$\rho(RV_i, RV_j) = exp(-d^2/l_c^2) \tag{205.43}$$

Variable d is the Euclidean distance between RV_i and RV_j, that is calculated from the spatial locations of random variables within the random field.

For example:

```
add random field # 1 with Gaussian correlation correlation_length = 10*m;
```

adds a random field number 1 with Gaussian correlation and correlation length 10m.

205.3.4.166 Modeling, Random Field: Remove Random Fields

Remove random Fields.

The command is:

```
remove random field # <.> ;
```

For example:

```
remove random field # 2;
```

Remove random field 2 from the analysis.

205.3.4.167 Modeling, Random Field: Adding Random Variable to Random Field

Add random variable to random field.

The command is:

```
add random variable # <.> to random field # <.>;
```

For example:

```
add random variable # 2 to random field # 1;
```

Adds random variable 2 to random field 1.

205.3.4.168 Modeling, Random Field: Remove Random Variable From Random Field

Remove random variable from random field.

The command is:

```
remove random variable # <.> from random field # <.>;
```

For example:

```
remove random variable # 2 from random field # 1;
```

Remove random variable 2 from random field 1.

205.3.4.169 Modeling, Random Field: Hermite Polynomial Chaos Karhunen Loève Expansion

Perform Hermite polynomial chaos Karhunen Loève expansion for random field of any arbitrary marginal distribution and correlation structure according to Sakamoto and Ghanem (2002).

The command is:

```
Hermite polynomial chaos Karhunen Loeve expansion to random field # <.> with ↩
    Hermite polynomial chaos dimension <.> order <.>;
```

Where:

- `dimension`: specifies the number of dimensions of Hermite polynomial chaos to capture the correlation structure of the random field
- `order`: specifies the order of Hermite polynomial chaos to capture the marginal distribution of the random field

For example:

```
Hermite polynomial chaos Karhunen Loeve expansion to random field # 1 with ↩
    Hermite polynomial chaos dimension 4 order 3;
```

Perform the Hermite polynomial chaos Karhunen Loève expansion for random field 1 using Hermite polynomial chaos of dimension 4 and order 3.

205.3.4.170 Modeling, Random Field: Hermite Polynomial Chaos Karhunen Loève Expansion with Inverse Order

Perform Hermite polynomial chaos Karhunen Loève expansion for random field of any arbitrary marginal distribution and correlation structure according to Sakamoto and Ghanem (2002).

The user can explicitly state the order used in the inversion of underlying Gaussian correlation kernel. The command is:

```
Hermite polynomial chaos Karhunen Loeve expansion to random field # <.> with ↩
    Hermite polynomial chaos dimension <.> order <.> ↩
    correlation_kernel_inverse_order = <.>;
```

Where:

- `dimension`: specifies the number of dimensions of Hermite polynomial chaos to capture the correlation structure of the random field
- `order`: specifies the order of Hermite polynomial chaos to capture the marginal distribution of the random field
- `correlation_kernel_inverse_order`: specifies the order used in the inversion of underlying Gaussian correlation kernel. For the exact Gaussian kernel inversion, set up `correlation_kernel_inverse_order` equal to `order` of Hermite polynomial chaos. `correlation_kernel_inverse_order` should not exceed `order` of Hermite polynomial chaos. If `correlation_kernel_inverse_order` is not stated, by default linear Gaussian kernel inversion, i.e., `correlation_kernel_inverse_order` equal to 1, is performed as the approximation of higher order inversion. See Sakamoto and Ghanem (2002) for more details.

For example:

```
Hermite polynomial chaos Karhunen Loeve expansion to random field # 1 with ↩
    Hermite polynomial chaos dimension 4 order 2 ↩
    correlation_kernel_inverse_order = 2;
```

Perform the Hermite polynomial chaos Karhunen Loève expansion for random field 1 using Hermite polynomial chaos of dimension 4 and order 2. The 2^{nd} order Gaussian correlation kernel inversion is adopted.

205.3.4.171 Modeling, Random Field: Hermite Polynomial Chaos Karhunen Loève Expansion with Number of FE Elements Larger than Dimension of Hermite Polynomials

Perform Hermite polynomial chaos Karhunen Loève expansion for random field described by either Gaussian or lognormal distribution and Gaussian auto-correlation coefficient function. The PCE coefficients are computed using Eq. (11) from Sakamoto and Ghanem (2002).

This command should be used when the number of Gauss points (GPs), i.e., integration points, is much larger than the number of discrete locations needed to solve the eigenproblem of auto-covariance, that is, than the dimension of Hermite polynomials. This allows to save some computation time. In such case, the eigenproblem of auto-covariance function will be solved instead of the eigenproblem of auto-covariance matrix.

Here, "shear beam" element is used. It has only one GP, in the middle of an element. Hence the number of GPs is here equal to the number of FE elements. It is easier for a user to input the number of FE elements than to input the number of GPs.

The command is:

```
Hermite polynomial chaos Karhunen Loeve expansion to random field # <.> with ↩
    Hermite polynomial chaos dimension <.> order <.> ↩
    correlation_kernel_inverse_order = 1 number_of_FE_elements = <.>;
```

Where:

- `dimension`: specifies the number of dimensions of Hermite polynomial chaos to capture the correlation structure of the random field, in other words, it is the number of discrete points used in the solution of the eigenproblem of auto-covariance
- `order`: specifies the order of Hermite polynomial chaos to capture the marginal distribution of the random field
- `correlation_kernel_inverse_order = 1`: does not specify anything and must be equal to 1
- `number_of_FE_elements`: here, "shear beam" element is used and it has only one GP, in the middle of an element, hence the number of FE elements is here equal to the number of GPs, the user must provide (via command `add random variable...`) the type of distribution (here, either Gaussian or lognormal), mean and standard deviation in the middle of each FE element (mean and standard deviation at discrete points used in the solution of the eigenproblem of auto-covariance function are interpolated using the values at GPs), they should be sorted for increasing coordinates in 1D

  `number_of_FE_elements = dimension` is equivalent to:

```
Hermite polynomial chaos Karhunen Loeve expansion to random field # <.> ↩
    with Hermite polynomial chaos dimension <.> order <.> ;
```

and:

```
Hermite polynomial chaos Karhunen Loeve expansion to random field # <.> ↩
    with Hermite polynomial chaos dimension <.> order <.> ↩
    correlation_kernel_inverse_order = 1;
```

For example:

```
Hermite polynomial chaos Karhunen Loeve expansion to random field # 1 with ↩
    Hermite polynomial chaos dimension 4 order 2 ↩
    correlation_kernel_inverse_order = 1 number_of_FE_elements = 20;
```

Perform the Hermite polynomial chaos Karhunen Loève expansion for random field 1 using Hermite polynomial chaos of dimension 4 and order 2 with number of FE elements 20 (i.e., with 20 GPs, in the middle of elements).

Command:

```
Hermite polynomial chaos Karhunen Loeve expansion to random field # <.> with ↩
    Hermite polynomial chaos dimension <.> order <.> number_of_FE_elements = <.>;
```

with optional argument `number_of_FE_elements` cannot be defined because

```
Hermite polynomial chaos Karhunen Loeve expansion to random field # <.> with ↩
    Hermite polynomial chaos dimension <.> order <.> ↩
    correlation_kernel_inverse_order = <.>;
```

with optional argument `correlation_kernel_inverse_order` exists already.

205.3.4.172 Modeling, Random Field: Hermite Polynomial Chaos Karhunen Loève Expansion Using HDF5 Input

Add a random field with marginal distribution and correlation information defined in a given HDF5 (.hdf5) file. Perform Hermite polynomial chaos Karhunen Loève expansion for the random field.

The command is:

```
Hermite polynomial chaos Karhunen Loeve expansion to random field # <.> with
    Hermite polynomial chaos dimension <.> order <.> hdf5_file = "file_name";
```

Where:

- `dimension`: specifies the number of dimensions of Hermite polynomial chaos to capture the correlation structure of the random field
- `order`: specifies the order of Hermite polynomial chaos to capture the marginal distribution of the random field
- `hdf5_file`: specifies the filename of the input HDF5 file that defines the marginal distribution and correlation information of the random field

The input HDF5 file should contain the following datasets:

- Dataset Random Field contains a single integer, which is the ID of the random field.
- Dataset Marginal Mean is a column vector specifying marginal mean of the random field corresponding to each random variable.
- Dataset Marginal Variance is a column vector specifying marginal variance of the random field for each random variable.
- Dataset Marginal Distributions is a column vector integers specifying the marginal distribution IDs of the random field for each random variable. Specifically, the ID is 1 for Gaussian distribution, 2 for Lognormal distribution, 3 for Gamma distribution and 4 for Weibull distribution.
- Dataset Correlation is a 2D array specifying correlation of the random field among random variables.
- Dataset PC Order contains a single integer, which specifies the order of Hermite polynomial chaos (PC) Karhunen Loève expansion.
- Dataset PC Dimension contains a single integer, which specifies the dimension of Hermite polynomial chaos (PC) Karhunen Loève expansion.

- Dataset Index to Global Dimension contains a column vector of integers, which specifies the index of corresponding global PC dimension for each local PC dimension used in the uncertainty expansion of the random field.

For example:

```
Hermite polynomial chaos Karhunen Loeve expansion to random field # 1 with ↩
   Hermite polynomial chaos dimension 4 order 4 hdf5_file = "PC_RF1.hdf5";
```

Perform the Hermite polynomial chaos Karhunen Loève expansion for random field 1 using Hermite polynomial chaos of dimension 4 and order 4 with input marginal distribution and correlation information defined in HDF5 file "PC_RF1.hdf5".

205.3.4.173 Modeling, Random Field: Hermite Polynomial Chaos Karhunen Loève Expansion with Inverse Order Using HDF5 Input

Add a random field with marginal distribution and correlation information defined in a given HDF5 (.hdf5) file. Perform Hermite polynomial chaos Karhunen Loève expansion for the random field.

The user can explicitly state the order used in the inversion of underlying Gaussian correlation kernel.

The command is:

```
Hermite polynomial chaos Karhunen Loeve expansion to random field # <.> with ↩
    Hermite polynomial chaos dimension <.> order <.> ↩
    correlation_kernel_inverse_order = <.> hdf5_file = "file_name";
```

Where:

- `dimension`: specifies the number of dimensions of Hermite polynomial chaos to capture the correlation structure of the random field
- `order`: specifies the order of Hermite polynomial chaos to capture the marginal distribution of the random field
- `correlation_kernel_inverse_order`: specifies the order used in the inversion of underlying Gaussian correlation kernel. For the exact Gaussian kernel inversion, set up `correlation_kernel_inverse_order` equal to `order` of Hermite polynomial chaos. `correlation_kernel_inverse_order` should not exceed `order` of Hermite polynomial chaos. If `correlation_kernel_inverse_order` is not stated, by default linear Gaussian kernel inversion, i.e., `correlation_kernel_inverse_order` equal to 1, is performed as the approximation of higher order inversion. See Sakamoto and Ghanem (2002) for more details.
- `hdf5_file`: specifies the filename of the input HDF5 file that defines the marginal distribution and correlation information of the random field

The input HDF5 file should contain the following datasets:

- Dataset Random Field contains a single integer, which is the ID of the random field.
- Dataset Marginal Mean is a column vector specifying marginal mean of the random field corresponding to each random variable.
- Dataset Marginal Variance is a column vector specifying marginal variance of the random field for each random variable.

- Dataset Marginal Distributions is a column vector integers specifying the marginal distribution IDs of the random field for each random variable. Specifically, the ID is 1 for Gaussian distribution, 2 for Lognormal distribution, 3 for Gamma distribution and 4 for Weibull distribution.
- Dataset Correlation is a 2D array specifying correlation of the random field among random variables.
- Dataset PC Order contains a single integer, which specifies the order of Hermite polynomial chaos (PC) Karhunen Loève expansion.
- Dataset PC Dimension contains a single integer, which specifies the dimension of Hermite polynomial chaos (PC) Karhunen Loève expansion.
- Dataset Index to Global Dimension contains a column vector of integers, which specifies the index of corresponding global PC dimension for each local PC dimension used in the uncertainty expansion of the random field.

For example:

```
1 Hermite polynomial chaos Karhunen Loeve expansion to random field # 1 with ↩
    Hermite polynomial chaos dimension 4 order 4 ↩
    correlation_kernel_inverse_order = 3 hdf5_file = "PC_RF1.hdf5";
```

Perform the Hermite polynomial chaos Karhunen Loève expansion for random field 1 using Hermite polynomial chaos of dimension 4 and order 4. The 3^{rd} order Gaussian correlation kernel inversion is adopted. The input marginal distribution and correlation information are defined in HDF5 file "PC_RF1.hdf5".

205.3.4.174 Modeling, Random Field: Output Hermite Polynomial Chaos Karhunen Loève Expansion Result

A HDF5 (.hdf5) file contains all the information for Hermite polynomial chaos Karhunen Loève expansion for random field can be generated.

The command is:

```
generate Hermite polynomial chaos Karhunen Loeve expansion file from random field # <.> hdf5_file = "file_name";
```

where:

- `file_name` is a string that specifies the name of the output hdf5 file.

The generated hdf5 file contains the following datasets:

- Dataset Random Field contains a single integer, which is the ID of the random field.
- Dataset Random Variables contains a column vector of integers, which are the IDs of the random variables that constitute the random field.
- Dataset Marginal Mean is a column vector specifying marginal mean of the random field corresponding to each random variable.
- Dataset Marginal Variance is a column vector specifying marginal variance of the random field for each random variable.
- Dataset Marginal Distributions is a column vector integers specifying the marginal distribution IDs of the random field for each random variable. Specifically, the ID is 1 for Gaussian distribution, 2 for Lognormal distribution, 3 for Gamma distribution and 4 for Weibull distribution.
- Dataset Correlation is a 2D array specifying correlation of the random field among random variables.
- Dataset PC Order contains a single integer, which specifies the order of Hermite polynomial chaos (PC) Karhunen Loève expansion.
- Dataset PC Dimension contains a single integer, which specifies the dimension of Hermite polynomial chaos (PC) Karhunen Loève expansion.
- Dataset Index to Global Dimension contains a column vector of integers, which specifies the index of corresponding global PC dimension for each local PC dimension used in the uncertainty expansion of the random field.

- Dataset PC is a 2D array that describes the multi-dimensional Hermite polynomial chaos (PC) basis. $\boldsymbol{PC_{ij}}$ denotes the order of polynomial chaos dimension ξ_j that contributes to the i^{th} multi-dimensional Hermite PC basis.
- Dataset PC Coefficients is a 2D array. The j^{th} component of the i^{th} row is the polynomial chaos coefficient corresponding to the j^{th} PC base as described by PC for the i^{th} random variable specified in Random Variables.
- Dataset PC Variance is a column vector specifying the variances of Hermite PC basis.

For example:

```
generate Hermite polynomial chaos Karhunen Loeve expansion file from random ↩
    field # 1 hdf5_file = "PC_RF1.hdf5";
```

Generate HDF5 file named "PC_RF1.hdf5" that contains all the information for Hermite polynomial chaos Karhunen Loève expansion of random field 1.

205.3.4.175 Modeling, Random Field: Adding Random Field from Hermite Polynomial Chaos Karhunen Loève Expansion HDF5 File

Add a random field with marginal distribution, correlation information and multi-dimensional Hermite polynomial chaos (PC) coefficients specified in a given HDF5 (.hdf5) file.

The command is:

```
add random field # <.> with Hermite polynomial chaos Karhunen Loeve expansion ↩
    hdf5_file = "file_name";
```

where:

- `file_name` is a string that specifies the name of the input hdf5 file.

The input hdf5 file should contain the following datasets:

- Dataset Random Field contains a single integer, that represents the ID of the random field.
- Dataset Random Variables contains a column vector of integers, that are the IDs of the random variables that constitute the random field.
- Dataset Marginal Mean is a column vector specifying marginal mean of the random field corresponding to each random variable.
- Dataset Marginal Variance is a column vector specifying marginal variance of the random field for each random variable.
- Dataset Marginal Distributions is a column vector of integers specifying the marginal distribution IDs of the random field for each random variable. Specifically, the ID is 1 for Gaussian distribution, 2 for Lognormal distribution, 3 for Gamma distribution and 4 for Weibull distribution.
- Dataset Correlation is a 2D array specifying correlation of the random field among random variables.
- Dataset PC Order contains a single integer, that specifies the order of Hermite polynomial chaos (PC) Karhunen Loève expansion.
- Dataset PC Dimension contains a single integer, that specifies the dimension of Hermite polynomial chaos (PC) Karhunen Loève expansion.
- Dataset Index to Global Dimension contains a column vector of integers, that specify the index of corresponding global PC dimension for each local PC dimension used in the uncertainty expansion of the random field.

- Dataset PC is a 2D array that describes the multi-dimensional Hermite polynomial chaos (PC) basis. $\boldsymbol{PC_{ij}}$ denotes the order of polynomial chaos dimension ξ_j that contributes to the i^{th} multi-dimensional Hermite PC basis.
- Dataset PC Coefficients is a 2D array. The j^{th} component of the i^{th} row is the polynomial chaos coefficient corresponding to the j^{th} PC base as described by PC for the i^{th} random variable specified in Random Variables.
- Dataset PC Variance is a column vector specifying the variances of Hermite PC basis.

For example:

```
add random field # 1 with Hermite polynomial chaos Karhunen Loeve expansion ↩
   hdf5_file = "PC_RF1.hdf5";
```

Add random field 1 with the marginal distribution, correlation structure and Hermite polynomial chaos Karhunen Loève expansion information defined in HDF5 file named "PC_RF1.hdf5".

205.3.4.176 Modeling, Random Field: Adding Random Field from Marginal Distribution and Correlation

Add a random field with specified marginal distribution and correlation information.

The command is:

```
add random field # <.> with <distribution_type> distribution
marginal_mean_file = "file_name"
marginal_standard_deviation_file = "file_name"
correlation_file = "file_name";
```

where:

- `distribution_type` is a string specifying the marginal distribution type of the random field, can be `Gaussian`, `Lognormal`, or `Gamma`.
- `marginal_mean_file` is a string specifying the name of a plain text file that contains a single column of marginal mean of the random field.
- `marginal_standard_deviation_file` is a string specifying the name of a plain text file that contains a single column of marginal standard deviation of the random field.
- `correlation_file` is a string specifying the name of a plain text file that contains a 2D array of the correlation structure of the random field.

205.3.4.177 Modeling, Random Field: Add Triple Product of Hermite Polynomial Chaos Basis

Compute and add triple product of Hermite polynomial chaos basis from three different random fields.

The command is:

```
add triple product # <.> with Hermite polynomial chaos from random field (<.>, <.>, <.>);
```

For example:

```
add triple product # 1 with Hermite polynomial chaos from random field (1, 2, 3);
```

Compute and add triple product #1 with Hermite polynomial chaos basis from random field 1, 2 and 3;

205.3.4.178 Modeling, Random Field: Add Double Product of Hermite Polynomial Chaos Basis

Compute and add double product of Hermite polynomial chaos basis from two different random fields.

The command is:

```
add double product # <.> with Hermite polynomial chaos from random field (<.>, <.>);
```

For example:

```
add double product # 1 with Hermite polynomial chaos from random field (2, 3);
```

Compute and add double product #1 with Hermite polynomial chaos basis from random field 2 and 3;

205.3.4.179 Modeling, Random Field: Generate Triple Product of Hermite Polynomial Chaos Basis

The computation of triple product of Hermite polynomial chaos basis for different random fields is a key part for stochastic finite element analysis.

A HDF5 (.hdf5) file contains triple product of Hermite polynomial chaos basis for random fields can be generated.

The command is:

```
generate triple product of Hermite polynomial chaos from random field (<.>, <.>, <.>) hdf5_file = "file_name";
```

where:

- `file_name` is a string that specifies the name of the output HDF5 file.

The generated HDF5 file contains the following datasets:

- Dataset PC1 is a 2D array that describes the multi-dimensional Hermite polynomial chaos (PC) basis of the first random field. $\boldsymbol{PC1_{ij}}$ denotes the order of polynomial chaos dimension ξ_j that contributes to the i^{th} basis of PC1.
- Dataset PC1 Index to Global Dimension contains a column vector of integers, which specifies the global dimension index of each local PC dimension in basis PC1.
- Dataset PC1 Variance is a column vector specifying the variances of basis PC1.
- Dataset PC2 is a 2D array that describes the multi-dimensional Hermite polynomial chaos (PC) basis of the first random field. $\boldsymbol{PC2_{ij}}$ denotes the order of polynomial chaos dimension ξ_j that contributes to the i^{th} basis of PC2.
- Dataset PC2 Index to Global Dimension contains a column vector of integers, which specifies the global dimension index of each local PC dimension in basis PC2.
- Dataset PC2 Variance is a column vector specifying the variances of basis PC2.
- Dataset PC3 is a 2D array that describes the multi-dimensional Hermite polynomial chaos (PC) basis of the first random field. $\boldsymbol{PC3_{ij}}$ denotes the order of polynomial chaos dimension ξ_j that contributes to the i^{th} basis of PC3.
- Dataset PC3 Index to Global Dimension contains a column vector of integers, which specifies the global dimension index of each local PC dimension in basis PC3.

- Dataset PC3 Variance is a column vector specifying the variances of basis PC3.
- Dataset Triple Product is a column vector containing the non-zero triple products of polynomial chaos basis from PC1, PC2 and PC3.
- Dataset Triple Product PC1 Index is a column vector containing the indexes of polynomial chaos basis from PC1 that contributes to the non-zero triple products in dataset Triple Product.
- Dataset Triple Product PC2 Index is a column vector containing the indexes of polynomial chaos basis from PC2 that contributes to the non-zero triple products in dataset Triple Product.
- Dataset Triple Product PC3 Index is a column vector containing the indexes of polynomial chaos basis from PC3 that contributes to the non-zero triple products in dataset Triple Product.

