

Real-ESSI Simulator

Domain Specific Language

José Abell, Yuan Feng, Han Yang, Hexiang Wang
and
Boris Jeremić

University of California, Davis, CA



Version: 3Jul2025, 10:20

<http://real-essi.us>

This document is an excerpt from: <http://sokocalo.engr.ucdavis.edu/~jeremic/LectureNotes/>

please use google-chrome to view this PDF so that hyperlinks work



Contents

1	Input, Domain Specific Language (DSL)	1991-2005-2010-2011-2012-2015-2016-2017-2020-2021-	11
1.1	Chapter Summary and Highlights		12
1.2	Introduction		12
1.3	Domain Specific Language (DSL), English Language Binding		13
1.3.1	Running Real-ESSI		13
1.3.2	Finishing Real-ESSI Program Run		15
1.3.3	Real-ESSI Variables, Basic Units and Flow Control		16
1.3.4	Modeling		20
	Modeling, Material Model: Adding a Material Model to the Finite Element Model . . .		22
	Modeling, Material Model: Linear Elastic Isotropic Material Model		23
	Modeling, Material Model: Cross Anisotropic Linear Elastic Material Model		24
	Modeling, Material Model: von Mises Associated Material Model with Linear Isotropic and/or Kinematic Hardening		25
	Modeling, Material Model: von Mises Associated Material Model with Isotropic Hard- ening and/or Armstrong-Frederic Nonlinear Kinematic Hardening		26
	Modeling, Material Model: Drucker-Prager Associated Material Model with Linear Isotropic and/or Kinematic Hardening		27
	Modeling, Material Model: Drucker-Prager Associated Material Model with Isotropic Hardening and/or Armstrong-Frederick Nonlinear Kinematic Hardening		28
	Modeling, Material Model: Drucker-Prager Associated Material Model with Isotropic Hardening and/or Armstrong-Frederick Nonlinear Kinematic Hardening and Nonlinear Duncan-Chang Elasticity		29
	Modeling, Material Model: Drucker-Prager Nonassociated Material Model with Linear Isotropic and/or Kinematic Hardening		30
	Modeling, Material Model: Drucker-Prager Nonassociated Material Model with Linear Isotropic and/or Armstrong-Frederick Nonlinear Kinematic Hardening		31

Modeling, Material Model: Hyperbolic Drucker-Prager Nonassociated Material Model with Linear Isotropic and/or Armstrong-Frederick Nonlinear Kinematic Hardening	33
Modeling, Material Model: Rounded Mohr-Coulomb Associated Linear Isotropic Hardening Material Model	35
Modeling, Material Model: Cam Clay Material Model	36
Modeling, Material Model: von Mises Associated Multiple Yield Surface Material Model	37
Modeling, Material Model: von Mises Associated Multiple Yield Surface Material Model that Matches G/G_{max} Curves	38
Modeling, Material Model: Drucker-Prager Nonassociated Multi-Yield Surface Material Model	39
Modeling, Material Model: Drucker-Prager Nonassociated Material Model that Matches G/G_{max} Curves	41
Modeling, Material Model: Rounder Mohr-Coulomb Nonassociated Multi-Yield Surface Material Model	43
Modeling, Material Model: Tsinghua Liquefaction Material Model	45
Modeling, Material Model: SANISand Material Model, version 2004	47
Modeling, Material Model: SANISand Material Model, version 2008	49
Modeling, Material Model: Cosserat Linear Elastic Material Model	52
Modeling, Material Model: von Mises Cosserat Material Model	53
Modeling, Material Model: Uniaxial Linear Elastic, Fiber Material Model	54
Modeling, Material Model: Stochastic Uniaxial Linear Elastic Model	55
Modeling, Material Model: Stochastic Uniaxial Nonlinear Armstrong Frederick Model	56
Modeling, Material Model: Uniaxial Nonlinear Concrete, Fiber Material Model, version 02	58
Modeling, Material Model: Faria-Oliver-Cervera Concrete Material	59
Modeling, Material Model: Plane Stress Layered Material	60
Modeling, Material Model: Uniaxial Nonlinear Steel, Fiber Material Model, version 01	61
Modeling, Material Model: Uniaxial Nonlinear Steel, Fiber Material Model, version 02	62
Modeling, Material Model: Plane Stress Plastic Damage Concrete Material	63
Modeling, Material Model: Plane Stress Rebar Material	64
Modeling, Nodes: Adding Nodes	65
Modeling, Nodes: Adding Stochastic Nodes	66
Modeling, Nodes: Define Nodal Physical Group	67
Modeling, Nodes: Adding Nodes to Nodal Physical Group	68