For example:

```
generate triple product of Hermite polynomial chaos from random field (1, 2, 3) ↩
    hdf5_file = "Triple_product_4(3)_4(3)_4(3).hdf5";
```

Compute the triple product of Hermite polynomial chaos basis of random field 1, 2 and 3 and write all the results into a HDF5 file named "Triple_product_4(3)_4(3)_4(3).hdf5";

205.3.4.180 Modeling, Random Field: Generate Double Product of Hermite Polynomial Chaos Basis

The computation of double product of Hermite polynoial chaos basis for different random fields is a key part for stochastic finite element analysis.

A HDF5 (.hdf5) file contains double product of Hermite polynomial chaos basis for random fields can be generated.

The command is:

```
generate double product of Hermite polynomial chaos from random field (<.>, <.>) hdf5_file = "file_name";
```

where:

- `file_name` is a string that specifies the name of the output HDF5 file.

The generated HDF5 file contains the following datasets:

- Dataset PC1 is a 2D array that describes the multi-dimensional Hermite polynomial chaos (PC) basis of the first random field. $\boldsymbol{PC1_{ij}}$ denotes the order of polynomial chaos dimension ξ_j that contributes to the i^{th} basis of PC1.
- Dataset PC1 Index to Global Dimension contains a column vector of integers, which specifies the global dimension index of each local PC dimension in basis PC1.
- Dataset PC1 Variance is a column vector specifying the variances of basis PC1.
- Dataset PC2 is a 2D array that describes the multi-dimensional Hermite polynomial chaos (PC) basis of the first random field. $\boldsymbol{PC2_{ij}}$ denotes the order of polynomial chaos dimension ξ_j that contributes to the i^{th} basis of PC2.
- Dataset PC2 Index to Global Dimension contains a column vector of integers, which specifies the global dimension index of each local PC dimension in basis PC2.
- Dataset PC2 Variance is a column vector specifying the variances of basis PC2.
- Dataset Double Product is a column vector containing the non-zero double products of polynomial chaos basis from PC1 and PC2.
- Dataset Double Product PC1 Index is a column vector containing the indexes of polynomial chaos basis from PC1 that contributes to the non-zero double products in dataset Double Product.

- Dataset Double Product PC2 Index is a column vector containing the indexes of polynomial chaos basis from PC2 that contributes to the non-zero double products in dataset Double Product.

For example:

```
1 generate double product of Hermite polynomial chaos from random field (1, 2) ↩
    hdf5_file = "doubleproduct_153(2)_150(1).hdf5";
```

Compute the triple product of Hermite polynomial chaos basis of random field 1 and 2 and write all the results into a HDF5 file named "doubleproduct_153(2)_150(1).hdf5";

205.3.4.181 Modeling, Random Field: Add Triple Product of Hermite Polynomial Chaos Basis Using HDF5 Input

Add triple product of Hermite polynomial chaos basis using HDF5 input.

The command is:

```
add triple product # <.> from hdf5_file = "file_name";
```

where:

- `file_name` is a string that specifies the name of the input HDF5 file.

The input HDF5 file should contain the following datasets:

- Dataset PC1 is a 2D array that describes the multi-dimensional Hermite polynomial chaos (PC) basis of the first random field. $\boldsymbol{PC1_{ij}}$ denotes the order of polynomial chaos dimension ξ_j that contributes to the i^{th} basis of PC1.
- Dataset PC1 Index to Global Dimension contains a column vector of integers, which specifies the global dimension index of each local PC dimension in basis PC1.
- Dataset PC1 Variance is a column vector specifying the variances of basis PC1.
- Dataset PC2 is a 2D array that describes the multi-dimensional Hermite polynomial chaos (PC) basis of the first random field. $\boldsymbol{PC2_{ij}}$ denotes the order of polynomial chaos dimension ξ_j that contributes to the i^{th} basis of PC2.
- Dataset PC2 Index to Global Dimension contains a column vector of integers, which specifies the global dimension index of each local PC dimension in basis PC2.
- Dataset PC2 Variance is a column vector specifying the variances of basis PC2.
- Dataset PC3 is a 2D array that describes the multi-dimensional Hermite polynomial chaos (PC) basis of the first random field. $\boldsymbol{PC3_{ij}}$ denotes the order of polynomial chaos dimension ξ_j that contributes to the i^{th} basis of PC3.
- Dataset PC3 Index to Global Dimension contains a column vector of integers, which specifies the global dimension index of each local PC dimension in basis PC3.
- Dataset PC3 Variance is a column vector specifying the variances of basis PC3.
- Dataset Triple Product is a column vector containing the non-zero triple products of polynomial chaos basis from PC1, PC2 and PC3.

- Dataset Triple Product PC1 Index is a column vector containing the indexes of polynomial chaos basis from PC1 that contributes to the non-zero triple products in dataset Triple Product.
- Dataset Triple Product PC2 Index is a column vector containing the indexes of polynomial chaos basis from PC2 that contributes to the non-zero triple products in dataset Triple Product.
- Dataset Triple Product PC3 Index is a column vector containing the indexes of polynomial chaos basis from PC3 that contributes to the non-zero triple products in dataset Triple Product.

For example:

```
1 add triple product # 1 from hdf5_file = "tripleproduct_3(2)_153(2)_153(2).hdf5";
```

Add triple product #1 using HDF5 input file named "tripleproduct_3(2)_153(2)_153(2).hdf5".

205.3.4.182 Modeling, Random Field: Add Double Product of Hermite Polynomial Chaos Basis Using HDF5 Input

Add double product of Hermite polynomial chaos basis using HDF5 input.

The command is:

```
add double product # <.> from hdf5_file = "file_name";
```

where:

- `file_name` is a string that specifies the name of the input HDF5 file.

The input HDF5 file should contain the following datasets:

- Dataset PC1 is a 2D array that describes the multi-dimensional Hermite polynomial chaos (PC) basis of the first random field. $\boldsymbol{PC1_{ij}}$ denotes the order of polynomial chaos dimension ξ_j that contributes to the i^{th} basis of PC1.
- Dataset PC1 Index to Global Dimension contains a column vector of integers, which specifies the global dimension index of each local PC dimension in basis PC1.
- Dataset PC1 Variance is a column vector specifying the variances of basis PC1.
- Dataset PC2 is a 2D array that describes the multi-dimensional Hermite polynomial chaos (PC) basis of the first random field. $\boldsymbol{PC2_{ij}}$ denotes the order of polynomial chaos dimension ξ_j that contributes to the i^{th} basis of PC2.
- Dataset PC2 Index to Global Dimension contains a column vector of integers, which specifies the global dimension index of each local PC dimension in basis PC2.
- Dataset PC2 Variance is a column vector specifying the variances of basis PC2.
- Dataset Double Product is a column vector containing the non-zero double products of polynomial chaos basis from PC1 and PC2.
- Dataset Double Product PC1 Index is a column vector containing the indexes of polynomial chaos basis from PC1 that contributes to the non-zero double products in dataset Double Product.
- Dataset Double Product PC2 Index is a column vector containing the indexes of polynomial chaos basis from PC2 that contributes to the non-zero double products in dataset Double Product.

For example:

```
add double product # 1 from hdf5_file = "doubleproduct_153(2)_150(1).hdf5";
```

Add double product #1 using HDF5 input file named "doubleproduct_153(2)_150(1).hdf5".

205.3.4.183 Modeling, Solid-Fluid Interaction: Adding Solid-Fluid Interface

For solid-fluid interaction analysis, solid fluid interface should be defined and added to the analysis domain.

The command is:

```
add solid fluid interface <string>
```

where:

- <string> specifies the name of the boundary of fluid domain that is solid fluid interface. It is noted that the boundary name should be consistent with the definition in the OpenFOAM input file at *constant/polyMesh/boundary*. More information about the organization and format of OpenFOAM input files can be found at OpenFOAM User Guide (OpenCFD Ltd, 2019).

For example:

```
add solid fluid interface "bottom_fluid_surface";
```

Adds fluid boundary named "bottom_fluid_surface" as one of the interface boundaries between solid domain and fluid domain.

205.3.4.184 Modeling, Solid-Fluid Interaction: Defining Solid-Fluid Interface, ESSI Element Nodes

For solid-fluid interaction analysis, solid fluid interface contains information about the Real-ESSI solid interface and OpenFOAM fluid interface. The Real-ESSI solid interface defines Real-ESSI interface nodes and faces.

Real-ESSI interface nodes can be defined as the following:

```
define solid fluid interface ESSI nodes <string>;
```

where:

- <string> specifies the plain text file name that contains the information about Real-ESSI interface nodes.

The format of such text file containing Real-ESSI interface nodes information is:

- Comment line starts with "//"
- Each line defining a Real-ESSI interface node has four entries separated by space(s):
  - 1^{st} entry: Real-ESSI node ID;
  - 2^{nd} entry: x coordinate of Real-ESSI interface node;
  - 3^{rd} entry: y coordinate of Real-ESSI interface node;
  - 4^{th} entry: z coordinate of Real-ESSI interface node;

For example:

```
define solid fluid interface ESSI nodes "ESSI_nodes_info.fei";
```

Defines Real-ESSI nodes at solid fluid interface with file named "ESSI_nodes_info.fei". An example file of "ESSI_nodes_info.fei" is provided below:

```
//=====================================================
// Files contains information about ESSI nodes at solid fluid interface, have 4 columns
// 1st column: ESSI node ID
// 2nd column: coordinate x
// 3rd column: coordinate y
// 4th column: coordinate z
//=====================================================
1 0.00 0.00 0.00
12 30.00 0.00 0.00
33 0.00 0.00 10.00
...
```

205.3.4.185 Modeling, Solid-Fluid Interaction: Defining Solid-Fluid Interface, ESSI Element Faces

For solid-fluid interaction analysis, solid fluid interface contains information about Real-ESSI solid interface and OpenFOAM fluid interface. The Real-ESSI solid interface defines Real-ESSI interface nodes and faces.

Real-ESSI interface faces are quads that can be defined as:

```
1 define solid fluid interface ESSI faces <string>;
```

where:

- <string> specifies the plain text file name that contains the information about Real-ESSI interface elements faces.

The format of such text file containing Real-ESSI interface element faces information is:

- Comment line starts with "//"
- Each line defining a Real-ESSI interface element face, i.e., a face of a brick element, in effect a quad consisting of four Real-ESSI interface nodes. Each line has six entries separated by spaces:
  - 1^{st} entry: Real-ESSI interface face ID;
  - 2^{nd} entry: Real-ESSI element ID that contains the Real-ESSI interface face;
  - 3^{rd} entry: Real-ESSI node ID for Real-ESSI interface face, i.e., a brick face, a quad, vertex 1;
  - 4^{th} entry: Real-ESSI node ID for Real-ESSI interface face, i.e., a brick face, a quad, vertex 2;
  - 5^{th} entry: Real-ESSI node ID for Real-ESSI interface face, i.e., a brick face, a quad, vertex 3;
  - 6^{th} entry: Real-ESSI node ID for Real-ESSI interface face, i.e., a brick face, a quad, vertex 4;

  It is noted that the ordering of the four vertex Real-ESSI nodes is very important. The ordering should be taken such that the face normal vector points outwards to the fluid domain following the convention of right hand rule.

For example:

```
1 define solid fluid interface ESSI faces "ESSI_faces_info.fei";
```

Defines Real-ESSI faces at solid fluid interface with file named "ESSI_faces_info.fei". An example file of "ESSI_faces_info.fei" is provided below:

```
//================================================
// Files contains information about ESSI faces (quads) at solid fluid ↩
    interface, have 6 columns
// 1st column: ESSI face ID
// 2nd column: ESSI Element ID that ESSI face belongs to
// 3rd column: ESSI node ID for quad vertex 1
// 4th column: ESSI node ID for quad vertex 2
// 5th column: ESSI node ID for quad vertex 3
// 6th column: ESSI node ID for quad vertex 4
//================================================
1 276 1 25 272 5
2 277 25 26 273 272
3 278 26 27 274 273
4 279 27 28 275 274
...
```

205.3.4.186 Modeling, Solid-Fluid Interaction: Defining Solid-Fluid Interface FOAM Nodes

For solid-fluid interaction analysis, solid fluid interface contains information about the Real-ESSI solid interface and OpenFOAM fluid interface. The OpenFOAM fluid interface needs to define OpenFOAM interface nodes and faces.

OpenFOAM interface nodes can be defined as the following:

```
define solid fluid interface FOAM nodes <string>;
```

where:

- <string> specifies the plain text file name that contains the information about OpenFOAM interface nodes.

The format of such text file containing OpenFOAM interface nodes information is:

- Comment line starts with "//"
- Each line defining a OpenFOAM interface node has four entries separated by space(s):
  - 1^{st} entry: OpenFOAM node ID;
  - 2^{nd} entry: x coordinate of OpenFOAM interface node;
  - 3^{rd} entry: y coordinate of OpenFOAM interface node;
  - 4^{th} entry: z coordinate of OpenFOAM interface node;

For example:

```
define solid fluid interface FOAM nodes "foam_nodes_info.fei";
```

Defines OpenFOAM nodes at solid fluid interface with file named "foam_nodes_info.fei". An example file of "foam_nodes_info.fei" can be:

```
//=====================================================
// Files contains information about Foam nodes at solid fluid interface, have 4 columns
// 1st column: Foam node ID
// 2nd column: coordinate x
// 3rd column: coordinate y
// 4th column: coordinate z
//=====================================================
1 0.00 0.00 0.00
12 30.00 0.00 0.00
33 0.00 0.00 10.00
...
```

205.3.4.187 Modeling, Solid-Fluid Interaction: Defining Solid-Fluid Interface FOAM Faces

For solid-fluid interaction analysis, solid fluid interface contains information about the Real-ESSI solid interface and OpenFOAM fluid interface. The OpenFOAM fluid interface needs to define OpenFOAM interface nodes and faces.

OpenFOAM interface faces are quads that can be defined as the following:

```
1 define solid fluid interface FOAM faces <string>;
```

where:

- `<string>` specifies the plain text file name that contains the information about OpenFOAM interface faces.

The format of such text file containing OpenFOAM interface faces information is:

- Comment line starts with "//"
- Each line defining a OpenFOAM interface face, i.e., a quad consisting of four OpenFOAM interface nodes. Each line has five entries separated by space(s):
  - 1^{st} entry: OpenFOAM face ID;
  - 2^{nd} entry: OpenFOAM node ID for OpenFOAM interface face, i.e., a quad, vertex 1;
  - 3^{rd} entry: OpenFOAM node ID for OpenFOAM interface face, i.e., a quad, vertex 2;
  - 4^{th} entry: OpenFOAM node ID for OpenFOAM interface face, i.e., a quad, vertex 3;
  - 5^{th} entry: OpenFOAM node ID for OpenFOAM interface face, i.e., a quad, vertex 4;

  It is noted that the ordering of the four vertex OpenFOAM nodes is very important. The ordering should be consistent with the OpenFOAM faces definition in the OpenFOAM input file at *constant/polyMesh/faces*. Detailed information about the organization and format of OpenFOAM input files can be found at OpenFOAM User Guide (OpenCFD Ltd, 2019).

For example:

```
1 define solid fluid interface FOAM faces "foam_faces_info.fei";
```

Defines OpenFOAM faces at solid fluid interface with file named "foam_faces_info.fei". An example file of "foam_faces_info.fei" can be:

```
1  //=========================================
2  // Files contains information about Foam faces (quads) at solid fluid ↩
       interface, have 5 columns
3  // 1st column: Foam face ID
4  // 2rd column: Foam node ID for quad vertex 1
5  // 3th column: Foam node ID for quad vertex 2
6  // 4th column: Foam node ID for quad vertex 3
7  // 5th column: Foam node ID for quad vertex 4
8  //=========================================
9  50 8 0 4 404
10 51 9 8 404 405
11 52 10 9 405 406
12 53 11 10 406 407
13 54 12 11 407 408
14 55 13 12 408 409
15 ...
```

205.3.5 Simulation

205.3.5.1 Simulation, Solvers: Sequential Solvers

```
1 define solver sequential <profilespd|umfpack >;
```

Available sequential solvers are:

- `ProfileSPD` is used for symmetric matrices
- `UMFPack` is used for non-symmetric matrices and indefinite matrices

NOTE: USE THE SAME SOLVER FOR EACH ANALYSIS, FOR ALL LOAD STAGES!
Use the same solver, parallel or sequential, with same solver options, for all loading stages in an analysis. System matrices, mass, damping and stiffness and packaged in a different way for different solvers, and different solver options, so changing solver type between different stages will affect access to those matrices, and will affect results...

205.3.5.2 Simulation, Solvers: Parallel Solvers

```
define solver parallel petsc <petsc_options> ;
```

NOTE: USE THE SAME SOLVER FOR EACH ANALYSIS, FOR ALL LOAD STAGES!

Use the same solver, parallel or sequential, with same solver options, for all loading stages in an analysis. System matrices, mass, damping and stiffness and packaged in a different way for different solvers, and different solver options, so changing solver type between different stages will affect access to those matrices, and will affect results...

Direct Solvers

Command Example for a direct solver:

```
define solver parallel petsc "-ksp_type preonly -pc_type lu" ;
define solver parallel petsc "-pc_type lu -pc_factor_mat_solver_package mumps" ;
define solver parallel petsc "-pc_type lu -pc_factor_mat_solver_package ↩
    superlu" ;
```

As shown in the Command Example, "-ksp_type" represents the solver type, "-pc_type" represents the preconditioner types. By defining "preonly", petsc will use the direct solver, and its type is defined in the preconditioner types. In addition, "lu" represents LU factorization in the direct solver. The solver package "mumps" is designed for finite-element methods, and can interleave Gauss elimination process with the assembly process of global stiffness from the local element stiffness matrices. It is also noted that "mumps" can solve symmetric indefinite matrices.

The solver package "superlu" pivots the large-scale sparse matrices to numerous small-scale dense matrices for acceleration.

Iterative Solvers

Command Example for an iterative solver:

```
define solver parallel petsc "-ksp_type gmres -pc_type jacobi";
define solver parallel petsc "-ksp_type cg -pc_type ilu";
```

PETSc contains many iterative solvers and preconditioner for large-scale problems, and they are all available in with Real-ESSI.

Tables 205.1 and 205.2 on next pages present a full set of options for iterative solvers and preconditioners.

Table 205.1: Available Parallel Iterative Solvers

Solver Name	Method
"richardson"	Richardson
"chebyshev"	Chebyshev
"cg"	Conjugate Gradient
"bicg"	BiConjugate Gradient
"gmres"	Generalized Minimal Residual
"fgmres"	Flexible Generalized Minimal Residual
"dgmres"	Deflated Generalized Minimal Residual
"gcr"	Generalized Conjugate Residual
"bcgs"	BiCGSTAB
"cgs"	Conjugate Gradient Squared
"tfqmr"	Transpose-Free Quasi-Minimal Residual (1)
"tcqmr"	Transpose-Free Quasi-Minimal Residual (2)
"cr"	Conjugate Residual
"lsqr"	Least Squares Method

Table 205.2: Available Parallel Iterative Preconditioners

Preconditioner Name	Method
jacobi	Jacobi
bjacobi	Block Jacobi
sor	SOR (and SSOR)
eisenstat	SOR with Eisenstat trick
icc	Incomplete Cholesky
ilu	Incomplete LU
asm	Additive Schwarz
gasm	Generalized Additive Schwarz
gamg	Algebraic Multigrid
bddc	Balancing Domain Decomposition by Constraints
ksp	Linear solver
composite	Combination of preconditioners

For more available solver options please consult the PETSc documentation (http://www.mcs.anl.gov/petsc/petsc-current/docs/manual.pdf).

205.3.5.3 Simulation: Static Solution Advancement

```
simulate <.> steps using static algorithm;
```

205.3.5.4 Simulation: Dynamic Solution Advancement with the Constant Time Step

```
1 simulate <.> steps using transient algorithm time_step = <T>;
```

205.3.5.5 Simulation: Dynamic Solution Advancement with Variable Time Step

```
simulate <.> steps using variable transient algorithm
  time_step = <T>
  minimum_time_step = <.>
  maximum_time_step = <.>
  number_of_iterations = <.>
```

205.3.5.6 Simulation: Generalized Eigenvalue Analysis

At any given point in an analysis a generalized eigenvalue analysis of the system can be performed, based on the current mass and tangent stiffness matrices. The command to do this is:

```
1 simulate using eigen algorithm number_of_modes = <.>;
```

The first `number_of_modes` eigenvalues are displayed on screen after the analysis is performed. If more eigenvalues are requested than degrees-of-freedom the system has, the excess reported values are set to NaN (not a number).

Description of output for nodes of different dof types can be found in section 206.5.6.

205.3.5.7 Simulation: Displacement Control

```
1 define static integrator displacement_control using node # <.> dof DOFTYPE ↩
      increment <L>;
```

205.3.5.8 Simulation: Load, Control, Factor Increment

```
1 define load factor increment <.>;
```

205.3.5.9 Simulation: Dynamic Integrator, Newmark Method

```
1 define dynamic integrator Newmark with gamma = <.> beta = <.>;
```

See also the list of the reserved keywords from Section 205.7 on page 1163.

205.3.5.10 Simulation: Dynamic Integrator, Hilber Hughes Taylor, HHT, α Method

```
1 define dynamic integrator Hilber_Hughes_Taylor with alpha = <.>;
```

See also the list of the reserved keywords from Section 205.7 on page 1163.