Modeling, Nodes: Removing Nodal Physical Group	69
Modeling, Nodes: Print Nodal Physical Group	70
Modeling, Nodes: Removing Nodes	71
Modeling, Nodes: Adding Nodal Mass, for 3DOFs and/or 6DOFs	72
Modeling, Finite Element: Adding Finite Elements	73
Modeling, Finite Element: Define Finite Element Physical Group	74
Modeling, Finite Element: Adding Elements to Physical Element Group	75
Modeling, Finite Element: Remove Physical Finite Element Group	76
Modeling, Finite Element: Print Physical Finite Element Group	77
Modeling, Finite Element: Remove Finite Element	78
Modeling, Finite Element: Truss Element	79
Modeling, Finite Element: Kelvin-Voigt Element	80
Modeling, Finite Element: Inerter Element	81
Modeling, Finite Element: Shear Beam Element	82
Modeling, Finite Element: Stochastic Shear Beam Element	83
Modeling, Finite Element: Elastic Beam–Column Element	84
Modeling, Finite Element: Large Displacement Elastic Beam–Column Element, with Corotational Transformation	87
Modeling, Finite Element: Timoshenko Elastic Beam–Column Element	88
Modeling, Finite Element: Timoshenko Elastic Beam–Column Element with Directional Shear Correction Coefficients	90
Modeling, Finite Element: Adding 1D Fiber to a Beam Cross Section	92
Modeling, Finite Element: Adding Fiber Section to the Finite Element Model	93
Modeling, Finite Element: 3D Displacement Based Fiber Beam-Column Element	94
Modeling, Finite Element: 3D Displacement Based Fiber Beam-Column Element with Corotational Coordinate Transformation	95
Modeling, Finite Element: 3DOF+6DOF=9DOF Beam-Column Element	96
Modeling, Finite Element: 4 Node ANDES Shell with Drilling DOFs	98
Modeling, Finite Element: 3 Node ANDES Shell with Drilling DOFs	99
Modeling, Finite Element: 4 Node Shell NLDKGQ, or 4 Node Shell Xin-Zheng-Lu	100
Modeling, Finite Element: Inelastic Layered Shell Section	101
Modeling, Finite Element: ElasticMembranePlaneStress Element (to be removed!)	102
Modeling, Finite Element: InelasticMembranePlaneStress Element (to be removed!)	103
Modeling, Finite Element: SuperElementLinearElasticImport	104
Modeling, Finite Element: 8 Node Brick Element	106