205.3.5.11 Simulation: Absolute Convergence Criteria

```
define convergence test
  < Absolute_Norm_Unbalanced_Force | Absolute_Norm_Displacement_Increment >
  tolerance = <.>
  maximum_iterations = <.>;
```

This command sets the convergence criteria for global iterative solvers. If the system-of-equation to be solved is

$$K_T \Delta U = \Delta R$$

where K_T is the current tangent stiffness operator/matrix, ΔU is the displacement increment, and ΔR is the residual. The convergence criteria is based on:

- The l^2 norm of the displacement increment: $\|\Delta U\|_2 < TOL$.
- The l^2 norm of the unbalanced force: $\|\Delta R\|_2 < TOL$.

The convergence test should be defined before the algorithms.

205.3.5.12 Simulation: Average Convergence Criteria

```
1 define convergence test ↩
      <Average_Norm_Unbalanced_Force|Average_Norm_Displacement_Increment>
2 tolerance = <.>
3 maximum_iterations = <.> ;
```

This command sets the convergence criteria for global iterative solvers. If the system-of-equation to be solved is

$$K_T \Delta U = \Delta R$$

Where K_T is the current tangent stiffness operator (dynamic tangent for dynamic analysis), ΔU is the displacement increment, and ΔR is the residual. The convergence criteria can be based off

- The average l^2 norm of the displacement increment: $\|\Delta U\|_2/\sqrt{N} < TOL$.
- The average l^2 norm of the unbalanced force: $\|\Delta R\|_2/\sqrt{N} < TOL$.

where N is the number of DOFs in the system-of-equations.

The convergence test should be defined before the algorithms.

205.3.5.13 Simulation: Relative Convergence Criteria

```
define convergence test ↩
    <Relative_Norm_Unbalanced_Force|Relative_Norm_Displacement_Increment>
tolerance = <.>
minimum_absolute_tolerance = <.>
maximum_iterations = <.> ;
```

This command sets the convergence criteria for global iterative solvers. If the system-of-equation to be solved is

$$K_T \Delta U = \Delta R$$

Where K_T is the current tangent stiffness operator (dynamic tangent for dynamic analysis), ΔU is the displacement increment, and ΔR is the residual. The convergence criteria can be based on

- The relative l^2 norm of the displacement increment: $\|\Delta U\|_2/\|U_0\|_2 < TOL$ or $\|\Delta U\|_2 < MIN\_ABS\_TOL$.
- The relative l^2 norm of the unbalanced force: $\|\Delta R\|_2/\|R_0\|_2 < TOL$ or $\|\Delta R\|_2 < MIN\_ABS\_TOL$.

Where,
R_0 is the external force in the beginning
U_0 is the solution after the first iteration.
Since U_0 is zero before the first iteration, the relative norm of the displacement increment in the first iteration would be equal to 1.

The convergence test should be defined before the algorithms.

205.3.5.14 Simulation: Solution Algorithms

```
1 define algorithm < With_no_convergence_check | linear_elastic | Newton | ↩
    Modified_Newton | Newton_With_LineSearch >;
```

```
1 define algorithm < Newton_With_Subincrement >
2   using minimum_time_step = <.> ;
```

If the current specified load factor $\Delta\lambda$ (for static) or time step Δt (for dynamic) fails to achieve the convergence in the specified maximum number of iterations, the algorithm `Newton_With_Subincrement` will subdivide the current step into two sub steps of load increment $\Delta\lambda^{new} = \Delta\lambda/2$ (for static) or time step $\Delta t^{new} = \Delta t/2$ (for dynamic).

- `minimum_time_step` specifies the allowed minimum load factor $\Delta\lambda$ (for static) or time step Δt (for dynamic), that the algorithm should sub-divide to achieve convergence. If the subdivided step size becomes less than the `minimum_time_step`, the algorithm returns failure to convergence.

Note: If any Newton algorithm is used, the convergence test should be defined before the algorithms.

205.3.5.15 Simulation: Constitutive Integration Algorithm

Starting with version 03-NOV-2015, NDMaterial class of materials require explicit specification of the constitutive integration algorithm. This is done with the command:

```
define NDMaterial constitutive integration algorithm Forward_Euler;
```

```
define NDMaterial constitutive integration algorithm Forward_Euler_Subincrement
  number_of_subincrements = <.> ;
```

```
define NDMaterial constitutive integration algorithm Backward_Euler
  yield_function_relative_tolerance = <.>
  stress_relative_tolerance = <.>
  maximum_iterations = <.>;
```

```
define NDMaterial constitutive integration algorithm Backward_Euler_Subincrement
  yield_function_relative_tolerance = <.>
  stress_relative_tolerance = <.>
  maximum_iterations = <.>
  allowed_subincrement_strain = <.> ;
```

The command specifies the method, tolerances and maximum number of iterations used to do material point integrations. The parameters are:

- `number_of_subincrements` Specify the number of subincrements in forward Euler subincrement algorithm.
- `yield_function_relative_tolerance` Specify the relative tolerance of the yield surface value in the family of backward Euler algorithm.
- `stress_relative_tolerance` Specify the relative stress tolerance in the family of backward Euler algorithm. The stress increment is within this tolerance for each step unless the integration fails. Frobenius norm is used to calculate the stress norm.
- `maximum_iterations` Specify the maximum number of iterations in backward Euler algorithm.
- `allowed_subincrement_strain` defines the maximum value of allowed strain increment in backward Euler subincrement method. If one of strain component increments is greater than the user-defined allowed strain increment, strain increment will be divided into subincrements based on the allowed subincrement. For example, if the `strain_increment` is 0.05, and the `allowed_subincrement_strain` is 0.01. The number of subincrements will be $0.05/0.01 = 5$. A small allowed subincrement leads to more accurate results, however, it takes more time. For the simple nonlinear materials, like

von Mises linear hardening, the allowed subincrement can be as big as 5 percent. For the complicated nonlinear materials, like hyperbolic Drucker-Prager Armstrong-Frederick hardening material, the allowed subincrement should be much smaller in the range of 1E-4.

205.3.5.16 Simulation: Status Check

All simulate commands set the variable SIMULATE_EXIT_FLAG automatically upon exit. This flag can be used to check whether the simulation concluded normally (SIMULATE_EXIT_FLAG = 0), failed (SIMULATE_EXIT_FLAG < 0), or finished with warnings (SIMULATE_EXIT_FLAG > 0).

For example, the following simulations will fail.

```
atmospheric_pressure = 101325*Pa;

pstart = 3000*kPa;

//SAniSand 2004 calibration for Toyoura Sand.
add material # 1 type sanisand2004
    mass_density = 2100.0 * kg / m^3
    e0 = 0.735
    sanisand2004_G0 = 125. poisson_ratio = 0.05
    sanisand2004_Pat = atmospheric_pressure
    sanisand2004_p_cut = 0.1*atmospheric_pressure
    sanisand2004_Mc = 1.25 sanisand2004_c = 0.712
    sanisand2004_lambda_c = 0.019 sanisand2004_xi = 0.7
    sanisand2004_ec_ref = 0.934 sanisand2004_m = 0.01
    sanisand2004_h0 = 7.05 sanisand2004_ch = 0.968
    sanisand2004_nb = 1.1 sanisand2004_A0 = 0.704
    sanisand2004_nd = 3.5 sanisand2004_z_max = 4.
    sanisand2004_cz = 600.
    initial_confining_stress = 1*Pa ;

simulate constitutive testing DIRECT_STRAIN
    use material # 1
    scale_factor = 1.
    series_file = "increments.txt"
    sigma0 = ( -pstart*kPa , -pstart*kPa , -pstart*kPa , 0*Pa , 0*Pa , 0*Pa )
    verbose_output = 1;


if(SIMULATE_EXIT_FLAG == 0)
{
    print "All Good!";
}
else
{
    print "Something went wrong. Error code = ";
    print SIMULATE_EXIT_FLAG;
}

bye;
```

The above simulation fails because the integration method for the constitutive model is not set (see 205.3.5.15). Therefore, the second branch of the 'if' statement will execute.

205.3.5.17 Simulation: Save State

Save ESSI system state to a file, to prepare for a restart.

```
save model;
```

This command will save the state of a model, an ESSI system, in file. Filename for the save file will be created from a model name, loading stage name, and current loading time.

For example, for a model that contains the following commands in the input file:

```
model name "ESSI_model";
. . .
new loading stage "Loading_stage_2";
. . .
simulate 100 steps using transient algorithm time_step = 0.005*s;

save model;
```

will be saved in file with the following name:

```
ESSI_model_Loading_stage_2_at_time_0.5second_RESTART.essi
```

since there were 100 steps with $\Delta t = 0.005$s, and that advances the solution to $t = 0.5$s.

205.3.5.18 Simulation: Restart Simulation

Restart simulation after stage or a step, from a saved ESSI system model file.

```
1 restart model using file "filename";
```

Here, ESSI system model is saved in a file `filename`, see command for saving model on page 1120.

For example, to restart simulation from a saved file described above, restart will be initiated in a new input file by using the following command:

```
1 restart model using file ↩
    "ESSI_model_Loading_stage_2_at_time_0.5second_RESTART.essi";
```

All the results from previous loading stages will be saved in the restart file. From here on, analyst can start new loading stages, etc.

205.3.5.19 Simulation: Return Value for `simulate` Command

Simulate command, `simulate`, returns status of simulation progress. For each successful step, `simulate` returns value 0 while for a failed step it returns –1. This is useful as analyst can control solution process, and change algorithm if predefined algorithm fails to converge.

For example the example if listing 205.8, simulation part of a larger examples, will perform a change of stepping algorithm from the load control to displacement control upon failure of load control to converge.

```
step=0;
Nsteps = 100;
define load factor increment 0.01; // Start with load-control

simulation_status=simulate 1 steps using static algorithm;

while (step<(Nsteps-1))
  {
    if(simulation_status>=0) // Converged, continue using load-control
      {
        simulation_status=simulate 1 steps using static algorithm;
      }
    else // Not converged, so change to displacement-control
     {
       define static integrator displacement_control using node # 1 dof ux ↩
           increment 1E-3*m;
       simulate 1 steps using static algorithm;
     }
    step=step+1;
  }
```

Figure 205.8: Interactive simulation control using feedback (return value) from the `simulate` command.

It should be noted that the idea for interactive control of simulation process comes from FEAP (Zienkiewicz and Taylor, 1991b) and later from OpenSEES (Mazzoni et al., 2002) where it was implemented with early extension of OpenSees command language with using Tcl in early 2000s.

An example of the above feedback mechanism is provided below

```
model name "vm";
add material # 1 type vonMises
    mass_density = 0.0*kg/m^3
    elastic_modulus = 2E7*N/m^2
    poisson_ratio = 0.0
    von_mises_radius = 1E5*Pa
    kinematic_hardening_rate = 0*Pa
    isotropic_hardening_rate = 0*Pa;
// define the node:
add node # 1 at (0*m,0*m,1*m) with 3 dofs;
add node # 2 at (1*m,0*m,1*m) with 3 dofs;
add node # 3 at (1*m,1*m,1*m) with 3 dofs;
add node # 4 at (0*m,1*m,1*m) with 3 dofs;
add node # 5 at (0*m,0*m,0*m) with 3 dofs;
add node # 6 at (1*m,0*m,0*m) with 3 dofs;
add node # 7 at (1*m,1*m,0*m) with 3 dofs;
add node # 8 at (0*m,1*m,0*m) with 3 dofs;
// Define the element.
add element # 1 type 8NodeBrick using 2 Gauss points each direction with nodes ↩
    (1, 2, 3, 4, 5, 6, 7, 8) use material # 1;

new loading stage "shearing";
//fix the bottom totally
fix node # 5 dofs all;
fix node # 6 dofs all;
fix node # 7 dofs all;
fix node # 8 dofs all;
// Fix the other 2 directions on the top.
fix node # 1 dofs uy uz ;
fix node # 2 dofs uy uz ;
fix node # 3 dofs uy uz ;
fix node # 4 dofs uy uz ;
add load # 101 to node # 1 type linear Fx = 40 * kN;
add load # 102 to node # 2 type linear Fx = 40 * kN;
add load # 103 to node # 3 type linear Fx = 40 * kN;
add load # 104 to node # 4 type linear Fx = 40 * kN;
define solver UMFPack;
//define algorithm With_no_convergence_check ↩
    ;Norm_Displacement_Increment;Norm_Unbalance
define convergence test Absolute_Norm_Displacement_Increment
    tolerance = 1E-3
    maximum_iterations = 5
    ;
define algorithm Newton;
define NDMaterial constitutive integration algorithm Backward_Euler
    yield_function_relative_tolerance = 1E-7
    stress_relative_tolerance = 1E-7
    maximum_iterations = 100;

// ********************************************************************************
step=0;
```

```
Nsteps = 10;
define load factor increment 1/Nsteps; // Start with load-control
// ********************************************************************************
// Simulate with status check:
// ********************************************************************************
mystatus=simulate 1 steps using static algorithm;
while (step<(Nsteps-1)){
    step=step+1;
    if(mystatus>=0){ // Converged, so continue using load-control
        mystatus=simulate 1 steps using static algorithm;
    }
    else{ // Not converged, so change to displacement-control
        define static integrator displacement_control using node # 1 dof ux ↩
            increment 1E-3*m ;
        simulate 1 steps using static algorithm;
    }
}

bye;
```

Resulting terminal output, showing a switch between two solution control mechanisms is provided below:

```


  The Finite Element Interpreter

  MS ESSI
  Earthquake Soil Structure Interaction Simulator

  Sequential processing mode.

Version Name : Academic Version. Release date: Apr 14 2017 at 17:19:55. Tag: ↩
    c24e557a56
Version Branch : yuan
Compile Date : Apr 15 2017 at 20:28:11
Compile User : yuan
Compile Sysinfo: cml01 4.4.0-72-generic x86_64 GNU/Linux
Runtime User : Runtime Sysinfo: Time Now : Apr 15 2017 at 21:15:40

Static startup tips:
 * Remember: Every command ends with a semicolon ';'.
 * Type 'quit;' or 'exit;' to finish.
 * Run 'essi -h' to see available command line options.

Including: "main.fei"


Model name is being set to "vm"

```

```
28
29
30 Starting new stage: shearing
31
32 changing previous_stage_name from to shearing
33 Setting set_constitutive_integration_method = 2
34
35 Starting sequential static multi-step analysis
36 ================================================================================================
37 Creating analysis ↩
      model.....................................................................Pass!
38 Checking constraint ↩
      handler...................................................................Pass!
39 Checking ↩
      numberer..................................................................Pass!
40 Checking analysis ↩
      algorithm.................................................................Pass!
41 Checking system of equation ↩
      handler...................................................................Pass!
42 Checking static integration ↩
      handler...................................................................Pass!
43
44
45  Writing Initial Conditions and (0) - Outputting mesh.
46
47 Static Analysis: [ 1/1 ]
48   [iteration 1 /5 ] Convergence Test: Absolute Norm Displacement ↩
        Increment::(tol: 0.001)
49                    Absolute Norm deltaF: 1.6396e-12
50                    Absolute Norm deltaU: 0.0032
51                    Average Norm deltaF: 8.1981e-13
52                    Average Norm deltaU: 0.0016
53                    Relative Norm deltaF: 2.0495e-16
54                    Relative Norm deltaU: 0.0032
55   [iteration 2 /5 ] Convergence Test: Absolute Norm Displacement ↩
        Increment::(tol: 0.001)
56                    Absolute Norm deltaF: 4.5475e-13
57                    Absolute Norm deltaU: 5.2683e-19
58                    Average Norm deltaF: 2.2737e-13
59                    Average Norm deltaU: 2.6341e-19
60                    Relative Norm deltaF: 5.6843e-17
61                    Relative Norm deltaU: 1.6463e-16
62
63 > Analysis End ↩
      ---------------------------------------------------------------------------
64
65 Starting sequential static multi-step analysis
66 ================================================================================================
67 Creating analysis ↩
      model.....................................................................Pass!
68 Checking constraint ↩
```

```
   handler.....Pass!
Checking ↩
   numberer.....Pass!
Checking analysis ↩
   algorithm.....Pass!
Checking system of equation ↩
   handler.....Pass!
Checking static integration ↩
   handler.....Pass!

Static Analysis: [1 /1 ]
  [iteration 1 /5 ] Convergence Test: Absolute Norm Displacement ↩
     Increment::(tol: 0.001)
                  Absolute Norm deltaF: 2.7285e-12
                  Absolute Norm deltaU: 0.0032
                  Average Norm deltaF: 1.3642e-12
                  Average Norm deltaU: 0.0016
                  Relative Norm deltaF: 3.4106e-16
                  Relative Norm deltaU: 0.0032
  [iteration 2 /5 ] Convergence Test: Absolute Norm Displacement ↩
     Increment::(tol: 0.001)
                  Absolute Norm deltaF: 3.5225e-12
                  Absolute Norm deltaU: 7.8429e-19
                  Average Norm deltaF: 1.7612e-12
                  Average Norm deltaU: 3.9214e-19
                  Relative Norm deltaF: 4.4031e-16
                  Relative Norm deltaU: 2.4509e-16

> Analysis End ↩
    ------------------------------------------------------------------------

Starting sequential static multi-step analysis
================================================================================
Creating analysis ↩
   model.....Pass!
Checking constraint ↩
   handler.....Pass!
Checking ↩
   numberer.....Pass!
Checking analysis ↩
   algorithm.....Pass!
Checking system of equation ↩
   handler.....Pass!
Checking static integration ↩
   handler.....Pass!

Static Analysis: [1 /1 ]
  [iteration 1 /5 ] Convergence Test: Absolute Norm Displacement ↩
     Increment::(tol: 0.001)
                  Absolute Norm deltaF: 1.819e-12
                  Absolute Norm deltaU: 0.0032
```

```
                    Average Norm deltaF: 9.0949e-13
                    Average Norm deltaU: 0.0016
                    Relative Norm deltaF: 2.2737e-16
                    Relative Norm deltaU: 0.0032
  [iteration 2 /5 ] Convergence Test: Absolute Norm Displacement ↩
     Increment::(tol: 0.001)
                    Absolute Norm deltaF: 2.5724e-12
                    Absolute Norm deltaU: 5.0807e-19
                    Average Norm deltaF: 1.2862e-12
                    Average Norm deltaU: 2.5403e-19
                    Relative Norm deltaF: 3.2155e-16
                    Relative Norm deltaU: 1.5877e-16

> Analysis End ↩
    ------------------------------------------------------------------------

Starting sequential static multi-step analysis
==========================================================================================
Creating analysis ↩
    model.....................................................................Pass!
Checking constraint ↩
    handler...................................................................Pass!
Checking ↩
    numberer..................................................................Pass!
Checking analysis ↩
    algorithm.................................................................Pass!
Checking system of equation ↩
    handler...................................................Pass!
Checking static integration ↩
    handler...................................................Pass!

Static Analysis: [1 /1 ]
  [iteration 1 /5 ] Convergence Test: Absolute Norm Displacement ↩
     Increment::(tol: 0.001)
                    Absolute Norm deltaF: 3132.5
                    Absolute Norm deltaU: 0.0032
                    Average Norm deltaF: 1566.2
                    Average Norm deltaU: 0.0016
                    Relative Norm deltaF: 0.39156
                    Relative Norm deltaU: 0.0032
  [iteration 2 /5 ] Convergence Test: Absolute Norm Displacement ↩
     Increment::(tol: 0.001)
                    Absolute Norm deltaF: 3132.5
                    Absolute Norm deltaU: 0.001253
                    Average Norm deltaF: 1566.2
                    Average Norm deltaU: 0.0006265
                    Relative Norm deltaF: 0.39156
                    Relative Norm deltaU: 0.39156
  [iteration 3 /5 ] Convergence Test: Absolute Norm Displacement ↩
     Increment::(tol: 0.001)
                    Absolute Norm deltaF: 3132.5
```

```
                    Absolute Norm deltaU: 0.001253
                    Average Norm deltaF: 1566.2
                    Average Norm deltaU: 0.0006265
                    Relative Norm deltaF: 0.39156
                    Relative Norm deltaU: 0.39156
  [iteration 4 /5 ] Convergence Test: Absolute Norm Displacement Increment::(tol: 0.001)
                    Absolute Norm deltaF: 3132.5
                    Absolute Norm deltaU: 0.001253
                    Average Norm deltaF: 1566.2
                    Average Norm deltaU: 0.0006265
                    Relative Norm deltaF: 0.39156
                    Relative Norm deltaU: 0.39156
  [iteration 5 /5 ] Convergence Test: Absolute Norm Displacement Increment::(tol: 0.001)
                    Absolute Norm deltaF: 3132.5
                    Absolute Norm deltaU: 0.001253
                    Average Norm deltaF: 1566.2
                    Average Norm deltaU: 0.0006265
                    Relative Norm deltaF: 0.39156
                    Relative Norm deltaU: 0.39156
  [iteration 5/5 ] Convergence Test: Absolute Norm Displacement Increment::(tol: 0.001) !!!FAILED TO CONVERGE!!! [EXITING..]
                    Absolute Norm deltaF: 3132.5
                    Absolute Norm deltaU: 0.001253
                    Average Norm deltaF: 1566.2
                    Average Norm deltaU: 0.0006265
                    Relative Norm deltaF: 0.39156
                    Relative Norm deltaU: 0.39156
NewtonRaphson::solveCurrentStep() -the ConvergenceTest object failed in test()

Static Analysis: [1 /1 ] The Algorithm failed at load factor 0.4
> Analysis End ----------------------------------------------------------------------------

Starting sequential static multistep analysis
=============================================================================================
Creating analysis
   model.....................................................................Pass!
Checking constraint
   handler...................................................................Pass!
Checking
   numberer..................................................................Pass!
Checking analysis
   algorithm.................................................................Pass!
Checking system of equation
   handler...................................................Pass!
Checking static integration
   handler...................................................Pass!

Static Analysis: [1 /1 ]
```

```
  [iteration 1 /5 ] Convergence Test: Absolute Norm Displacement ↩
     Increment::(tol: 0.001)
                  Absolute Norm deltaF: 9310.6
                  Absolute Norm deltaU: 2.6478e-19
                  Average Norm deltaF: 4655.3
                  Average Norm deltaU: 1.3239e-19
                  Relative Norm deltaF: 0.42857
                  Relative Norm deltaU: 2.6478e-19

> Analysis End ↩
    ------------------------------------------------------------------------

Starting sequential static multistep analysis
==========================================================================================
Creating analysis ↩
   model.....................................................................Pass!
Checking constraint ↩
   handler...................................................................Pass!
Checking ↩
   numberer..................................................................Pass!
Checking analysis ↩
   algorithm.................................................................Pass!
Checking system of equation ↩
   handler...................................................Pass!
Checking static integration ↩
   handler...................................................Pass!

Static Analysis: [1 /1 ]
  [iteration 1 /5 ] Convergence Test: Absolute Norm Displacement ↩
     Increment::(tol: 0.001)
                  Absolute Norm deltaF: 1437.5
                  Absolute Norm deltaU: 1.0012e-18
                  Average Norm deltaF: 718.76
                  Average Norm deltaU: 5.006e-19
                  Relative Norm deltaF: 0.10425
                  Relative Norm deltaU: 1.0012e-18

> Analysis End ↩
    ------------------------------------------------------------------------

Starting sequential static multistep analysis
==========================================================================================
Creating analysis ↩
   model.....................................................................Pass!
Checking constraint ↩
   handler...................................................................Pass!
Checking ↩
   numberer..................................................................Pass!
Checking analysis ↩
   algorithm.................................................................Pass!
Checking system of equation ↩
```

```
   handler.....Pass!
Checking static integration ↩
   handler.....Pass!

Static Analysis: [1 /1 ]
  [iteration 1 /5 ] Convergence Test: Absolute Norm Displacement ↩
     Increment::(tol: 0.001)
                  Absolute Norm deltaF: 112.08
                  Absolute Norm deltaU: 5.1789e-19
                  Average Norm deltaF: 56.038
                  Average Norm deltaU: 2.5895e-19
                  Relative Norm deltaF: 0.017652
                  Relative Norm deltaU: 5.1789e-19

> Analysis End ↩
   ------------------------------------------------------------------------

Starting sequential static multistep analysis
==========================================================================================
Creating analysis ↩
   model.....Pass!
Checking constraint ↩
   handler.....Pass!
Checking ↩
   numberer.....Pass!
Checking analysis ↩
   algorithm.....Pass!
Checking system of equation ↩
   handler.....Pass!
Checking static integration ↩
   handler.....Pass!

Static Analysis: [1 /1 ]
  [iteration 1 /5 ] Convergence Test: Absolute Norm Displacement ↩
     Increment::(tol: 0.001)
                  Absolute Norm deltaF: 7.142
                  Absolute Norm deltaU: 8.6792e-19
                  Average Norm deltaF: 3.571
                  Average Norm deltaU: 4.3396e-19
                  Relative Norm deltaF: 0.001399
                  Relative Norm deltaU: 8.6792e-19

> Analysis End ↩
   ------------------------------------------------------------------------

Starting sequential static multistep analysis
==========================================================================================
Creating analysis ↩
   model.....Pass!
Checking constraint ↩
   handler.....Pass!
```

```
Checking ↩
   numberer.....Pass!
Checking analysis ↩
   algorithm.....Pass!
Checking system of equation ↩
   handler.....Pass!
Checking static integration ↩
   handler.....Pass!

Static Analysis: [1 /1 ]
  [iteration 1 /5 ] Convergence Test: Absolute Norm Displacement ↩
     Increment::(tol: 0.001)
                  Absolute Norm deltaF: 0.44693
                  Absolute Norm deltaU: 1.0155e-18
                  Average Norm deltaF: 0.22347
                  Average Norm deltaU: 5.0774e-19
                  Relative Norm deltaF: 8.9267e-05
                  Relative Norm deltaU: 1.0155e-18

> Analysis End ↩
   ------------------------------------------------------------------------

Starting sequential static multistep analysis
==========================================================================================
Creating analysis ↩
   model.....Pass!
Checking constraint ↩
   handler.....Pass!
Checking ↩
   numberer.....Pass!
Checking analysis ↩
   algorithm.....Pass!
Checking system of equation ↩
   handler.....Pass!
Checking static integration ↩
   handler.....Pass!

Static Analysis: [1 /1 ]
  [iteration 1 /5 ] Convergence Test: Absolute Norm Displacement ↩
     Increment::(tol: 0.001)
                  Absolute Norm deltaF: 0.027935
                  Absolute Norm deltaU: 2.1658e-19
                  Average Norm deltaF: 0.013968
                  Average Norm deltaU: 1.0829e-19
                  Relative Norm deltaF: 5.5866e-06
                  Relative Norm deltaU: 2.1658e-19

> Analysis End ↩
   ------------------------------------------------------------------------
How polite! Bye, have a nice day!
```