Modeling, Finite Element: 20 Node Brick Element	108
Modeling, Finite Element: 27 Node Brick Element	110
Modeling, Finite Element: Variable 8-27 Node Brick Element	112
Modeling, Finite Element: 8 Node Brick u-p Element	114
Modeling, Finite Element: 20 Node Brick u-p Element	117
Modeling, Finite Element: 27 Node Brick u-p Element	120
Modeling, Finite Element: 8 Node Brick u-p-U Element	123
Modeling, Finite Element: 20 Node Brick u-p-U Element	126
Modeling, Finite Element: 27 Node Brick u-p-U Element	129
Modeling, Finite Element: 8 Node Cosserat Brick Element	132
Modeling, Finite Element: Bonded Contact/Interface/Joint Element	133
Modeling, Finite Element: Coupled Bonded Contact/Interface/Joint Element	134
Modeling, Finite Element: Force Based Dry Hard Contact/Interface/Joint Element	135
Modeling, Finite Element: Force Based Dry Soft Contact/Interface/Joint Element	136
Modeling, Finite Element: Force Based Coupled Hard Contact/Interface/Joint Element	138
Modeling, Finite Element: Force Based Coupled Soft Contact/Interface/Joint Element	140
Modeling, Finite Element: Stress Based Dry Hard Contact/Interface/Joint Element with Elastic Perfectly Plastic Shear Behavior	142
Modeling, Finite Element: Stress Based Dry Hard Contact/Interface/Joint Element with Nonlinear Hardening Shear Behavior	144
Modeling, Finite Element: Stress Based Dry Hard Contact/Interface/Joint Element with Nonlinear Hardening and Softening Shear Behavior	146
Modeling, Finite Element: Stress Based Dry Soft Contact/Interface/Joint Element with Elastic Perfectly Plastic Shear Behavior	149
Modeling, Finite Element: Stress Based Dry Soft Contact/Interface/Joint Element with Nonlinear Hardening Shear Behavior	151
Modeling, Finite Element: Stress Based Dry Soft Contact/Interface/Joint Element with Nonlinear Hardening and Softening Shear Behavior	153
Modeling, Finite Element: Stress Based Coupled Hard Contact/Interface/Joint Element with Elastic Perfectly Plastic Shear Behavior	156
Modeling, Finite Element: Stress Based Coupled Hard Contact/Interface/Joint Element with Nonlinear Hardening Shear Behavior	158
Modeling, Finite Element: Stress Based Coupled Hard Contact/Interface/Joint Element with Nonlinear Hardening and Softening Shear Behavior	160

Modeling, Finite Element: Stress Based Coupled Soft Contact/Interface/Joint Element with Elastic Perfectly Plastic Shear Behavior	163
Modeling, Finite Element: Stress Based Coupled Soft Contact/Interface/Joint Element with Nonlinear Hardening Shear Behavior	165
Modeling, Finite Element: Stress Based Coupled Soft Contact/Interface/Joint Element with Nonlinear Hardening and Softening Shear Behavior	167
Modeling, Finite Element: Neoprene Isolator Finite Element	170
Modeling, Finite Element: Lead Core Rubber Isolator/Dissipator Element	171
Modeling, Finite Element: Frictional Pendulum Isolator/Dissipator Finite Element ver- sion01	172
Modeling, Finite Element: Frictional Pendulum Isolator/Dissipator Finite Element ver- sion03	173
Modeling, Damping: Adding Rayleigh Damping	174
Modeling, Damping: Adding 3rd Order Caughey Damping	175
Modeling, Damping: Adding 4th Caughey Damping	176
Modeling, Constraints and Supports: Adding Constraints or Supports	177
Modeling, Constraints and Supports: Adding Stochastic Constraints or Supports	178
Modeling, Constraints and Supports: Free Constraint or Support	179
Modeling, Constraints and Supports: Add Tied/Connected Main-Follower Nodes for the Same DOFs	180
Modeling, Constraints and Supports: Adding Tied/Connected, Main-Follower Nodes for Different DOFs	181
Modeling, Constraints and Supports: Remove Tied/Connected Main-Follower equal DOFs	182
Modeling, Constraints and Supports: Adding Single Point Constraint to Nodes	183
Modeling, Acceleration Field: Adding Acceleration/Inertia Field	184
Modeling, Loads: Nodal Loads	185
Modeling, Loads: Nodal Path Loads	187
Modeling, Loads: Nodal Loads From Reactions	188
Modeling, Loads: Selfweight Element Load	189
Modeling, Loads: Selfweight Nodal Load	190
Modeling, Loads: 8 Node Brick Surface Load with the Constant Pressure	191
Modeling, Loads: 8 Node Brick Surface Load with Variable Pressure	192
Modeling, Loads: 20Node Brick Surface Load with the Constant Pressure	193
Modeling, Loads: 20 Node Brick, Surface Load with Variable Pressure	194

Modeling, Loads: 27 Node Brick Surface Load with the Constant Pressure	195
Modeling, Loads: 27 Node Brick Surface Load with Variable Pressure	196
Modeling, Loads: Removing Loads	197
Modeling, Loads: Domain Reduction Method, DRM	198
Modeling, Wave Field for Creating DRM Loads: Add Wave Field	202
Modeling, Wave Field for Creating DRM Loads: Deconvolution	205
Modeling, Wave Field for Creating DRM Input: Motions	206
Modeling, Wave Field for Creating DRM Input: Forces	207
Modeling, Wave Field for Creating DRM Loads: Add Inclined Plane Wave Field from Incident SV Wave Potential Magnitude	208
Modeling, Wave Field for Creating DRM Loads: Add Inclined Plane Wave Field from Incident SV Wave Time Series Signal	211
Modeling, Wave Field for Creating DRM Loads: Add Inclined Plane Wave Field from Incident P Wave Potential Magnitude	214
Modeling, Wave Field for Creating DRM Loads: Add Inclined Plane Wave Field from Incident P Wave Time Series Signal	216
Modeling, Wave Field for Creating DRM Loads: DRM Inclined Motion	218
Modeling, Imposed Motions: through Loads, Motion Time History, Constant Time Step	219
Modeling, Imposed Motions: through Loads, Stochastic Motion Time History, Constant Time Step	220
Modeling, Imposed Motions: through Loads, Stochastic Random Process Motions, Constant Time Step	222
Modeling, Imposed Motions: through Loads, Motion Time History, Variable Time Step	223
Modeling, Imposed Motions: Adding Load for Uniform Acceleration Time History . . .	224
Modeling, Imposed Motions: Remove Imposed Motions	225
Modeling, Random Variable: Adding Gaussian Random Variables	226
Modeling, Random Variable: Adding Gaussian Random Variables with Location	227
Modeling, Random Variable: Adding Lognormal Random Variables	228
Modeling, Random Variable: Adding Lognormal Random Variables with Location . . .	229
Modeling, Random Variable: Adding Lognormal Random Variables using Logarithmic Input	230
Modeling, Random Variable: Adding Lognormal Random Variables using Logarithmic Input with Location	231
Modeling, Random Variable: Adding Gamma Random Variables using Shape and Scale Parameters	232

Modeling, Random Variable: Adding Gamma Random Variables using Shape and Scale Parameters with Location	233
Modeling, Random Variable: Adding Gamma Random Variables using Mean and Standard Deviation Parameters	234
Modeling, Random Variable: Adding Gamma Random Variables using Mean and Standard Deviation Parameters with Location	235
Modeling, Random Variable: Adding Weibull Random Variables using Shape and Scale Parameters	236
Modeling, Random Variable: Adding Weibull Random Variables using Shape and Scale Parameters with Location	237
Modeling, Random Variable: Remove Random Variables	238
Modeling, Random Variable: Hermite Polynomial Chaos Expansion	239
Modeling, Random Variable: Output Hermite Polynomial Chaos Expansion Result . .	240
Modeling, Random Variable: Hermite Polynomial Chaos Expansion & Output Results .	241
Modeling, Random Field: Adding Random Field with Dimension and Order	242
Modeling, Random Field: Define Global Dimension Index of Random Field	243
Modeling, Random Field: Define Global Dimension Index of Random Field from File Input	244
Modeling, Random Field: Set Number of Polynomial Chaos Terms of Random Field .	245
Modeling, Random Field: Adding Random Field with Zero Correlation	246
Modeling, Random Field: Adding Random Field with Exponential Correlation	247
Modeling, Random Field: Adding Random Field with Triangular Correlation	248
Modeling, Random Field: Adding Random Field with Exponentially Damped Cosine Correlation	249
Modeling, Random Field: Adding Random Field with Gaussian Correlation	250
Modeling, Random Field: Remove Random Fields	251
Modeling, Random Field: Adding Random Variable to Random Field	252
Modeling, Random Field: Remove Random Variable From Random Field	253
Modeling, Random Field: Hermite Polynomial Chaos Karhunen Loève Expansion . . .	254
Modeling, Random Field: Hermite Polynomial Chaos Karhunen Loève Expansion with Inverse Order	255
Modeling, Random Field: Hermite Polynomial Chaos Karhunen Loève Expansion with Number of FE Elements Larger than Dimension of Hermite Polynomials . . .	256
Modeling, Random Field: Hermite Polynomial Chaos Karhunen Loève Expansion Using HDF5 Input	258