205.3.5.20 Simulation: New Elastic Loading Case

For design applications, linear elastic analysis cases are performed and later combined, using factors of safety (see section 205.3.5.21 on page 1133) to obtain sectional forces for design.

The command for elastic analysis is:

```
new elastic loading case <string> ;
```

One example is

```
new elastic loading case "case1" ;
```

In a new elastic loading case, all previous loads, load patterns are removed.

To guarantee a fresh start, all commit-displacement at nodes are reset to 0, and all commit-stress/strain at Gauss points are reset to 0.

The following components are kept unchanged in a new elastic loading case:

- material properties.
- mesh connectivity
- boundary conditions.
- acceleration fields.
- damping.

If users want to modify the mesh, a new model is suggested instead of a new elastic loading case.

205.3.5.21 Simulation: Combine Elastic Load Cases

For design applications, elastic load cases, that have been analyzed beforehand, can be superimposed, combined using factors of safety, to obtain internal forces that used for design.

The command for this is

```
combine elastic load cases
  hdf5_filenames_list = <string>
  load_factors_list = <string>
  output_filename = <string>
  ;
```

One example is

```
combine elastic load cases
  hdf5_filenames_list = "test_case1.h5.feioutput test_case2.h5.feioutput"
  load_factors_list = "1.2 1.5"
  output_filename = "combine.h5.feioutput"
  ;
```

- `hdf5_filenames_list` specifies the list of HDF5 output filenames. The list should be separated by either space or comma.
- `load_factors_list` specifies the list of scale factors for each loading case. The list should be separated by either space or comma.
- `output_filename` specifies one output filename of the combined loading cases.

The number of specified files in `hdf5_filenames_list` should be equal to the number of scale factors (factors of safety) in `load_factors_list`.

205.3.5.22 Simulation, Dynamic Solution Advancement for Solid-Fluid Interaction

Dynamic analysis of solid-fluid interaction can be performed using:

```
simulate <.> steps using solid fluid interaction transient algorithm time_step ↩
    = <T>;
```

where:

- <.> is an integer specifying total number of time steps in transient solid fluid interaction analysis.
- time_step = <T> defines the time step for solid fluid interaction transient analysis.

For example:

```
simulate 300 steps using solid fluid interaction transient algorithm time_step ↩
    = 0.01*s;
```

Performs transient solid fluid interaction analysis for 300 steps with time step 0.01 s.

205.3.5.23 Simulation, Dynamic Solution Advancement for Stochastic Finite Element Method

Dynamic analysis of stochastic finite element modeling can be performed using:

```
simulate <.> steps using stochastic transient algorithm time_step = <T>;
```

where:

- <.> is an integer specifying total number of time steps in transient stochastic finite element analysis.
- time_step = <T> defines the time step for stochastic finite element transient analysis.

Please note that the stochastic transient algorithm is different from the general transient algorithm in section 205.3.5.4 in the formulation of unbalanced load for each time step. The stochastic transient algorithm uses directly the incremental external loads to compute the incremental displacements, while the general transient algorithm accounts for the correction from resisting forces in the formulation of unbalanced forces.

For deterministic and probabilistic, linear elastic problems, both transient algorithms would produce the same response. Stochastic transient algorithm is more efficient because there is no need to compute resisting forces.

For deterministic, nonlinear inelastic problems, the general transient algorithm is more accurate due to the corrections from resisting forces. The accuracy of stochastic transient algorithm can be improved using smaller loading increments. For probabilistic, nonlinear inelastic problems, the accuracy of the general transient algorithm can only be guaranteed if the number of polynomial chaos terms used in probabilistic constitutive modeling is equal or close enough to the number of polynomial chaos terms in global level. Otherwise, it is recommended to use the stochastic transient algorithm for dynamic analysis of stochastic finite element modeling.

For example:

```
simulate 300 steps using stochastic transient algorithm time_step = 0.01*s;
```

Performs transient stochastic finite element analysis for 300 steps with time step 0.01s.

205.3.5.24 Simulation, Sobol Sensitivity Analysis

Sobol sensitivity analysis can be performed using:

```
1 Sobol sensitivity analysis of node # <.> dof DOFTYPE peak response from random ↩
     field # <.>
2 pc_coefficient_hdf5 = "pc_coefficient_hdf5_file_name"
3 output_hdf5 = "output_hdf5_file_name";
```

where:

- `node #` specify the node tag.
- `DOFTYPE` specify the dof to perform sensitivity analysis. It can be either `ux`, `uy` or `uz`.
- `random field #` specify the random field polynomial chaos bases of the stochastic nodal response.
- `pc_coefficient_hdf5` specify the name of a hdf5 file that contains simulation results of polynomial chaos coefficients.
- `output_hdf5` specify the name of the output hdf5 file for sensitivity analysis.

The output hdf5 format for sensitivity analysis is given as below:

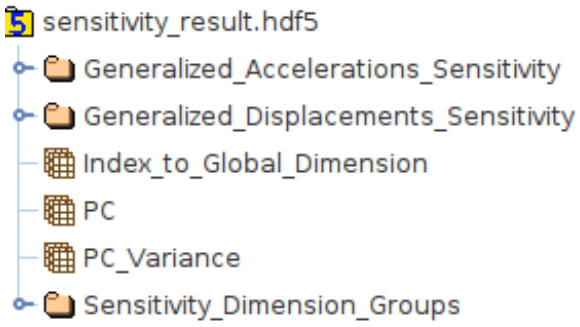


Figure 205.9: Overall data structure of output hdf5 file for sensitivity analysis.

Figure 205.9 shows the overall data structure organization of the output hdf5 file of sensitivity analysis.

- Generalized_Accelerations_Sensitivity data group:

  Contains Sobol sensitivity analysis results for stochastic nodal acceleration response. It contains the following datasets and data groups as shown in Figure 205.10.

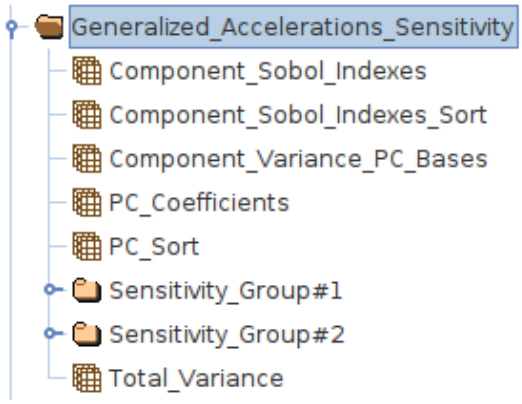


Figure 205.10: Datasets and data groups in Generalized_Accelerations_Sensitivity data group.

- Component_Sobol_Indexes dataset:

  Is a column vector containing the computed Sobol Indexes for each component of polynomial chaos (PC) bases.

- Component_Sobol_Indexes_Sort dataset:

  Is a column vector containing the computed Sobol Indexes for each component of polynomial chaos (PC) bases, in descending order.

- Component_Variance_PC_Bases dataset:

  Is a column vector containing the computed variance for each component of polynomial chaos (PC) bases.

- PC_Coefficients dataset:

  Is a column vector containing the polynomial chaos coefficients corresponding to PC bases specified in PC dataset.

- PC_Sort dataset:

  Is a 2D array that describes the sorted PC bases corresponding to Component_Sobol_Indexes_Sort dataset. PC_Sort$_{ij}$ denotes the order of polynomial chaos dimension ξ_j that contributes to the i^{th} sorted, PC basis.

- Total_Variance dataset:

  Is a scalar, variance of nodal stochastic response.

  In addition to the above datasets, there will be sub data group(s) containing sensitivity analysis results corresponding to each of the defined sensitivity dimension groups. For example, we

have two sub data groups Sensitivity_Group#1 and Sensitivity_Group#2 for this specific hdf5 output for sensitivity analysis. Within these sub data groups, taking Sensitivity_Group#1 as an example, the output data is organized as shown in Figure 205.11.

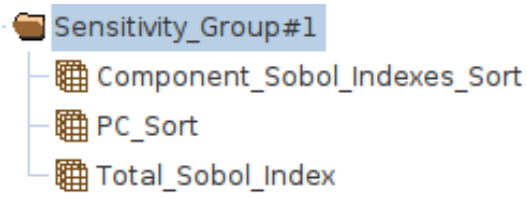


Figure 205.11: Datasets in Sensitivity_Group#1.

It includes the following datasets:

* Component_Sobol_Indexes_Sort dataset:
  Is a column vector containing the computed Sobol Indexes for part of polynomial chaos (PC) bases specified through the sensitivity dimension group, in descending order.
* PC_Sort dataset:
  Is a 2D array that describes the sorted, part of PC bases specified through the sensitivity dimension group, corresponding to Component_Sobol_Indexes_Sort dataset. PC_Sort$_{ij}$ denotes the order of polynomial chaos dimension ξ_j that contributes to the i^{th} sorted, PC basis.
* Total_Sobol_Index dataset:
  Is a scalar, total Sobol sensitivity index for PC bases specified in the sensitivity dimension group.

- Generalized_Displacements_Sensitivity data group:

  Contains Sobol sensitivity analysis results for stochastic nodal displacement response. The configuration of data structure for Generalized_Displacements_Sensitivity data group is the same as Generalized_Accelerations_Sensitivity data group.

- Index_to_Global_Dimension dataset:

  Contains a column vector of integers, which specifies the global dimension IDs for the polynomial chaos bases used to represent the nodal stochastic response.

- PC dataset:

Is a 2D array that describes the multi-dimensional Hermite polynomial chaos (PC) bases for representing the nodal stochastic response. $\boldsymbol{PC_{ij}}$ denotes the order of polynomial chaos dimension ξ_j that contributes to the i^{th} multidimensional Hermite PC basis.

- PC_Variance dataset:

  Is a column vector specifying the variances of Hermite PC basis.

- Sensitivity_Dimension_Groups data group:

  Contains the information about the sensitivity dimension groups defined for sensitivity analysis. Each sensitivity dimension group would be defined by a dataset within the data group as shown in Figure 205.12. Each dataset contains a column vector of integers, specifying the dimension IDs in specific sensitivity dimension group.

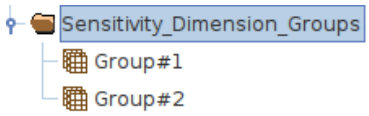


Figure 205.12: Datasets in Sensitivity_Dimension_Groups data group.

205.3.5.25 Simulation: 3D 3C Wave Field Inversion

```
generate domain reduction method motion file from 3D_3C_wave_field_inversion
    target_nodes_filename = <string>
    target_motions_filename = <string>
    time_step = <T>
    hdf5_file = <string>;
```

Example:

```
generate domain reduction method motion file from 3D_3C_wave_field_inversion
    target_nodes_filename = "Target_Nodes.txt"
    target_motions_filename = "Target_Motions.txt"
    time_step = 0.01*s
    hdf5_file = "DRM_Input_from_Inverse_Motion.hdf5";
```

where:

- `target_nodes_filename` is the file name for a file that contains the list of nodes, represented by their tags, where target motions are designated.

  One example of target nodes file is given below.

```
5
15
25
35
46
48
50
52
```

- `target_motions_filename` is the file name for a file that contains the list of target motions at the corresponding target nodes. Note that the length, or number of rows, and the ordering of the target motions file must be the same as those of the target nodes file.

  One example of target motions file is given below.

```
Displacement_1000steps_with_CT.txt
Displacement_1000steps_with_CT.txt
Displacement_1000steps_with_CT.txt
Displacement_1000steps_with_CT.txt
Zero_Displacement_1000steps_with_CT.txt
Zero_Displacement_1000steps_with_CT.txt
Zero_Displacement_1000steps_with_CT.txt
Zero_Displacement_1000steps_with_CT.txt
```

Time series of target motion is defined in each of the file listed in `target_motions_filename`.

- `time_step` is the time step/interval corresponding to the target motions.
- `hdf5_file` specifies the HDF5 file which contains the geometric information about the DRM elements and DRM nodes. This is also the file in which the DRM forces obtained from 3D wave field inversion are stored. Then this file can be used in future simulations as the DRM input. See previous DRM-related DSLs for the format of this file.

205.3.6 Output Options

Real-ESSI Simulator outputs total displacements at all the nodes, as well total stress, total strain and total plastic strain at all the Gauss points of the element in each time step of each stage of loading. Real-ESSI also outputs any/all other element output in addition to the integration/Gauss point output. Generally, 3-D elements have only integration/Gauss point outputs and structure elements have only element output. The output options are reset to the default options in the beginning of each loading stage. More information about output organization is given in section 206.2.

205.3.6.1 Output Options: Enable/Disable Output

This option is used to enable or disable the outputting of results from all nodes and elements to HDF5 (.feioutput) output file.

Note:- By default output is always enabled for each loading stage.

Command to disable output is

```
disable all output;
```

Command to enable output is

```
enable all output;
```

205.3.6.2 Output Options: Enable/Disable Element Output

This option is used to enable or disable the outputting of element results from all elements to HDF5 (.feioutput) output file, per stage of loading.

Note:- By default all results from elements are output for each loading stage, so this option can be used to enable or disable output per loading stage.

Command to disable element output is

```
1 disable element output;
```

Command to enable element output is

```
1 enable element output;
```

205.3.6.3 Output Options: Enable/Disable Displacement Output

This option is used to enable or disable the displacement output at nodes to HDF5 (.feioutput) file.

Note:- By default displacement output is enabled.

Command to disable displacement output is

```
disable displacement output;
```

Command to enable displacement output is

```
enable displacement output;
```

205.3.6.4 Output Options: Enable/Disable Acceleration Output

This option is used to enable or disable the acceleration output at nodes to HDF5 (.feioutput) file.

Note:- By default acceleration output is disabled.

Command to disable acceleration output is

```
disable acceleration output;
```

Command to enable acceleration output is

```
enable acceleration output;
```

205.3.6.5 Output Options: Enable/Disable Asynchronous Output

This option is used to enable or disable the asynchronous method of writing output to HDF5 (.feioutput) file.

Note:- By default asynchronous output is disabled. Asynchronous output is an advanced output feature. Asynchronous output is suitable for I/O-bound simulation.

Command to disable asynchronous output is

```
1 disable asynchronous output;
```

Command to enable asynchronous output is

```
1 enable asynchronous output;
```

205.3.6.6 Output Options: Output Every n Steps

This option is used to output results at intervals of n time steps.

Note:- By default results are output for every time step.

Command to enable output only at n^{th} time step interval

```
output every <.> steps;
```

For example: To output only at interval of two time steps for a simulation of 100 steps. One can write

```
output every 2 steps;
```

This will only output for steps 2,4,6,... until 100th step.

205.3.6.7 Output Options: Output Support Reactions

This option is used to output reactions at constrained supports.

Note:- By default output reactions at constrained supports are disabled.

Command to enable reactions for support is

```
1 output support reactions;
```

205.4 Checking the Model

Real-ESSI provides model check capability:

```
check model ;
```

This command will cycle over all the domain components, including Nodes, Elements, Loads, Constraints, etc. and execute the `checkModel()` function for each. Each domain component writes/reports to the terminal and to the *essi.log* file, if an error is found. For example, bricks will report when the computed Jacobian is negative and other similar errors. Nodes that are not connected will be reported as well. If the diagnostic log is empty, it means that the mesh has passed all tests. Additionally, an output HDF5 file is produced that can be used to display the mesh and do further visual inspections of the model. This file will have initial conditions as outputs for element sand nodes.

Command `check model;` represents a dry run through the model that is used to check the model before a full analysis. Model check is highly recommended before initial stages of a full analysis are executed.

205.5 Constitutive Testing

Material models can be tested using constitutive drivers which exercise single material models. RealE-SSI implements two such drivers.

1. Bardet Driver. Bardet-type constraints can be used to simulate conditions such as drained or undrained triaxial testing with strain or stress control or direct shear testing with shear control.
2. Direct Strain Driver. This driver applies a given strain history (specified by the user) to a material model.

Both these drivers produce identical output: the files `Stress.feioutput` and `Strain.feioutput` which contain stress and strain tensor components at each step. Additionally, the drivers may print out material internal information to the file `Material_Output.feioutput`. For the stress and strain files, each line of these files contain the stresses and strains organized in the following manner:

`Stress.feioutput` $\rightarrow \sigma_{11}\ \sigma_{22}\ \sigma_{33}\ \sigma_{12}\ \sigma_{13}\ \sigma_{23}$.