Modeling, Random Field: Hermite Polynomial Chaos Karhunen Loève Expansion with Inverse Order Using HDF5 Input	260
Modeling, Random Field: Output Hermite Polynomial Chaos Karhunen Loève Expansion Result	262
Modeling, Random Field: Adding Random Field from Hermite Polynomial Chaos Karhunen Loève Expansion HDF5 File	264
Modeling, Random Field: Adding Random Field from Marginal Distribution and Correlation	266
Modeling, Random Field: Add Triple Product of Hermite Polynomial Chaos Basis . . .	267
Modeling, Random Field: Add Double Product of Hermite Polynomial Chaos Basis . .	268
Modeling, Random Field: Generate Triple Product of Hermite Polynomial Chaos Basis	269
Modeling, Random Field: Generate Double Product of Hermite Polynomial Chaos Basis	271
Modeling, Random Field: Add Triple Product of Hermite Polynomial Chaos Basis Using HDF5 Input	273
Modeling, Random Field: Add Double Product of Hermite Polynomial Chaos Basis Using HDF5 Input	275
Modeling, Solid-Fluid Interaction: Adding Solid-Fluid Interface	277
Modeling, Solid-Fluid Interaction: Defining Solid-Fluid Interface, ESSI Element Nodes	278
Modeling, Solid-Fluid Interaction: Defining Solid-Fluid Interface, ESSI Element Faces .	279
Modeling, Solid-Fluid Interaction: Defining Solid-Fluid Interface FOAM Nodes	281
Modeling, Solid-Fluid Interaction: Defining Solid-Fluid Interface FOAM Faces	282
1.3.5 Simulation	284
Simulation, Solvers: Sequential Solvers	285
Simulation, Solvers: Parallel Solvers	286
Simulation: Static Solution Advancement	289
Simulation: Dynamic Solution Advancement with the Constant Time Step	290
Simulation: Dynamic Solution Advancement with Variable Time Step	291
Simulation: Generalized Eigenvalue Analysis	292
Simulation: Displacement Control	293
Simulation: Load, Control, Factor Increment	294
Simulation: Dynamic Integrator, Newmark Method	295
Simulation: Dynamic Integrator, Hilber Hughes Taylor, HHT, α Method	296
Simulation: Absolute Convergence Criteria	297
Simulation: Average Convergence Criteria	298
Simulation: Relative Convergence Criteria	299

Simulation: Solution Algorithms	300
Simulation: Constitutive Integration Algorithm	301
Simulation: Status Check	303
Simulation: Save State	304
Simulation: Restart Simulation	305
Simulation: Return Value for <code>simulate</code> Command	306
Simulation: New Elastic Loading Case	316
Simulation: Combine Elastic Load Cases	317
Simulation, Dynamic Solution Advancement for Solid-Fluid Interaction	318
Simulation, Dynamic Solution Advancement for Stochastic Finite Element Method	319
Simulation, Sobol Sensitivity Analysis	320
Simulation: 3D 3C Wave Field Inversion	324
1.3.6 Output Options	326
Output Options: Enable/Disable Output	327
Output Options: Enable/Disable Element Output	328
Output Options: Enable/Disable Displacement Output	329
Output Options: Enable/Disable Acceleration Output	330
Output Options: Enable/Disable Asynchronous Output	331
Output Options: Output Every n Steps	332
Output Options: Output Support Reactions	333
1.4 Checking the Model	334
1.5 Constitutive Testing	335
1.6 List of Available Commands (tentative, not up to date)	337
1.7 List of reserved keywords	347
1.8 Integrated Development Environment (IDE) for DSL	360
1.9 Mesh Generation using GiD	361
1.10 Model Development and Mesh Generation using gmesh	362
1.11 Model Input File Editing using Sublime	363

Chapter 1

Input, Domain Specific Language (DSL)

1991-2005-2010-2011-2012-2015-2016-2017-2020-2021-

1.1 Chapter Summary and Highlights

1.2 Introduction

This chapter presents the domain specific language developed for the Real-ESSI. The language was designed with a primary goal of developing FEA models and interfacing them with various Real-ESSI functionalities. In addition to that, syntax is used to self-document models, provide physical-unit safety, provide common flow control structures, provide modularity to scripting via user functions and “include” files, and provide an interactive environment within which models can be created, validated and verified.

The development of Real-ESSI Domain specific language (DSL) (the Finite Element Interpreter, $\text{\textcircled{F}}$) is based on LEX (Lesk and Schmidt, 1975) and YACC (Johnson, 1975).

Self-documenting ensures that the resulting model script is readable and understandable with little or no reference to the users manual. This is accomplished by providing a command grammar structure and wording similar to what would be used in a natural language description of the problem.

FEA analysis is unitless, that is, all calculations are carried out without referencing a particular unit system. This leaves the task of unit correctness up to the user of FEA analysis. This represents a recurring source of error in FEA analysis. Physical unit safety is enforced in Real-ESSI by implementing all base variables as physical quantities, that is, all variables have a unit associated with it. The adimensional unit is the base unit for those variables which have no relevant unit (like node numbers). Command calls are sensitive to units. For example, the node creation command call expects the node coordinates to be input with the corresponding units (length in this case). Additionally, the programming/command language naturally supports operation with units like arithmetic operations (quantities with different unit types will not add or subtract but may be multiplied). This approach to FEA with unit awareness provides an additional layer of security to FEA calculations, and forces the user to carefully think about units. This can help catch some common mistakes.

The Real-ESSI language provides modularity through the `include` directive/command, and user functions. This allows complex analysis cases to be parameterized into modules and functions which can be reused in other models.

Finally, an emphasis is placed on model verification and validation. To this end, Real-ESSI provides an interactive programming environment with all the ESSI syntax available. By using this environment. the user can develop tests to detect errors in the model that are not programming errors. For example, the user can query nodes and elements to see if they are set to appropriate states. Also, several standard tools are provided to check element validity (Jacobian, etc.).

The ESSI language provides reduced model development time by providing the aforementioned features along with meaningful error reporting (of syntax and grammatical errors), a help system, command completion and highlighting for several open source and commercial text editors.

Some additional ideas are given by [Dmitriev \(2004\)](#), [Stroustrup \(2005\)](#), [Niebler \(2005\)](#), [Mernik et al. \(2005\)](#), [Ward \(2003\)](#), etc.

1.3 Domain Specific Language (DSL), English Language Binding

Overview of the language syntax.

- Each command line has to end with a semicolon ";"
- Comment on a line begins with either "//" or "!" and last until the end of current line.
- Units are required (see more below) for all quantities and variables.
- Include statements allow splitting source into several files
- All variables are double precision (i.e. floats) with a unit attached.
- All standard arithmetic operations are implemented, and are unit sensitive.
- Internally, all units are represented in the base SI units ($m - s - kg$).
- The syntax ignores extra white spaces, tabulations and newlines. Wherever they appear, they are there for code readability only. (This is why all commands need to end with a semicolon).
- The user should be familiar with the list of the reserved keywords from [Section 1.7](#) on page [347](#).

1.3.1 Running Real-ESSI

At the command line type "essi", to get to the ESSI prompt and start Real-ESSI in interactive mode.