`Strain.feioutput` $\rightarrow \epsilon_{11}\ \epsilon_{22}\ \epsilon_{33}\ \epsilon_{12}\ \epsilon_{13}\ \epsilon_{23}$.

The Bardet driver has the following format.

```
simulate constitutive testing BARDETMETHOD use material # <.>
  scale_factor = <.>
  series_file = <string>
  sigma0 = ( <F/L^2> , <F/L^2> , <F/L^2> , <F/L^2> , <F/L^2> , <F/L^2> )
  verbose_output = <.>
```

Where,

- `BARDETMETHOD` can have any one of the following values:
  - `CONSTANT_P_TRIAXIAL_LOADING_STRAIN_CONTROL`: Triaxial loading with p kept constant. In this case the input file is interpreted as strain increments in the ϵ_{11} component.
  - `DRAINED_TRIAXIAL_LOADING_STRESS_CONTROL`: Drained Triaxial loading. In this case the input file is interpreted as stress increments in the σ_{11} component.
  - `DRAINED_TRIAXIAL_LOADING_STRAIN_CONTROL`: Drained Triaxial loading. In this case the input file is interpreted as strain increments in the ϵ_{11} component.
  - `UNDRAINED_TRIAXIAL_LOADING_STRAIN_CONTROL`: Undrained Triaxial loading. In this case the input file is interpreted as strain increments in the ϵ_{11} component.
  - `UNDRAINED_TRIAXIAL_LOADING_STRESS_CONTROL`: Undrained Triaxial loading. In this case the input file is interpreted as stress increments in the σ_{11} component.

  - UNDRAINED_SIMPLE_SHEAR_LOADING_STRAIN_CONTROL: Undrained simple-shear loading. In this case the input file is interpreted as angular strain increments in the $\gamma_{12} = 2\epsilon_{12}$ component.

- scale_factor: Can be used to scale the series file arbitrarily.
- series_file: String specifying the path to the file containing the increments (might be interpreted as strain or stress depending on the method chosen). Each line of the file contains one increment.
- sigma0: Components of the initial stress for the material, given in the order: (σ_{11}, σ_{22}, σ_{33}, σ_{12}, σ_{13}, σ_{23}).
- verbose_output(=N) Whether the driver should print extra information about the material model every N steps. If Takes value 0 (no output) or N (do output every N increments). Each material implements its own output, so the format of the Material_Output.feioutput file is variable and material dependent.

The direct strain driver has the following format.

```
simulate constitutive testing DIRECT_STRAIN use material # <.>
  scale_factor = <.>
  series_file = <string>
  sigma0 = ( <F/L^2> , <F/L^2> , <F/L^2> , <F/L^2> , <F/L^2> , <F/L^2> )
  verbose_output = <.>
```

Where al the arguments are the same as the Bardet driver. In this case each line of the file contains all six components of the strain increment to be applied. For example:
series_file = "increments.txt" where each line in increments.txt contains $\mathrm{d}\epsilon_{11}$ $\mathrm{d}\epsilon_{22}$ $\mathrm{d}\epsilon_{33}$ $\mathrm{d}\epsilon_{12}$ $\mathrm{d}\epsilon_{13}$ $\mathrm{d}\epsilon_{23}$.

205.6 List of Available Commands (tentative, not up to date)

```
add acceleration field # <.> ax = <accel> ay = <accel> az = <aaccel> ;
add constraint equal_dof with master node # <.> and slave node # <.> dof to constrain <.>;
add constraint equal_dof with node # <.> dof <.> master and node # <.> dof <.> slave;
add damping # <.> to element # <.>;
add damping # <.> to node # <.>;
add damping # <.> type Caughey3rd with a0 = <1/time> a1 = <time> a2 = <time^3> stiffness_to_use = <Initial_Stiffness|Current_Stiffness|Last_Committed_Stiffness>;
add damping # <.> type Caughey4th with a0 = <1/time> a1 = <time> a2 = <time^3> a3 = <time^5> stiffness_to_use = <Initial_Stiffness|Current_Stiffness|Last_Committed_Stiffness>;
add damping # <.> type Rayleigh with a0 = <1/time> a1 = <time> stiffness_to_use = <Initial_Stiffness|Current_Stiffness|Last_Committed_Stiffness>;
add domain reduction method loading # <.> hdf5_file = <string> scale_factor = <.>;
add domain reduction method loading # <.> hdf5_file = <string>;
add element # <.> type 20NodeBrick using <.> Gauss points each direction with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>) use material # <.>;
add element # <.> type 20NodeBrick with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>) use material # <.>;
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>;
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>;
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>;
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>;
add element # <.> type 27NodeBrick using <.> Gauss points each direction with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>) use material # <.>;
add element # <.> type 27NodeBrick with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>) use material # <.>;
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>;
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>;
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>;
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>;
add element # <.> type 3NodeShell_ANDES with nodes (<.>, <.>, <.>) use material # <.> thickness = <l> ;
add element # <.> type 4NodeShell_ANDES with nodes (<.>, <.>, <.>, <.>) use material # <.> thickness = <l> ;
add element # <.> type 4NodeShell_MITC4 with nodes (<.>, <.>, <.>, <.>) use material # <.> thickness = <L>;
add element # <.> type 4NodeShell_NewMITC4 with nodes (<.>, <.>, <.>, <.>) use material # <.> thickness = <L>;
add element # <.> type 8_27_NodeBrick using <.> Gauss points each direction with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>) use material # <.>;
```

```
28 add element # <.> type 8_27_NodeBrick with nodes (<.>, <.>, <.>, <.>, <.>, <.>, ↩
       <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, ↩
       <.>, <.>, <.>, <.>, <.>, <.>) use material # <.>;
29 add element # <.> type 8_27_NodeBrick_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>;
30 add element # <.> type 8_27_NodeBrick_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>;
31 add element # <.> type 8_27_NodeBrick_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>;
32 add element # <.> type 8_27_NodeBrick_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>;
33 add element # <.> type 8NodeBrick using <.> Gauss points each direction with ↩
       nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>) use material # <.>;
34 add element # <.> type 8NodeBrick with nodes (<.>, <.>, <.>, <.>, <.>, <.>, ↩
       <.>, <.>) use material # <.>;
35 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>;
36 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>;
37 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>;
38 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>;
add element # <.> type beam_9dof_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> );
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> );
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> );
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> );
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> );
add element # <.> type HardContact with nodes (<.>, <.>) axial_stiffness = <F/L> shear_stiffness = <F/L> normal_damping = <F/L> tangential_damping = <F/L> friction_ratio = <.> contact_plane_vector = (<.>, <.>, <.> );
add element # <.> type HardWetContact with nodes (<.>, <.>) axial_stiffness = <F/L> shear_stiffness = <F/L> normal_damping = <F/L> tangential_damping = <F/L> friction_ratio = <.> contact_plane_vector = (<.>, <.>, <.> );
add element # <.> type ShearBeam with nodes (<.>, <.>) cross_section = <l^2> use material # <.>;
add element # <.> type SoftContact with nodes (<.>, <.>) initial_axial_stiffness = <F/L> stiffening_rate = <m^-1> shear_stiffness = <F/L> normal_damping = <F/L> tangential_damping = <F/L> friction_ratio = <.> contact_plane_vector = (<.>, <.>, <.> );
add element # <.> type SoftWetContact with nodes (<.>, <.>) initial_axial_stiffness = <F/L> stiffening_rate = <m^-1> shear_stiffness = <F/L> normal_damping = <F/L> tangential_damping = <F/L> friction_ratio = <.> contact_plane_vector = (<.>, <.>, <.> );
add element # <.> type truss with nodes (<.>, <.>) use material # <.> cross_section = <length^2> mass_density = <M/L^3> ;
add element # <.> type variable_node_brick_8_to_27 using <.> Gauss points each direction with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>,
```

```
    <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, ↩
    <.>, <.>) use material # <.>;
add elements (<.>) to physical_element_group "string";
add fiber # <.> using material # <.> to section # <.> fiber_cross_section = ↩
    <area> fiber_location = (<L>,<L>);
add imposed motion # <.> to node # <.> dof DOFTYPE displacement_scale_unit = ↩
    <displacement> displacement_file = "disp_filename" velocity_scale_unit = ↩
    <velocity> velocity_file = "vel_filename" acceleration_scale_unit = ↩
    <acceleration> acceleration_file = "acc_filename";
add imposed motion # <.> to node # <.> dof DOFTYPE time_step = <t> ↩
    displacement_scale_unit = <length> displacement_file = "disp_filename" ↩
    velocity_scale_unit = <velocity> velocity_file = "vel_filename" ↩
    acceleration_scale_unit = <acceleration> acceleration_file = "acc_filename";
add load # <.> to all elements type self_weight use acceleration field # <.>;
add load # <.> to element # <.> type self_weight use acceleration field # <.>;
add load # <.> to element # <.> type surface at nodes (<.> , <.> , <.> , <.>) ↩
    with magnitude <.>;
add load # <.> to element # <.> type surface at nodes (<.> , <.> , <.> , <.>) ↩
    with magnitudes (<.> , <.> , <.> , <.>);
add load # <.> to element # <.> type surface at nodes (<.> , <.> , <.> , <.>, ↩
    <.>, <.>, <.>, <.>) with magnitude <.>;
add load # <.> to element # <.> type surface at nodes (<.> , <.> , <.> , <.>, ↩
    <.>, <.>, <.>, <.>) with magnitudes (<.> , <.> , <.> , <.>, <.>, <.>, <.>, ↩
    <.>);
add load # <.> to element # <.> type surface at nodes (<.> , <.> , <.> , <.>, ↩
    <.>, <.>, <.>, <.>, <.>) with magnitude <.>;
add load # <.> to element # <.> type surface at nodes (<.> , <.> , <.> , <.>, ↩
    <.>, <.>, <.>, <.>, <.>) with magnitudes (<.> , <.> , <.> , <.>, <.>, <.>, ↩
    <.>, <.>, <.>);
add load # <.> to node # <.> type from_reactions;
add load # <.> to node # <.> type linear FORCETYPE = <force or moment>; ↩
    //FORCETYPE = Fx Fy Fz Mx My Mz F_fluid_x F_fluid_y F_fluid_z
add load # <.> to node # <.> type path_series FORCETYPE = <force or moment> ↩
    time_step = <time> series_file = "filename";
add load # <.> to node # <.> type path_time_series FORCETYPE = <force or ↩
    moment> series_file = "filename";
add load # <.> to node # <.> type self_weight use acceleration field # <.>;
add mass to node # <.> mx = <mass> my = <mass> mz = <mass> Imx = ↩
    <mass*length^2> Imy = <mass*length^2> Imz = <mass*length^2>;
add mass to node # <.> mx = <mass> my = <mass> mz = <mass>;
add material # <.> type CamClay mass_density = <M/L^3> M = <.> lambda = <.> ↩
    kappa = <.> e0 = <.> p0 = <F/L^2> Poisson_ratio = <.> ↩
    initial_confining_stress = <F/L^2>
```

```
71 add material # <.> type DruckerPrager mass_density = <M/L^3> elastic_modulus = <F/L^2> poisson_ratio = <.> druckerprager_k = <> kinematic_hardening_rate = <F/L^2> isotropic_hardening_rate = <F/L^2> initial_confining_stress = exp;
72 add material # <.> type DruckerPragerArmstrongFrederickLE mass_density = <M/L^3> elastic_modulus = <F/L^2> poisson_ratio = <.> druckerprager_k = <> armstrong_frederick_ha = <F/L^2> armstrong_frederick_cr = <F/L^2> isotropic_hardening_rate = <F/L^2> initial_confining_stress = <F/L^2>;
73 add material # <.> type DruckerPragerArmstrongFrederickNE mass_density = <M/L^3> DuncanChang_K = <.> DuncanChang_pa = <F/L^2> DuncanChang_n = <.> DuncanChang_sigma3_max = <F/L^2> DuncanChang_nu = <.> druckerprager_k = <> armstrong_frederick_ha = <F/L^2> armstrong_frederick_cr = <F/L^2> isotropic_hardening_rate = <F/L^2> initial_confining_stress = <F/L^2>;
74 add material # <.> type DruckerPragerNonAssociateArmstrongFrederick mass_density = <M/L^3> elastic_modulus = <F/L^2> poisson_ratio = <.> druckerprager_k = <> armstrong_frederick_ha = <F/L^2> armstrong_frederick_cr = <F/L^2> isotropic_hardening_rate = <F/L^2> initial_confining_stress = <F/L^2> plastic_flow_xi = <> plastic_flow_kd = <> ;
75 add material # <.> type DruckerPragerNonAssociateLinearHardening mass_density = <M/L^3> elastic_modulus = <F/L^2> poisson_ratio = <.> druckerprager_k = <> kinematic_hardening_rate = <F/L^2> isotropic_hardening_rate = <F/L^2> initial_confining_stress = <F/L^2> plastic_flow_xi = <> plastic_flow_kd = <> ;
76 add material # <.> type DruckerPragervonMises mass_density = <M/L^3> elastic_modulus = <F/L^2> poisson_ratio = <.> druckerprager_k = <> kinematic_hardening_rate = <F/L^2> isotropic_hardening_rate = <F/L^2> initial_confining_stress = exp;
77 add material # <.> type linear_elastic_crossanisotropic mass_density = <mass_density> elastic_modulus_horizontal = <F/L^2> elastic_modulus_vertical = <F/L^2> poisson_ratio_h_v = <.> poisson_ratio_h_h = <.> shear_modulus_h_v = <F/L^2>;
78 add material # <.> type linear_elastic_isotropic_3d mass_density = <M/L^3> elastic_modulus = <F/L^2> poisson_ratio = <.>;
79 add material # <.> type linear_elastic_isotropic_3d_LT mass_density = <M/L^3> elastic_modulus = <F/L^2> poisson_ratio = <.>;
80 add material # <.> type roundedMohrCoulomb mass_density = <M/L^3> elastic_modulus = <F/L^2> poisson_ratio = <.> RMC_m = <.> RMC_qa = <F/L^2> RMC_pc = <F/L^2> RMC_e = <.> RMC_eta0 = <.> RMC_Heta = <F/L^2> initial_confining_stress = <F/L^2>
81 add material # <.> type sanisand2004 mass_density = <M/L^3> e0 = <.> sanisand2004_G0 = <.> poisson_ratio = <.> sanisand2004_Pat = <stress> sanisand2004_p_cut = <.> sanisand2004_Mc = <.> sanisand2004_c = <.> sanisand2004_lambda_c = <.> sanisand2004_xi = <.> sanisand2004_ec_ref = <.> sanisand2004_m = <.> sanisand2004_h0 = <.> sanisand2004_ch = <.>
```

```
    sanisand2004_nb = <.> sanisand2004_A0 = <.> sanisand2004_nd = <.> ↩
    sanisand2004_z_max = <.> sanisand2004_cz = <.> initial_confining_stress = ↩
    <stress> ;
82  add material # <.> type sanisand2004_legacy mass_density = <M/L^3> e0 = <.> ↩
    sanisand2004_G0 = <.> poisson_ratio = <.> sanisand2004_Pat = <stress> ↩
    sanisand2004_p_cut = <.> sanisand2004_Mc = <.> sanisand2004_c = <.> ↩
    sanisand2004_lambda_c = <.> sanisand2004_xi = <.> sanisand2004_ec_ref = <.> ↩
    sanisand2004_m = <.> sanisand2004_h0 = <.> sanisand2004_ch = <.> ↩
    sanisand2004_nb = <.> sanisand2004_A0 = <.> sanisand2004_nd = <.> ↩
    sanisand2004_z_max = <.> sanisand2004_cz = <.> initial_confining_stress = ↩
    <stress> algorithm = <explicit|implicit> number_of_subincrements = <.> ↩
    maximum_number_of_iterations = <.> tolerance_1 = <.> tolerance_2 = <.>;
83  add material # <.> type sanisand2008 mass_density = <M/L^3> e0 = <.> ↩
    sanisand2008_G0 = <.> sanisand2008_K0 = <.> sanisand2008_Pat = <stress> ↩
    sanisand2008_k_c = <.> sanisand2008_alpha_cc = <.> sanisand2008_c = <.> ↩
    sanisand2008_xi = <.> sanisand2008_lambda = <.> sanisand2008_ec_ref = <.> ↩
    sanisand2008_m = <.> sanisand2008_h0 = <.> sanisand2008_ch = <.> ↩
    sanisand2008_nb = <.> sanisand2008_A0 = <.> sanisand2008_nd = <.> ↩
    sanisand2008_p_r = <.> sanisand2008_rho_c = <.> sanisand2008_theta_c = <.> ↩
    sanisand2008_X = <.> sanisand2008_z_max = <.> sanisand2008_cz = <.> ↩
    sanisand2008_p0 = <stress> sanisand2008_p_in = <.> algorithm = ↩
    <explicit|implicit> number_of_subincrements = <.> ↩
    maximum_number_of_iterations = <.> tolerance_1 = <.> tolerance_2 = <.>;
84  add material # <.> type uniaxial_concrete02 compressive_strength = <F/L^2> ↩
    strain_at_compressive_strength = <.> crushing_strength = <F/L^2> ↩
    strain_at_crushing_strength = <.> lambda = <.> tensile_strength = <F/L^2> ↩
    tension_softening_stiffness = <F/L^2>;
85  add material # <.> type uniaxial_elastic elastic_modulus = <F/L^2> ↩
    viscoelastic_modulus = <mass / length / time> ;
86  add material # <.> type uniaxial_steel01 yield_strength = <F/L^2> ↩
    elastic_modulus = <F/L^2> strain_hardening_ratio = <.> a1 = <.> a2 = <.> a3 ↩
    = <> a4 = <.> ;
87  add material # <.> type uniaxial_steel02 yield_strength = <F/L^2> ↩
    elastic_modulus = <F/L^2> strain_hardening_ratio = <.> R0 = <.> cR1 = <.> ↩
    cR2 = <.> a1 = <.> a2 = <.> a3 = <> a4 = <.> ;
88  add material # <.> type vonMises mass_density = <M/L^3> elastic_modulus = ↩
    <F/L^2> poisson_ratio = <.> von_mises_radius = <F/L^2> ↩
    kinematic_hardening_rate = <F/L^2> isotropic_hardening_rate = <F/L^2> ;
89  add material # <.> type vonMisesArmstrongFrederick mass_density = <M/L^3> ↩
    elastic_modulus = <F/L^2> poisson_ratio = <.> von_mises_radius = <> ↩
    armstrong_frederick_ha = <F/L^2> armstrong_frederick_cr = <F/L^2> ↩
    isotropic_hardening_rate = <F/L^2> ;
90  add node # <.> at (<length>,<length>,<length>) with <.> dofs;
91  add nodes (<.>) to physical_node_group "string";
```

```
add section # <.> type elastic3d elastic_modulus = <F/L^2> cross_section = <L^2> bending_Iz = <L^4> bending_Iy=<L^4> torsion_Jx=<L^4> ;
add section # <.> type Elastic_Membrane_Plate elastic_modulus = <F/L^2> poisson_ratio = <.> thickness = <length> mass_density = <M/L^3>;
add section # <.> type FiberSection TorsionConstant_GJ = <F*L^2>
add section # <.> type Membrane_Plate_Fiber thickness = <length> use material # <.>;
add single point constraint to node # <.> dof to constrain <dof_type> constraint value of <corresponding unit>;
add uniform acceleration # <.> to all nodes dof <.> time_step = <T> scale_factor = <.> initial_velocity = <L/S> acceleration_file = <string>;
check mesh filename;
compute reaction forces;
define algorithm With_no_convergence_check / Newton / Modified_Newton;
define convergence test Norm_Displacement_Increment / Energy_Increment / Norm_Unbalance / Relative_Norm_Displacement_Increment / Relative_Energy_Increment / Relative_Norm_Unbalance tolerance = <.> maximum_iterations = <.> verbose_level = <0>|<1>|<2>;
define dynamic integrator Hilber_Hughes_Taylor with alpha = <.>;
define dynamic integrator Newmark with gamma = <.> beta = <.>;
define load factor increment <.>;
define NDMaterial constitutive integration algorithm Forward_Euler;
define NDMaterial constitutive integration algorithm Forward_Euler_Subincrement number_of_subincrements =<.>;
define NDMaterial constitutive integration algorithm Forward_Euler|Forward_Euler_Subincrement|Backward_Euler|Backward_Euler_Subincrement| yield_function_relative_tolerance = <.> stress_relative_tolerance = <.> maximum_iterations = <.>;
define physical_element_group "string";
define physical_node_group "string";
define solver ProfileSPD / UMFPack;
define static integrator displacement_control using node # <.> dof DOFTYPE increment <length>;
disable asynchronous output;
disable element output;
disable output;
enable asynchronous output;
enable element output;
enable output;
fix node # <.> dofs <.>;
fix node # <.> dofs all;
free node # <.> dofs <.>;
help;
if (.) { } else {};
```

```
123 if (.) { };
124 model name "name_string";
125 new loading stage "name_string";
126 output every <.> steps;
127 output non_converged_iterations;
128 output support reactions;
129 print <.>;
130 print element # <.>;
131 print node # <.>;
132 print physical_element_group "string";
133 print physical_node_group "string";
134 remove constraint equal_dof node # <.>;
135 remove displacement from node # <.>;
136 remove element # <.>;
137 remove imposed motion # <.>;
138 remove load # <.>;
139 remove node # <.>;
140 remove physical_node_group "string";
141 remove strain from element # <.>;
142 remove physical_element_group "string";
143 runTest;
144 set output compression level to <.>;
145 simulate <.> steps using static algorithm;
146 simulate <.> steps using transient algorithm time_step = <time>;
147 simulate <.> steps using variable transient algorithm time_step = <time> ↩
    minimum_time_step = <time> maximum_time_step = <time> number_of_iterations ↩
    = <.>;
148 simulate constitutive testing BARDETMETHOD use material # <.> scale_factor = ↩
    <.> series_file = <string> sigma0 = ( <F/L^2> , <F/L^2> , <F/L^2> , <F/L^2> ↩
    , <F/L^2> , <F/L^2> ) verbose_output = <.>
149 simulate constitutive testing constant mean pressure triaxial strain control ↩
    use material # <.> strain_increment_size = <.> maximum_strain = <.> ↩
    number_of_times_reaching_maximum_strain = <.>;
150 simulate constitutive testing DIRECT_STRAIN use material # <.> scale_factor = ↩
    <.> series_file = <string> sigma0 = ( <F/L^2> , <F/L^2> , <F/L^2> , <F/L^2> ↩
    , <F/L^2> , <F/L^2> ) verbose_output = <.>
151 simulate constitutive testing drained triaxial strain control use material # ↩
    <.> strain_increment_size = <.> maximum_strain = <.> ↩
    number_of_times_reaching_maximum_strain = <.>;
152 simulate constitutive testing undrained simple shear use material # <.> ↩
    strain_increment_size = <.> maximum_strain = <.> ↩
    number_of_times_reaching_maximum_strain = <.>;
153 simulate constitutive testing undrained triaxial stress control use material # ↩
    <.> strain_increment_size = <.> maximum_strain = <.> ↩
```

```
   number_of_times_reaching_maximum_strain = <.>;
simulate constitutive testing undrained triaxial use material # <.> ↩
   strain_increment_size = <.> maximum_strain = <.> ↩
   number_of_times_reaching_maximum_strain = <.>;
simulate using eigen algorithm number_of_modes = <.>;
ux uy uz Ux Uy Uz rx ry rz;
while (.) { };
whos;
```

205.7 List of reserved keywords

The following keywords are reserved and cannot be used as variables in a script or interactive session. Doing so would result in a syntax error.