Command line output

```
The Finite Element Interpreter Endeavor

The Real-ESSI Simulator
Modeling and Simulation of Earthquakes, and Soils, and Structures ←
and their Interaction

Sequential processing mode.

Version Name      : Real-ESSI Global Release, June2018. Release date: ←
Jun 13 2018 at 11:02:19. Tag: adc085ae70
Version Branch    : GLOBAL_RELEASE
Compile Date      : Jun 13 2018 at 14:36:56
Compile User      : jeremic
Compile Sysinfo   : sokocalo 4.13.0-43-generic x86_64 GNU/Linux
```

```

Runtime User      : jeremic
Runtime Sysinfo   : sokocalo 4.13.0-43-generic x86_64 GNU/Linux
Time Now          : Jun 13 2018 at 15:32:52
Days From Release : 0
PostProcessing Compatible Version: ParaView 5.1.2
PostProcessing Compatible Version: ESSI-pvESSI Date: Feb 15 2018 at ↵
11:00:28. Tag: 58fe430a19

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.

ESSI>

```

A number of useful information about Real-ESSI is printed on the screen. From here, commands can be input manually or a file may be included via the `include` command which is as follows.

```
1 include "foobar.fei";
```

to include the file `foobar.fei`.

A more efficient way to start Real-ESSI and analyze an example is to pass input file name to the command line. Real-ESSI command to execute an input file immediately is done by issuing the following command: `essi -f foobar.fei`. This will execute `essi` directly on input file `foobar.fei`. After executing the file, the `essi` interpreter will continue in interactive mode unless the command line flag `-n` or `--no-interactive` is set. A list of command line options is available by calling `essi` from the command line as `essi -h`.

Command line output

```

The Finite Element Interpreter Endeavor

The Real-ESSI Simulator
Modeling and Simulation of Earthquakes, and Soils, and Structures ↵
and their Interaction

Sequential processing mode.

Version Name      : Real-ESSI Global Release, June2018. Release date: ↵
Jun 13 2018 at 11:02:19. Tag: adc085ae70
Version Branch    : GLOBAL_RELEASE
Compile Date      : Jun 13 2018 at 14:36:56
Compile User      : jeremic
Compile Sysinfo   : sokocalo 4.13.0-43-generic x86_64 GNU/Linux
Runtime User      : jeremic
Runtime Sysinfo   : sokocalo 4.13.0-43-generic x86_64 GNU/Linux
Time Now          : Jun 13 2018 at 16:22:08
Days From Release : 0
PostProcessing Compatible Version: ParaView 5.1.2

```

```
PostProcessing Compatible Version: ESSI-pvESSI Date: Feb 15 2018 at ↵
11:00:28. Tag: 58fe430a19
```

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.

The Real-ESSI Simulator

Modeling and Simulation of Earthquakes, and Soils, and Structures and ↵
their Interaction

Usage: essi [-cfhnsmb FILENAME]

```
-c --cpp-output           : Output cpp version of the model.
-f --filename [FILENAME] : run ESSI on a FILENAME.
-h --help                : Print this message.
-n --no-interactive       : Disable interactive mode.
-s --set-variable        : Set a variable from the command line.
-d --dry-run             : Do not execute ESSI API calls. ↵
  Just parse.
-m --model-name [NAME]   : Set the model name from the ↵
  command line.
-p --profile-report [FILENAME] : Set the filename for the profiler ↵
  report (and activate lightweight profiling)
```

Example to set a variable name from command line:

```
essi -s a=10,b=20,c=30
```

Runs ESSI with variables a, b, and c set to 10, 20 and 30 respectively.
At this time, only ESSIunits::unitless variables can be set.

1.3.2 Finishing Real-ESSI Program Run

To properly finish Real-ESSI program run, and save and close all the output files, user has to use final, closure command:

```
1  bye;
```

Command `bye;` has to be included at the end of input file script, or at the end of each interactive/interpretative session. Command `bye;` ensures that Real-ESSI program gracefully exits simulation, and that all the output files are properly saved and closed. Proper finishing of simulation using Real-ESSI Simulator is very much necessary, while the choice of command `bye;` is done as an homage to Professor Knuth and his Literate Programming endeavor ([Knuth, 1984](#)), that is driving much of the Real-ESSI DSL development.