First Order (commands)

```
a0
a1
a2
a3
a4
acceleration
acceleration_depth
acceleration_file
acceleration_filename
acceleration_scale_unit
add
algorithm
algorithm
all
all
allowed_subincrement_strain
alpha
alpha1
alpha2
and
angle
armstrong_frederick_cr
armstrong_frederick_ha
asynchronous
at
ax
axial_penalty_stiffness
axial_stiffness
axial_viscous_damping
ay
az
bending_Iy
bending_Iz
beta
Beta
beta_min
case
cases
characteristic_strength
check
chi
cohesion
combine
compression
```

```
compressive_strength
compressive_yield_strength
compute
confinement
confinement_strain
constitutive
constrain
constraint
contact_plane_vector
control
convergence
cR1
cR2
cross_section
crushing_strength
Current_Stiffness
cyclic
damage_parameter_An
damage_parameter_Ap
damage_parameter_Bn
damping
define
depth
dilatancy_angle
dilation_angle_eta
dilation_scale
direction
disable
displacement
displacement_file
displacement_scale_unit
dof
dofs
dofs
domain
druckerprager_k
DuncanChang_K
DuncanChang_n
DuncanChang_nu
DuncanChang_pa
DuncanChang_sigma3_max
DYNAMIC_DOMAIN_PARTITION
e0
each
elastic
elastic_modulus
elastic_modulus_horizontal
elastic_modulus_vertical
element
elements
else
```

```
enable
every
factor
fiber
fiber_cross_section
fiber_location
field
file
fix
fluid
free
friction_angle
friction_ratio
from
gamma
Gamma
Gauss
generate
GoverGmax
h_in
hardening_parameters_of_yield_surfaces
hardening_parameters_scale_unit
hdf5_file
hdf5_filenames_list
help
if
imposed
Imx
Imy
Imz
in
inclined
increment
initial_axial_stiffness
initial_confining_stress
initial_elastic_modulus
initial_shear_modulus
initial_shear_stiffness
initial_velocity
integration
integration_points
IntegrationRule
integrator
interface
isotropic_hardening_rate
joint_1_offset
joint_2_offset
K_f
K_s
k_x
k_y
```

```
k_z
kappa
kd_in
kinematic_hardening_rate
lambda
level
line_search_beta
line_search_eta
line_search_max_iter
liquefaction_Alpha
liquefaction_c_h0
liquefaction_Dir
liquefaction_dre1
liquefaction_Dre2
liquefaction_EXPN
liquefaction_gamar
liquefaction_mdc
liquefaction_mfc
liquefaction_pa
liquefaction_pmin
load
load_factors_list
loading
local_y_vector
local_z_vector
M
M_in
magnitude
magnitudes
mass
mass_density
master
material
max_axial_stiffness
maximum_iterations
maximum_number_of_iterations
maximum_strain
maximum_stress
maximum_time_step
method
minimal
minimum_time_step
model
model
moment_x_stiffness
moment_y_stiffness
monotonic
motion
mu
mx
my
```

```
mz
name
NDMaterial
new
newton_with_subincrement
node
nodes
number_of_cycles
number_of_files
number_of_increment
number_of_integration_points
number_of_iterations
number_of_layers
number_of_modes
number_of_subincrements
number_of_times_reaching_maximum_strain
of
output
output
output_filename
p0
parallel
peak_friction_coefficient_limit
peak_friction_coefficient_rate_of_decrease
penalty_stiffness
pi1
pi2
pi3
plastic_deformation_rate
plastic_flow_kd
plastic_flow_xi
plot
point
points
poisson_ratio
poisson_ratio_h_h
poisson_ratio_h_v
porosity
print
propagation
pure
R0
radiuses_of_yield_surface
radiuses_scale_unit
rate_of_softening
reduction
reference_pressure
remove
rempve
residual_friction_coefficient
restart
```

```
restart_files
results
rho_a
rho_f
rho_s
rho_w
RMC_e
RMC_eta0
RMC_Heta
RMC_m
RMC_pc
RMC_qa
RMC_shape_k
rounded_distance
runTest
sanisand2004_A0
sanisand2004_c
sanisand2004_ch
sanisand2004_cz
sanisand2004_ec_ref
sanisand2004_G0
sanisand2004_h0
sanisand2004_lambda_c
sanisand2004_m
sanisand2004_Mc
sanisand2004_nb
sanisand2004_nd
sanisand2004_p_cut
sanisand2004_Pat
sanisand2004_xi
sanisand2004_z_max
sanisand2008_A0
sanisand2008_alpha_cc
sanisand2008_c
sanisand2008_ch
sanisand2008_cz
sanisand2008_ec_ref
sanisand2008_G0
sanisand2008_h0
sanisand2008_K0
sanisand2008_k_c
sanisand2008_lambda
sanisand2008_m
sanisand2008_nb
sanisand2008_nd
sanisand2008_p0
sanisand2008_p_in
sanisand2008_p_r
sanisand2008_Pat
sanisand2008_rho_c
sanisand2008_theta_c
```

```
sanisand2008_X
sanisand2008_xi
sanisand2008_z_max
save
scale_factor
SCOTCHGRAPHPARTITIONER
section
section_number
sequential
series_file
set
shear
shear_length_ratio
shear_modulus
shear_modulus_h_v
shear_stiffness
shear_viscous_damping
shear_zone_thickness
ShearStrainGamma
sigma0
simulate
single
size_of_peak_plateau
sizes_of_yield_surfaces
slave
slave
soil
soil_profile_filename
soil_surface
solid
solver
stage
steps
steps
stiffening_rate
stiffness_to_use
strain
strain_at_compressive_strength
strain_at_crushing_strength
strain_hardening_ratio
strain_increment_size
stress
stress_increment_size
stress_relative_tolerance
sub-stepping
surface
surface_vector_relative_tolerance
tensile_strength
tensile_yield_strength
tension_softening_stiffness
test
```

```
test
testing
thickness
time_step
to
tolerance_1
tolerance_2
torsion_Jx
torsional_stiffness
TorsionConstant_GJ
total_number_of_shear_modulus
total_number_of_yield_surface
triaxial
type
uniaxial
uniaxial_material
uniform
unit_of_acceleration
unit_of_damping
unit_of_rho
unit_of_vs
use
using
value
velocity_file
velocity_scale_unit
verbose_output
viscoelastic_modulus
von_mises_radius
wave
wave1c
wave3c
while
whos
with
xi_in
xz_plane_vector
yield_function_relative_tolerance
yield_strength
yield_surface_scale_unit
x
y
z
```

Second Order (inside commands)

```
20NodeBrick
20NodeBrick_up
20NodeBrick_upU
27NodeBrick
27NodeBrick_up
```

```
27NodeBrick_upU
3NodeShell_ANDES
4NodeShell_ANDES
4NodeShell_MITC4
4NodeShell_NewMITC4
8_27_NodeBrick
8_27_NodeBrick_up
8_27_NodeBrick_upU
8NodeBrick
8NodeBrick_fluid_incompressible_up
8NodeBrick_up
8NodeBrick_upU
Absolute_Norm_Displacement_Increment
Absolute_Norm_Unbalanced_Force
arclength_control
Average_Norm_Displacement_Increment
Average_Norm_Unbalanced_Force
Backward_Euler
BARDETMETHOD
beam_9dof_elastic
beam_displacement_based
beam_elastic
beam_elastic_lumped_mass
BeamColumnDispFiber3d
BearingElastomericPlasticity3d
BFGS
BondedContact
CamClay
Caughey3rd
Caughey4th
constant mean pressure triaxial strain control
Cosserat8NodeBrick
Cosserat_linear_elastic_isotropic_3d
Cosserat_von_Mises
DIRECT_STRAIN
displacement_control
DOFTYPE
domain reduction method
drained triaxial strain control
DruckerPrager
DruckerPragerArmstrongFrederickLE
DruckerPragerArmstrongFrederickNE
DruckerPragerMultipleYieldSurface
DruckerPragerMultipleYieldSurfaceGoverGmax
DruckerPragerNonAssociateArmstrongFrederick
DruckerPragerNonAssociateLinearHardening
DruckerPragervonMises
dynamic
eigen
elastic3d
Elastic_Membrane_Plate
```

```
ElasticFourNodeQuad
Energy_Increment
equal_dof
F_fluid_x
F_fluid_y
F_fluid_z
FiberSection
ForceBasedCoupledHardContact
ForceBasedCoupledSoftContact
ForceBasedElasticContact
ForceBasedHardContact
ForceBasedSoftContact
FORCETYPE
Forward_Euler
Forward_Euler_Subincrement
from_reactions
Fx
Fy
Fz
Hilber_Hughes_Taylor
HyperbolicDruckerPragerArmstrongFrederick
HyperbolicDruckerPragerLinearHardening
HyperbolicDruckerPragerNonAssociateArmstrongFrederick
HyperbolicDruckerPragerNonAssociateLinearHardening
linear
linear_elastic_crossanisotropic
linear_elastic_isotropic_3d
linear_elastic_isotropic_3d_LT
Membrane_Plate_Fiber
Modified_Newton
Mx
My
Mz
Newmark
Newton
non_converged_iterations
NonlinearFourNodeQuad
Norm_Displacement_Increment
Norm_Unbalance
Parallel
path_series
path_time_series
petsc
petsc_options_string
physical_element_group
physical_node_group
Pisano
PlaneStressLayeredMaterial
PlaneStressRebarMaterial
PlasticDamageConcretePlaneStress
pressure
```

```
ProfileSPD
Rayleigh
reaction forces
reactions
Relative_Energy_Increment
Relative_Norm_Displacement_Increment
Relative_Norm_Unbalance
Relative_Norm_Unbalanced_Force
roundedMohrCoulomb
RoundedMohrCoulombMultipleYieldSurface
sanisand2004
sanisand2004_legacy
sanisand2008
self_weight
ShearBeam
solid fluid interaction transient
static
StressBasedCoupledHardContact_ElPPlShear
StressBasedCoupledHardContact_NonLinHardShear
StressBasedCoupledHardContact_NonLinHardSoftShear
StressBasedCoupledSoftContact
StressBasedCoupledSoftContact_ElPPlShear
StressBasedCoupledSoftContact_NonLinHardShear
StressBasedCoupledSoftContact_NonLinHardSoftShear
StressBasedHardContact_ElPPlShear
StressBasedHardContact_NonLinHardShear
StressBasedHardContact_NonLinHardSoftShear
StressBasedSoftContact_ElPPlShear
StressBasedSoftContact_NonLinHardShear
StressBasedSoftContact_NonLinHardSoftShear
SuperElementLinearElasticImport
support
surface
transient
truss
TsinghuaLiquefactionModelCirclePiPlane
TsinghuaLiquefactionModelNonCirclePiPlane
UMFPack
undrained simple shear
undrained triaxial
undrained triaxial stress control
uniaxial_concrete02
uniaxial_elastic
uniaxial_steel01
uniaxial_steel02
variable transient
variable_node_brick_8_to_27
vonMises
vonMisesArmstrongFrederick
vonMisesMultipleYieldSurface
vonMisesMultipleYieldSurfaceGoverGmax
```

```
With_no_convergence_check

beta
gamma
delta

ux
uy
uz
rx
ry
rz
Ux
Uy
Uz
p
M
m
kg
s
cm
mm
km
Hz
Minute
Hour
Day
Week
ms
ns
N
kN
Pa
kPa
MPa
GPa
pound
lbm
lbf
inch
in
feet
ft
yard
mile
psi
ksi
kip
g
pi

```

```
NUMBER_OF_NODES
NUMBER_OF_ELEMENTS
CURRENT_TIME
NUMBER_OF_SP_CONSTRAINTS
NUMBER_OF_MP_CONSTRAINTS
NUMBER_OF_LOADS
IS_PARALLEL
SIMULATE_EXIT_FLAG
then
while
do
let
vector

cos
sin
tan
cosh
sinh
tanh
acos
asin
atan
atan2
sqrt
exp
log10
ceil
fabs
floor
log
```

205.8 Integrated Development Environment (IDE) for DSL

205.9 Mesh Generation using GiD

1. Download the latest version of GiD from http://www.gidhome.com/, and also get a temporary license (or purchase it...).
2. Download essi.gid.tar.gz, unpack it (`tar -xvzf essi.gid.tar.gz`) in `problemtypes` directory that is located in GiD's root directory.
3. When you run GiD, you will see `essi` in "Data > Problem types", and can start using it...
4. A simple movie with instructions for mesh generation is available: (Link to a movie, 11MB).

205.10 Model Development and Mesh Generation using gmesh

205.11 Model Input File Editing using Sublime

http://www.sublimetext.com/

Chapter 206

Output Formats

(2012-2014-2017-2019-2021-)

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

206.1 Chapter Summary and Highlights

206.2 Introduction

All output from ESSI simulator is stored inside a database format, specifically designed for handling scientific array-oriented data, called HDF5 (Group, 2020). HDF stands for 'Hierarchical Data Format' and is a self-describing data format suitable for portable sharing of scientific data. The format was created and is maintained by the HDF group (http://www.hdfgroup.org/)

Data is stored within the file using a hierarchy similar to a unix filesystem, with groups to store related data and the actual data stored in so-called 'datasets' within each group.

HDF5 was chosen because it meets our design goals of provides:

- A simplified output format. Output is a single HDF5 file per analysis stage.
- An efficient binary (possibly compressed) file format that optimizes random access to data.
- A data format that is amenable to store output from parallel computations.
- Has a reasonable API exposed in several languages so that users can easily and customizably access simulation data.

One very convenient tool for the basic exploration of HDF5 files is the viewer 'hdfview' (http://www.hdfgroup.org/products/java/hdfview/index.html).

206.3 Output Filename and Format

On running any simulation on Real-ESSI simulator output files are produced for each analysis stage. The number of outputs and the filename is slightly different for sequential and parallel runs. Each output file, contains the information about the model mesh, nodal displacements, elements output, boundary conditions, material tags.. etc. The output files are designed as completely independent files containing all the data for the loading stage. In parallel each of the follower compute process outputs contains all the data corresponding to only the follower compute process. This is done to make the visualization and output process efficient.

206.3.1 Sequential

For sequential runs a single output file is produced per analysis stage. The files are named according to `model` and `stage names`, not by the filename that runs the analysis. The extension is set to be '`.h5.feioutput`', to distinguish from future possible alternative output formats.

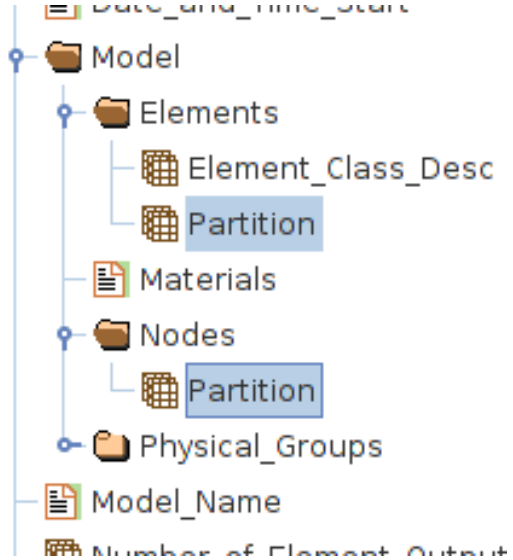


Figure 206.1: Partition info in output file produced by main compute node in parallel run

For example, if the model name is '`site_response`' and the stage name is '`earthquake_shaking`' the corresponding output filename will be '`site_response_earthquake_shaking.h5.feioutput`'.

206.3.2 Parallel

In parallel, for each stage, output files produced are equal to the number of CPU's used. For example, a simulation run of 8 CPU's will produce 8 output files per stage for each corresponding CPU's (cores).The filename remains the same as sequential output each CPU (process id) used, but the extension is set to be as '`.h5.pid.feioutput`', where *pid* refers to the process id of the CPU. However, the main compute process having *pid* equal to 0 follows the extension '`.h5.feioutput`'.

For example, In parallel, if the model name is '`site_response`' and the stage name is '`earthquake_shaking`' and the analysis is run on n CPU's, the corresponding output filename for the main compute process ($pid = 0$) would be '`site_response_earthquake_shaking.h5.feioutput`'. All the follower compute process having $pid > 0$ would have output filename '`site_response_earthquake_shaking.h5.pid.feioutput`'.

The main compute process usually does not contain any nodes and element once the partition is achieved and nodes and elements are transfered to their respective CPU's or cores. Thus, the output produced by main compute process does not contain any mesh or output results. However, it contains the partition data as shown in Figure 206.1 describing the process id on which any node or element is assigned. This information is quite useful, during post processing when the result of a particular node or element needs to be extracted. The main compute output can be read to find out the follower compute process id on which the data is located and then the output of that process id can be read to get the

data of interest.

206.4 Output Units

Real-ESSI `.feioutput` file stores all the results in standard units. The table below shows the units of all the data stored in HDF5 file.

Quantity	Unit
Force F_x, F_y, F_z	N
Moments M_x, M_y, M_z	$N-m$
Pressure p	Pa
Displacement u_x, u_y, u_z	m
Rotation r_x, r_y, r_z	$radian$
Stress σ	Pa
Strain ϵ	unit less
Acceleration a_x, a_y, a_z	m/s^2
Time t	s

206.5 Data organization

In HDF5 jargon a multidimensional array is called a dataset. Datasets are indexed arrays (up to 32 dimensions) that can contain different types of data. Supported data types are: integers (various sizes), floating point numbers (`float`, `float`, `long double`, etc.), strings of text (fixed and variable size `char *`), and arbitrary structures of data (similar to C language `struct`). A file can contain as many independent datasets as needed. Datasets can be organized into 'groups', which are like folders in a file system. HDF5 provides additionally convenience data-types such as 'references', which provide views (slices) into diferent datasets or portions of them.

In the particular case of the ESSI HDF5 output, the files are designed with the contents and structure explained hereafter and depicted in Figure 206.3.

206.5.1 The `Root` group

The root of the HDF5 file contains information about each stage of loading. In parallel simulations, the information corresponds to the process Id (follower compute node) involved in that stage. The objects under this group are shown in Figure 206.4 and described in List 206.5.1

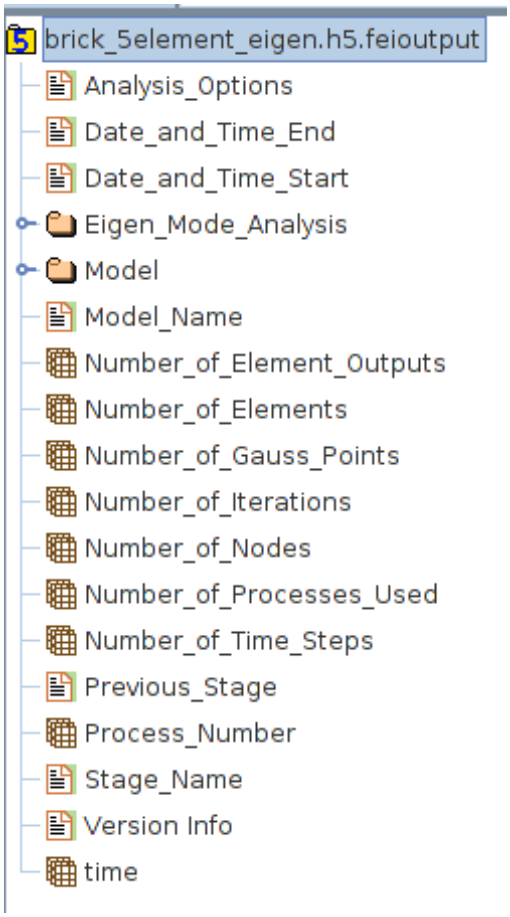


Figure 206.2: Output from a typical analysis.

- `time` : (`float`) A floating point array named which contains the available time steps for this analysis.
- `Number_of_Time_Steps`: (`int`) A single scalar integer array with the number of times steps.
- `Model_Name`: (`string`) A single string with the model name.
- `Stage_Name`: (`string`) A single string with the stage name.
- `Previous_Stage`: (`string`) A single string containing previous stage name.
- `Process_Number`: (`int`) An integer representing the *process id* by which output was generated. For sequential runs, *process id* would be zero. In parallel runs, *process id* corresponds to the *mpi rank* or *follower compute id* of processor involved in computation.
- `Number_of_Processes_Used`: (`int`) An integer representing total number of processors/CPU/nodes used in the simulation. For sequential runs, it is equal to one whereas for parallel runs, it is

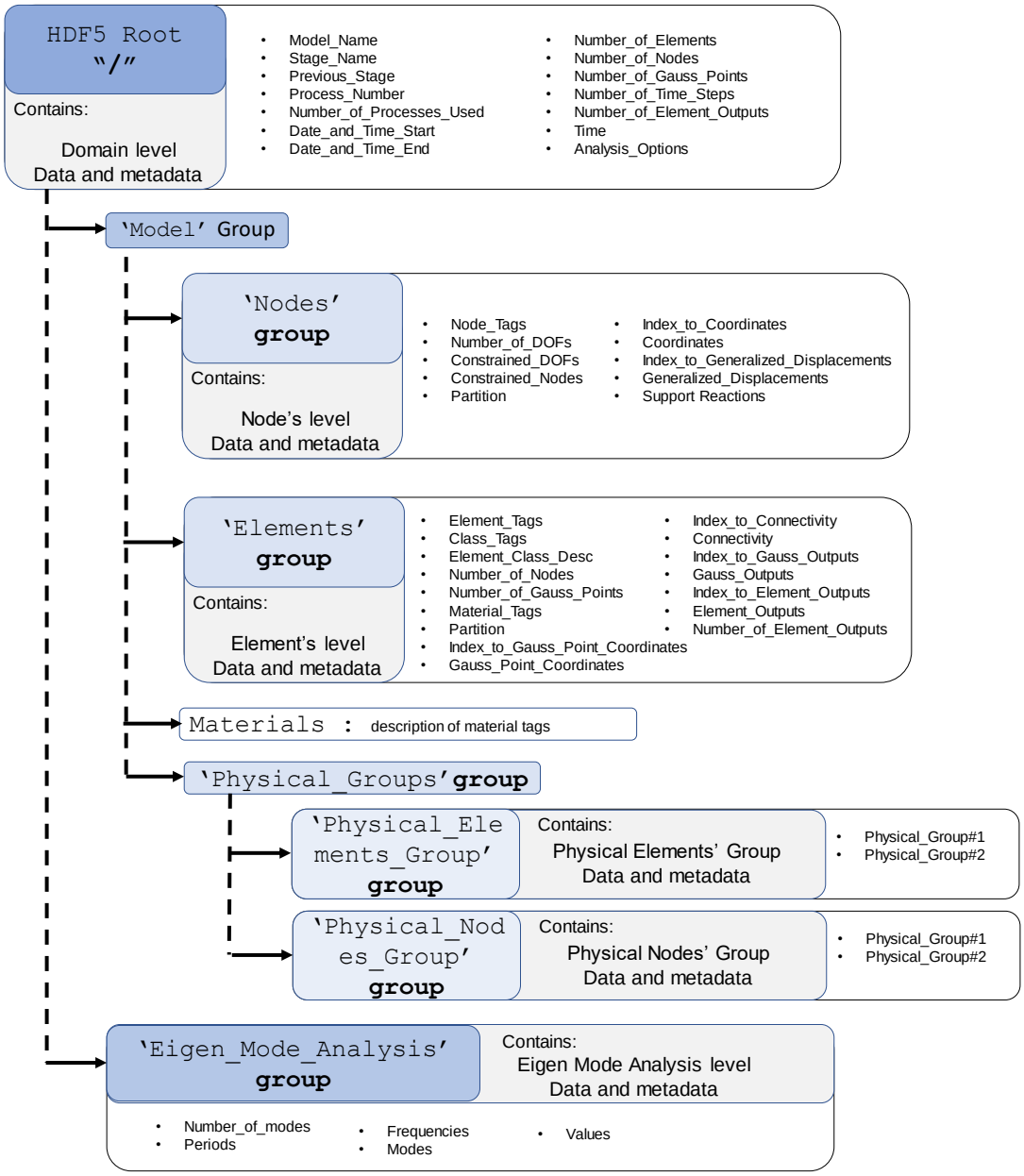


Figure 206.3: Design of the ESSI `.h5.feioutput` data format.

equal to the number of CPU's/Cores used.

- `Number_of_Nodes`: (`int`) A single scalar integer array with the number of nodes defined in that domain.

- `Number_of_Elements`: (`int`) A single scalar integer array with the number of elements defined in that domain.

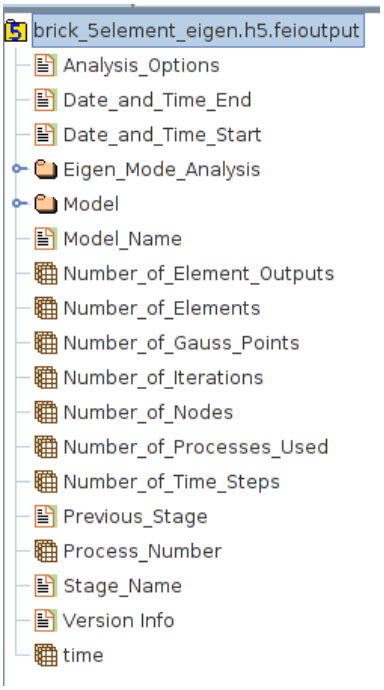


Figure 206.4: Data accessible in the `Root` directory of HDF5 file.

- `Number_of_Gauss_Points`: (`int`) A single scalar integer array with the number of gauss points in that domain.
- `Number_of_Element_Outputs`: (`int`) A single scalar integer array with stores the total length of `Element_Output` array in that domain.
- `Analysis_Options`: (`string`) An array of strings with the analysis options selected for the current analysis.
- `Date_and_Time_Start`: (`string`) A single string with the Date and Time of the start of the analysis. (In Coordinated Universal Time, UTC)
- `Date_and_Time_End`: (`string`) A single string with the Date and Time of the end of the analysis. (In Coordinated Universal Time, UTC)

  item `Version_Info`:(`string`) A Long string containing the version information of Real-ESSI simulator.
- `Model`: A group that contains the `Nodes` and `Elements` groups. It contains essential information about the mesh and analysis results for nodes and elements. See Section 206.5.2

- `Eigen_Mode_Analysis`: A group that contains the information about the the eigen mode analysis results of the doamin. See Section 206.5.6

206.5.2 The `Model` group

The `Model` group contains information about the mesh and analysis outputs. It contains the following groups as shown in Figure 206.5 and is also described below

- `Nodes`: A group that contains the information about the defined nodes and their output for this analysis. See Section 206.5.3
- `Elements`: A group that contains the information about the defined Elements and their output for this analysis. See Section 206.5.4
- `Physical_Groups`: A group that contains the information physical group of elements and nodes defined in that domain. See Section 206.5.5
- `Material`: (`string`) A string array which contains information about the material tag defined in the analysis for that loading stage. Section 206.5.7

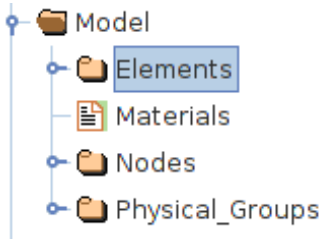


Figure 206.5: `Model` group directory of HDF5 file.

Subgroups `Nodes` and `Elements` store several integer and double precision arrays, that contain all necessary information for post processing.

206.5.3 The `Nodes` group

The `Nodes` group contains information about the nodal coordinates of the model, their tags, the number of DOFs defined at each node, and the corresponding solution variables (DOF results or generalized displacements) for each time step.

The format used to store the data is designed to give the fastest possible access time to the data of interest. Stored within the `Nodes` groups (and also in `Elements`) are two types of arrays: *data arrays* and *index arrays*.

- Data Arrays :: Data arrays might be floating-point arrays or integer arrays and have names not starting with 'Index_to_'.
- Index Arrays :: All index arrays are integer arrays and have names starting with the word 'Index_to_' followed by the name of the array which this array indexes.

The concept of *index array* is an important one regarding speed of access to data. These arrays map the integer tag number of the nodes (or elements) to the data. This allows fast access to components which minimizes searching within arrays to find the data of interest.

The Nodes group contains the following index arrays.

- Index_to_Coordinates: (int) Indexes the coordinates of nodes. See section 206.5.3.6
- Index_to_Generalized_Displacements: (int) Indexes the outputs of nodes (generalized displacements). See section 206.5.3.7

The following are the data arrays available in the Nodes group (shown in Figure ?? along with their respective indexing array:

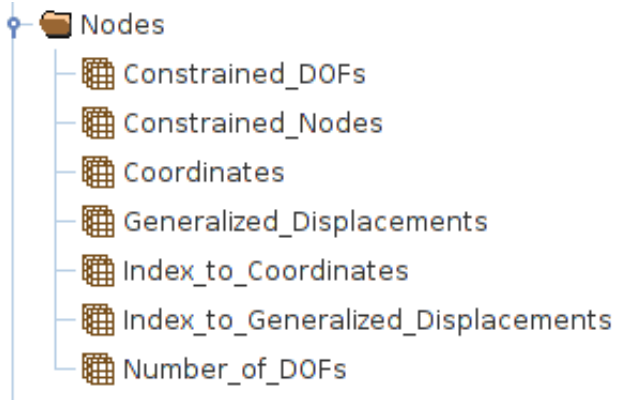


Figure 206.6: Nodes group directory of HDF5 file.

- Coordinates: (float) 1-D array containing nodal coordinates fixed in time. [Indexed by Index_to_Coordinates array]. See section 206.5.3.6
- Generalized_Displacements: (float) 2-D array containing the DOF values for the solution at each time step. [Indexed by Index_to_Generalized_Displacements array]. See section 206.5.3.7
- Number_of_DOFs: (int) 1-D array mapping the integer tag of each node to the number of DOFs at that node. See section 206.5.3.1

- `Constrained_Nodes`: (`int`) 1-D array mapping the integer tag of each node to the number of DOFs at that node. See section 206.5.3.3
- `Constrained_DOFs`: (`int`) 1-D array mapping the integer tag of each node to the number of DOFs at that node. See section 206.5.3.4
- `Partition`: (`int`) 1-D array mapping the integer tag of each node to the number of DOFs at that node. See section 206.5.3.2
- `Support_Reactions`: (`float`) 1-D array mapping the integer tag of each node to the number of DOFs at that node. See section 206.5.3.5

For example, lets imagine that the user has defined 4 nodes and applied the following constrained as shown below :

Listing 206.1: Node_Example

```
// defining nodes
add node # 2 at (0*m, 0*m, 0*m) with 3 dofs;
add node # 4 at (1*m, 1*m, 1*m) with 6 dofs;
add node # 5 at (2*m, 2*m, 2*m) with 3 dofs;
add node # 6 at (1*m, 0*m, 5*m) with 3 dofs;

// applying constraints
fix node # 2 dofs ux uy;
fix node # 4 rx ry rz;
fix node # 6 Ux p;
```

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

206.5.3.1 `Number_of_DOFs`

`Number_of_DOFs` array defines the number of degrees of freedom for each node defined in the model. It is an `integer` array of length equal to the maximum node tag + 1 (including tag 0). If a node tag does not exists, the corresponding dofs is output as -1. Figure 206.7 shows how to read `Number_of_DOFs` array. In the given example Listing 206.1 , node tag 2 has 3 degrees of freedom. Similarly, node tag 4 has 6 degrees of freedom and so on.

206.5.3.2 `Partition`

`Partition` array contains the domain or process id on which nodes tags were defined in case of a parallel simulation. For sequnetial runs, this dataset is not available. If a node tag does not exists, the

corresponding partition process id is output as -1. Figure 206.7 shows how to read `Partition` array. In the given example Listing 206.1, node tag 2 is assigned to process id 1. Similarly, node tag 4 is assigned to process id 2 and so on.

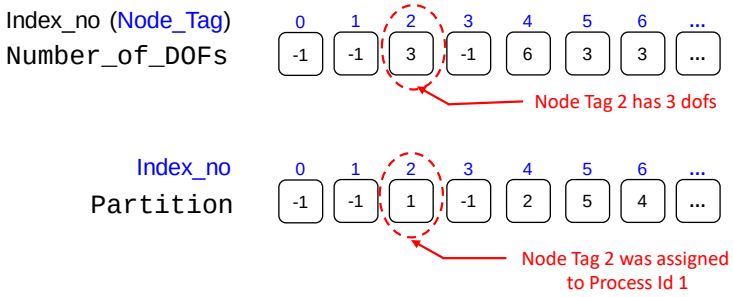


Figure 206.7: Arrays describing node information in `Nodes` group directory of HDF5 file.

206.5.3.3 `Constrained_Nodes`

`Constrained_Nodes` array contains a list of node tags for each dof on which fixities were applied. Figure 206.8 shows how to read `Constrained_Nodes`. In the given example Listing 206.1, dof ux and uy of node tag 2 is fixed. Similarly, for node tag 4 dofs rx ry and rz are fixed. Thats why in the `Constrained_Nodes` array contains node tags 2,2,4,4,4 and so on multiple times for each dof of the corresponding node tag fixed.

206.5.3.4 `Constrained_DOFs`

`Constrained_DOFs` array contains a list of dofs of the corresponding node tag on which fixities were applied. Figure 206.8 shows how to read `Constrained_DOFs`. In the given example Listing 206.1, dof ux (0) and uy (1) of node tag 2 is fixed. Similarly, for node tag 4 dofs rx (3) ry (4) and rz (5) are fixed. Figure 206.8 also show the DOF if numbering for different dof types i.e. for 3dof, 4dof, 6dof and 7dof nodes.

206.5.3.5 `Support_Reactions`

`Support_Reactions` contains a (`float`) array of reaction forces for the constrained degree of freedoms (DOFs). Figure 206.8 shows how to read `Constrained_DOFs`. In the given example Listing 206.1, dof ux (0) and uy (1) of node tag 2 has support reactions 1N and -5N respectively. Similarly, for node tag 4 dofs rx (3) ry (4) and rz (5) have reactions 45, 3 and -5 $N-m$ respectively. The reaction forces for displacement dofs (u_x, u_y, u_z) are forces in units of N, for rotational dofs (r_x, r_y, r_z) are moments in units

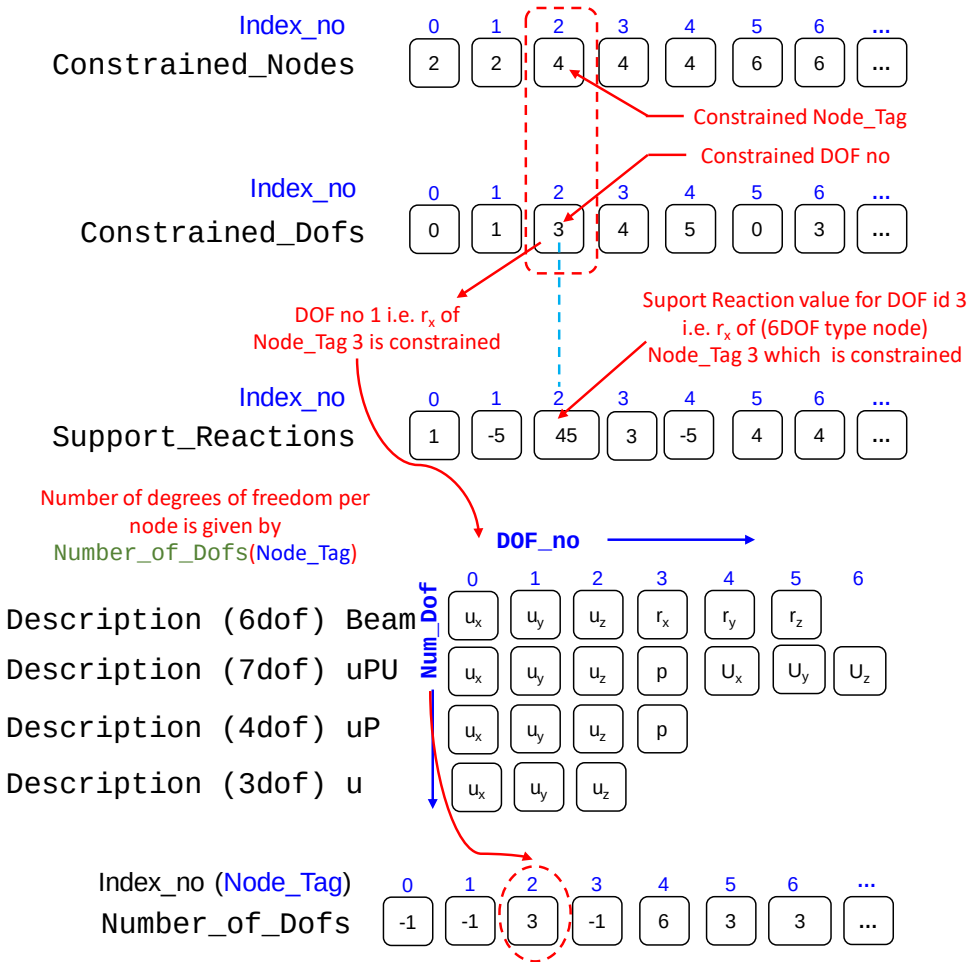


Figure 206.8: Arrays describing constrained nodes and reaction information in `Nodes` group directoory of HDF5 file.

of $N-m$ and for pressure dof (p) is pascal (Pa). Section 206.6 describes the output definitions for nodes with different dof-types.

206.5.3.6 `Coordinates`

The `Coordinates` array is a vertical stack of the nodal coordinate values. it is indexed by the `Index_to_Coordinates`, which relates the integer tag of each nodes (defined at the moment of creation of every node) with the position on this array of the 3 nodal coordinates. If the node with a given tag is not defined (if a tag number or several are skipped) this array will contain a negative number (-1) for that tag value. Figure 206.9 shows how to read `Coordinates` and `index_to_Coordinates` of nodes.

The size of `Index_to_Coordinates` is always the maximum tag defined plus one (zero can be a tag too). The size of the `Coordinates` array is three times the number of nodes defined. In the given

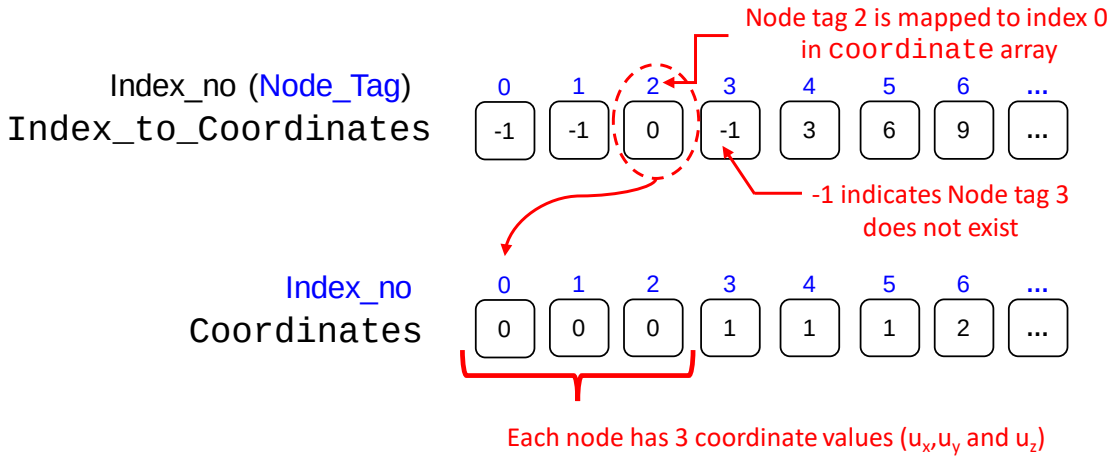


Figure 206.9: `Coordinates` and `Index_to_Coordinates` arrays in `Nodes` group directory of HDF5 file.

example Listing 206.1, the coordinates of node tag 2 is (0*m,0*m,0*m). The coordinate of node tag 4 is (1*m,1*m,1*m) and so on. The coordinates have unit of meter (m).

206.5.3.7 `Generalized_Displacements`

The `Generalized_Displacements` array is a 2-D array containing the computed solution at the nodal degrees of freedom for all nodes and all times steps. It is indexed by the `Index_to_Generalized_Displacements` array (first index) and time (second index). Figure 206.10 shows how to read `Generalized_Displacements` and `index_to_Generalized_Displacements` of nodes.

Output for displacement dofs (u_x, u_y, u_z) are in units of m, for rotational dofs (r_x, r_y, r_z) are in units of *radian* and for pressure dof (p) is in pascal (Pa).

With every time step, another column is added to the `Generalized_Displacements` array. This means that the time index (starting at 0) is directly related to the `time` array at the root of the HDF5 file.

The rows of the `Generalized_Displacements` array contain the results for each DOF of the current node. For a 3-DOF node these will be displacement in X, Y, and Z (u_x, u_y, and u_z) respectively. For higher number-of-dof nodes the first three components are the same but the remaining ones are going to depend on the connecting elements. For example, 6-DOF nodes might carry information on nodal rotations (beams and shells) or might be fluid displacements in case the elements connecting are u–U coupled fluid-porous-solid elements as shown in Figure 206.8. Section 206.6 describes the output definitions for nodes with different dof-types.

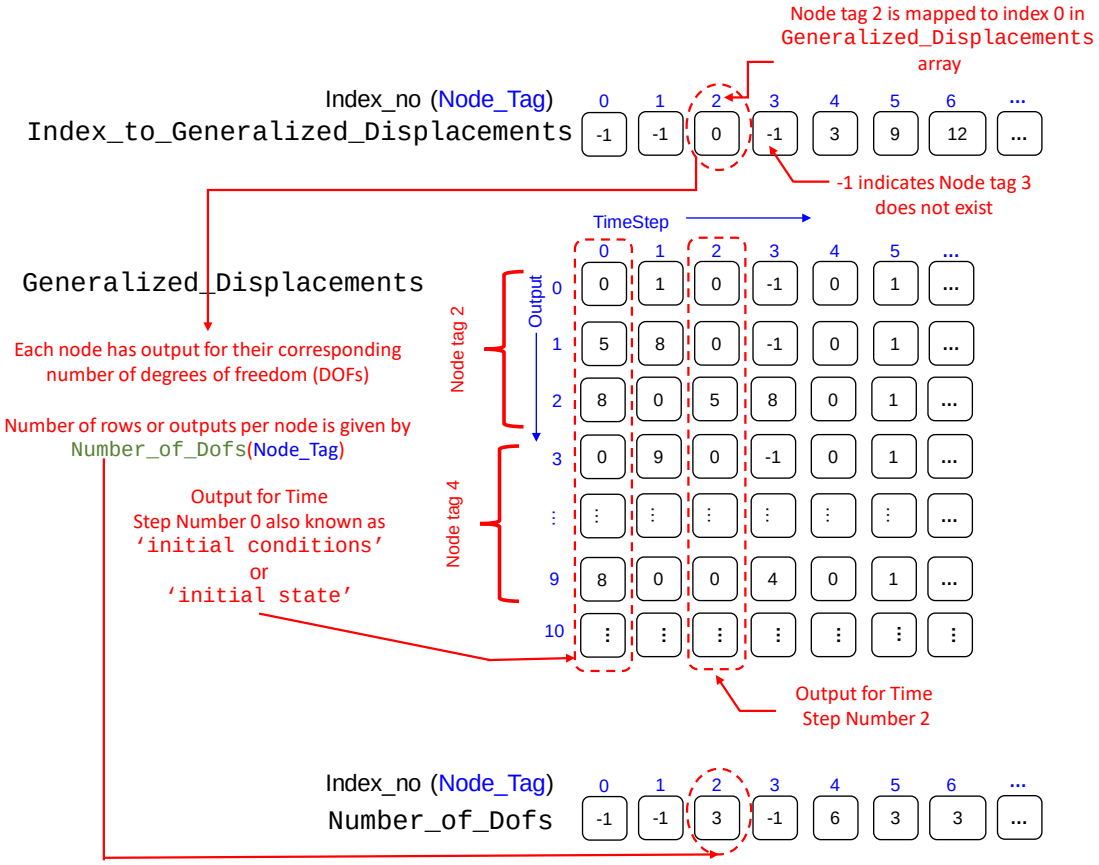


Figure 206.10: The `Index_to_Generalized_Displacements` and `Generalized_Displacements` in `Nodes` group directory of HDF5 file.

206.5.4 The `Elements` group

The `Elements` group contains information on the finite element mesh such as: connectivity array, element types, location of Gauss-point integration coordinates (global), materials defined at each element, and all available output from the elements. At this point it is important to note that the kind and amount of output contained in the element output arrays are dependent on element implementation as it is the elements who are in charge of controlling their output. For information on the specific output of each element, consult the documentation for the respective element.

The idea and organization of the datasets contained herein is analogous to the of the `Nodes` group. This group contains the following *index arrays*.

- `Index_to_Connectivity`: (`int`) Maps element tag to location within the `Connectivity` array.
- `Index_to_Gauss_Point_Coordinates`: Maps element tag to location of Gauss-point coordinates in the `Gauss_Point_Coordinates` array.

- `Index_to_Gauss_Outputs`: (`int`) Maps element tag to location of gauss output of element in the `Gauss_Outputs` array.
- `Index_to_Element_Outputs`: (`int`) Maps element tag to location of output in the `Elements_Outputs` array.

The following are the data arrays available in the `Elements` group (shown in Figure ?? along with their respective indexing array:

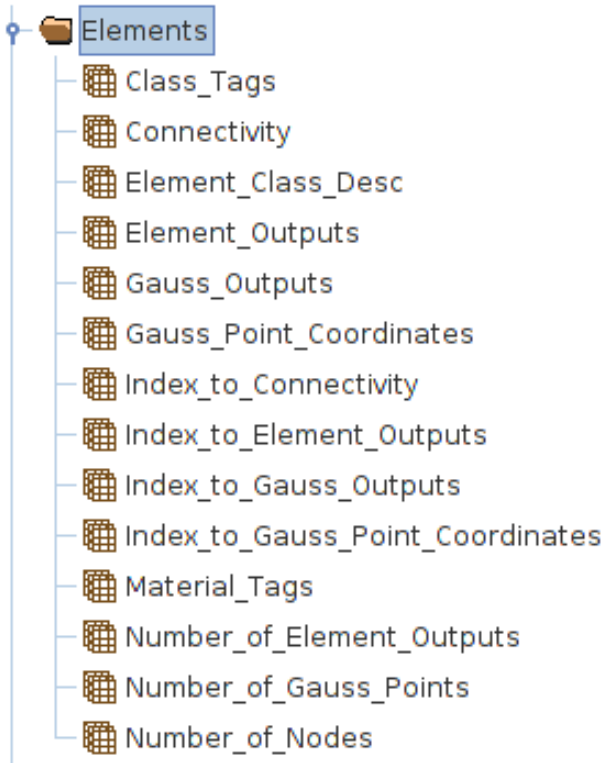


Figure 206.11: `Elements` group directory of HDF5 file.

- `Class_Tags`: (`int`) It is an array that contains the integer ids for each elements tag defined in the model and present in thet domain. See section 206.5.4.4
- `Number_of_Nodes`: (`int`) Maps element tag number to number of nodes in the element (-1 if element tag is not defined). See section 206.5.4.1.
- `Number_of_Gauss_Points`: (`int`) Maps tag number to the number of Gauss-integration points in that element (-1 if not element tag is not defined). It stores 0 in case of no gauss points (mostly for structural elements). See section 206.5.4.3

- `Material_Tags`: (`int`) Maps tag number to the tag number of the material contained in that element (-1 if element tag is not defined or material for that element is not defined). See section 206.5.4.6
- `Partition`: (`int`) Maps tag number to the tag number of the processor id on which it is assigned (-1 if element tag is not defined). See section 206.5.4.5
- `Connectivity`:(`int`) Contains the nodes tags which are connected by this element [indexed by the `Index_to_Connectivity` array] (-1 if element tag is not defined). See section 206.5.4.7
- `Gauss_Point_Coordinates`: (`float`) Contains the coordinates of all the gauss pints of that element tag [indexed by the `Index_to_Gauss_Point_Coordinates` array] (-1 if element tag is not defined) . See section 206.5.4.8
- `Gauss_Outputs`: (`float`) Contains the stress, strain and plastic strain outputs for each gauss point present in the corresponding element tag[indexed by the `Index_to_Gauss_Outputs` array]. See section 206.5.4.9
- `Element_Outputs`: (`float`) Contains output other than gauss points by the [indexed by the `Index_to_Element_Outputs` array]. See section 206.5.4.2

For example, lets imagine that the user has defined 4 elements and some materials as shown below :

Listing 206.2: Element_Example

```
1
2  // defining materials
3
4  add material #1 type uniaxial_elastic elastic_modulus = 1*Pa ↩
       viscoelastic_modulus = 0*Pa*s;
5  add material #2 type linear_elastic_isotropic_3d mass_density = 2000*kg/m^3 ↩
       elastic_modulus = 200*MPa poisson_ratio = 0.3;
6
7  // defining elements
8  add element #2 type truss with nodes (1,2) use material # 1 cross_section = ↩
       1*m^2 mass_density = 0*kg/m^3;
9  add element #4 type 8NodeBrick with nodes (1,8,6, 4, 3, 9, 2, 5) use material #2;
10 add element #5 type HardContact with nodes (3,2) normal_stiffness =1e10*N/m ↩
       tangential_stiffness = 1e4*Pa*m normal_damping = 0*kN/m*s ↩
       tangential_damping = 0*N/m*s friction_ratio = 1 contact_plane_vector = ↩
       (0,0,1);
11 add element #6 type 8NodeBrick with nodes (11,18,61,14, 3,19,22,15) use ↩
       material #2;
```

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

206.5.4.1 Number_of_Nodes

Number_of_Nodes array defines the number of nodes for each element defined in the model. It is an integer array of length equal to the maximum element tag + 1 (including tag 0). If a element tag does not exists, the corresponding number of nodes is output as -1. Figure 206.12 shows how to read Number_of_Nodes array. In the given example Listing 206.2 , element tag 2 has 2 nodes. Similarly, element tag 4 has 8 nodes and so on.

206.5.4.2 Number_of_Element_Outputs

Number_of_Element_Outputs array defines the number of outputs for each element defined in the model. It is an integer array of length equal to the maximum element tag + 1 (including tag 0). If a element tag does not exists, the corresponding number of outputs is stored as-1. Figure 206.12 shows how to read Number_of_Element_Outputs array. In the given example Listing 206.2 , element tag 2 has 2 outputs. Similarly, element tag 5 has 9 outputs but element tag 4 has 0 outputs and so on.

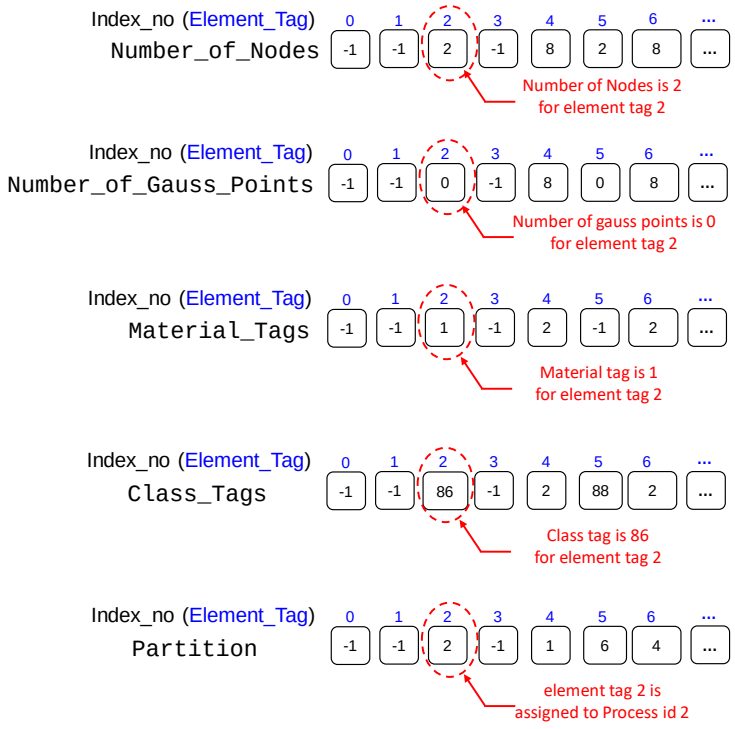


Figure 206.12: Arrays describing element information in Elements group directory of HDF5 file.

206.5.4.3 Number_of_Gauss_Points

Number_of_Gauss_Points array defines the number of gauss points for each element defined in the model. It is an integer array of length equal to the maximum element tag + 1 (including tag 0). If a element tag does not exists, the corresponding number of outputs is stored as-1. Figure 206.12 shows how to read Number_of_Gauss_Points array. In the given example Listing 206.2 , element tag 2 and 5 has 0 gauss points. Similarly, element tag 4 has 8 gauss points and so on.

206.5.4.4 Class_Tags

Class_Tags array defines an (unique) element type for each of the element defined in the model. It is an integer array of length equal to the maximum element tag + 1 (including tag 0). If a element tag does not exists, the corresponding number of outputs is stored as-1. Figure 206.12 shows how to read Class_Tags array. In the given example Listing 206.2 , element tag 2 has class tag of 88 (i.e. truss element) . Similarly, element tag 5 has class tag 2 (i.e. 8 node brick element) and so on. Table 206.1 and Table 206.2 shows class tags for different element types.

206.5.4.5 Partition

Partition array contains the domain or process id on which element tags were defined in case of a parallel simulation. For sequential runs, this dataset is not available. If a element tag does not exists, the corresponding partition process id is output as -1. Figure 206.12 shows how to read Partition array. In the given example Listing 206.2, element tag 2 is assigned to process id 2. Similarly, element tag 4 is assigned to process id 1 and so on.

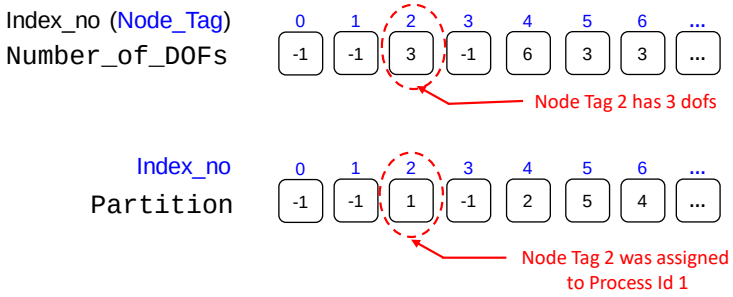


Figure 206.13: Arrays describing node information in Nodes group directory of HDF5 file.

Element Type	Class Tag	Element Type	Class Tag
EightNodeBrick	2	EightNodeBrick_up	3
EightNodeBrick_upU	4	TwentyNodeBrick	5
TwentyNodeBrick_up	6	TwentyNodeBrick_upU	7
TwentySevenNodeBrick	8	TwentySevenNodeBrick_up	9
TwentySevenNodeBrick_upU	10	VariableNodeBrick	11
VariableNodeBrick_up	12	VariableNodeBrick_upU	13
EightNodeBrickOrderOne	14	EightNodeBrickOrderOne_up	15
EightNodeBrickOrderOne_upU	16	TwentyNodeBrickOrderOne	17
TwentyNodeBrickOrderOne_up	18	TwentyNodeBrickOrderOne_upU	19
TwentySevenNodeBrickOrderOne	20	TwentySevenNodeBrickOrderOne_up	21
TwentySevenNodeBrickOrderOne_upU	22	VariableNodeBrickOrderOne	23
VariableNodeBrickOrderOne_up	24	VariableNodeBrickOrderOne_upU	25
EightNodeBrickOrderTwo	26	EightNodeBrickOrderTwo_up	27
EightNodeBrickOrderTwo_upU	28	TwentyNodeBrickOrderTwo	29
TwentyNodeBrickOrderTwo_up	30	TwentyNodeBrickOrderTwo_upU	31
TwentySevenNodeBrickOrderTwo	32	TwentySevenNodeBrickOrderTwo_up	33
TwentySevenNodeBrickOrderTwo_upU	34	VariableNodeBrickOrderTwo	35
VariableNodeBrickOrderTwo_up	36	VariableNodeBrickOrderTwo_upU	37
EightNodeBrickOrderThree	38	EightNodeBrickOrderThree_up	39
EightNodeBrickOrderThree_upU	40	TwentyNodeBrickOrderThree	41
TwentyNodeBrickOrderThree_up	42	TwentyNodeBrickOrderThree_upU	43
TwentySevenNodeBrickOrderThree	44	TwentySevenNodeBrickOrderThree_up	45
TwentySevenNodeBrickOrderThree_upU	46	VariableNodeBrickOrderThree	47
VariableNodeBrickOrderThree_up	48	VariableNodeBrickOrderThree_upU	49

Table 206.1: Class Tags for Real-ESSI Elements (Part -1)

206.5.4.6 `Material_Tags`

`Material_Tags` array defines the material tag number for each element defined in the model. It is an `integer` array of length equal to the maximum element tag + 1 (including tag 0). If a element tag does not exists, the corresponding number of outputs is stored as-1. Figure 206.12 shows how to read `Material_Tags` array. In the given example Listing 206.2 , element tag 2 has material tag of 1. Similarly, element tag 4 has material tag 2. Whereas, element tag 5 have material tag of -1.

Element Type	Class Tag	Element Type	Class Tag
EightNodeBrickOrderFour	50	EightNodeBrickOrderFour_up	51
EightNodeBrickOrderFour_upU	52	TwentyNodeBrickOrderFour	53
TwentyNodeBrickOrderFour_up	54	TwentyNodeBrickOrderFour_upU	55
TwentySevenNodeBrickOrderFour	56	TwentySevenNodeBrickOrderFour_up	57
TwentySevenNodeBrickOrderFour_upU	58	VariableNodeBrickOrderFour	59
VariableNodeBrickOrderFour_up	60	VariableNodeBrickOrderFour_upU	61
EightNodeBrickOrderFive	62	EightNodeBrickOrderFive_up	63
EightNodeBrickOrderFive_upU	64	TwentyNodeBrickOrderFive	65
TwentyNodeBrickOrderFive_up	66	TwentyNodeBrickOrderFive_upU	67
TwentySevenNodeBrickOrderFive	68	TwentySevenNodeBrickOrderFive_up	69
TwentySevenNodeBrickOrderFive_upU	70	VariableNodeBrickOrderFive	71
VariableNodeBrickOrderFive_up	72	VariableNodeBrickOrderFive_upU	73
EightNodeBrickOrderSix	74	EightNodeBrickOrderSix_up	75
EightNodeBrickOrderSix_upU	76	TwentyNodeBrickOrderSix	77
TwentyNodeBrickOrderSix_up	78	TwentyNodeBrickOrderSix_upU	79
TwentySevenNodeBrickOrderSix	80	TwentySevenNodeBrickOrderSix_up	81
TwentySevenNodeBrickOrderSix_upU	82	VariableNodeBrickOrderSix	83
VariableNodeBrickOrderSix_up	84	VariableNodeBrickOrderSix_upU	85
HardContact	86	SoftContact	87
Truss	88	ElasticBeam	89
ThreeNodeAndesShell	90	FourNodeAndesShell	91
ShearBeam	92	rank_one_deficient_elastic_pinned_fixed_beam	93
DispBeamColumn3d	94	Cosserat_8node_brick	95

Table 206.2: Class Tags for Real-ESSI Elements (Part -2)

206.5.4.7 `Connectivity`

`Connectivity` array stores the list of node tags in the same order as they were defined along with element declaration in the model. It is an `integer`. It is indexed by the `Index_to_Connectivity` array and the number of rows is defined by `Number_of_Nodes`. Figure 206.14 shows how to read `Connectivity` array for a particular element. In the given example Listing 206.2 , element tag 2 includes node tag 1 and 2. Similarly, element tag 4 includes 8 node tas 1,8,6, 4, 3, 9, 2 and 5 respectively.

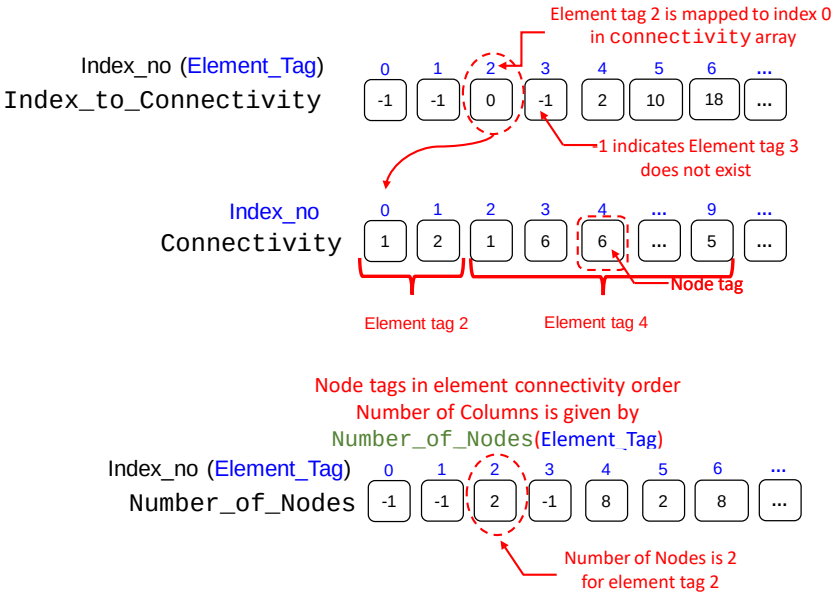


Figure 206.14: Arrays describing connectivity information in `Elements` group directory of HDF5 file.

206.5.4.8 `Gauss_Point_Coordinates`

`Gauss_Point_Coordinates` array stores the coordinates of the gauss points declared inside for each element defined in the model. It is a `float` array indexed by the `Index_to_Gauss_Point_Coordinates` array and the number of rows is defined by 3 (x, y, z) times `Number_of_Gauss_Points`. Figure 206.15 shows how to read `Gauss_Point_Coordinates` 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 point coordinates output for that element is $8 \times 3 = 24$. The index from which the coordinates information start is 0. Coordinate values for first 3 index $(0, 1, 2)$ corresponds to gauss point 1 and next 3 index $(3, 4, 5)$ corresponds to gauss point 2 and so on.

206.5.4.9 `Gauss_Output`

`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 206.16 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