There are a number of alternative final commands, for example:

```
1  exit;
2  quit;
3  zdravo;
```

```

4  vozdra;
5  dvojka;
6  voljno;
7  zaijian;
8  tschuess;
9  geia-sou;
10 tchau;
11 sair;
12 khoda-hafez;
13 doi;
14 nasvidenje;
15 ajde-bok;
16 izhod;
17 konec;
18 czesc;
19 ciao;
20 hoscakal;

```

These additional, alternative final commands can all be written using original scripts:

zdravo ↔ здраво

vozdra ↔ воздра

dvojka ↔ двојка

voljno ↔ вољно

zaijian ↔ 再见

tschuess ↔ tschüss

geia-sou ↔ γεια σου

khoda-hafez ↔ خداحافظ

hoscakal ↔ hoşçakal

1.3.3 Real-ESSI Variables, Basic Units and Flow Control

Variables are defined using the assignment (=) operator. For example,

```

1  var_x    =  7;          //Results in the variable x be set to 7 (unitless)
2  var_y    =  3.972e+2;   //Scientific notation is available.

```

The language contains a list of reserved keywords. Throughout this documentation, reserved keywords are highlighted in **blue** or **red**.

All standard arithmetic operations are available between variables. These operations can be combined arbitrarily and grouped together with parentheses.

```

1 var_a = var_x + var_y;      // Addition
2 var_b = var_x - var_y;      // Subtraction
3 var_c = var_x * var_y;      // Product
4 var_d = var_x / var_y;      // Quotient
5 var_e = var_y % var_x;      // Modulus (how many times x fits in y)

```

The 'print' command can be used to display the current value of a variable.

```

1 print var_x;
2 print var_y;
3 print var_a;
4 print var_b;
5 print var_c;
6 print var_d;
7 print var_e;

```

Command line output

```

var_x = 7 []
var_y = 397.2 []
var_a = 404.2 []
var_b = -390.2 []
var_c = 2780.4 []
var_d = 0.0176234 []
var_e = 5.2 []

```

Here the “unit” (sign) [] means that the quantities are unitless.

The command 'whos' is used to see all the currently defined variables and their values. After a fresh start of essi, needed to clear up all the previously defined variables, command whos;' produces a list of predefined variables:

Command line output

```

ESSI> whos;

Declared variables:
*      Day =          86400 [s]
*      GPa =           1 [GPa]
*      Hour =         3600 [s]
*      Hz =            1 [Hz]
*      MPa =           1 [MPa]
*      Minute =        60 [s]
*      N =             1 [N]
*      Pa =            1 [Pa]
*      Week =        604800 [s]
*      cm =            1 [cm]
*      feet =         0.3048 [m]
*      ft =            0.3048 [m]

```

```

*      g =          9.81 [m*s^-2]
*      inch =        0.0254 [m]
*      kN =          1 [kN]
*      kPa =          1 [kPa]
*      kg =          1 [kg]
*      kip =        4448.22 [N]
*      km =          1 [km]
*      ksi =        6.89476e+06 [Pa]
*      lbf =         4.44822 [N]
*      lbm =         0.453592 [kg]
*      m =           1 [m]
*      mile =        1609.35 [m]
*      mm =           1 [mm]
*      pi =          3.14159 []
*      psi =         6894.76 [Pa]
*      s =            1 [s]
*      yard =         0.9144 [m]

```

```

* = locked variable
ESSI>

```

Predefined variables shown above have a preceding asterisk to show they are locked variables which cannot be modified. The purpose of these locked variables are to provide names for units. Imperial units are also supported as shown above.

The units for variable are shown between the brackets. Note that unit variables have the same name as their unit, which is not the case for user defined variables. Variables preceded by a star (*) are locked variables which can't be modified.

For example, the variable 'm' defines 'meter'. So to define a new variable L1 which has meter units we do:

```

1 L1 = 1*m;      // Defines L1 to 1 m.
2 L2 = 40*mm;    // Defines L2 to be 40 millimeters.

```

Even though L2 was created with millimeter units, it is stored in base units.

```
print L2; displays
```

Command line output

```
L2 = 0.04 [m]
```

As additional examples, let us define few forces:

```

1 F1 = 10*kN;
2 F2 = 300*N;
3 F3 = 4*kg*g;

```

Here g is the predefined acceleration due to gravity.

Arithmetic operations do check (and enforce) for unit consistency. For example, `foo = L1 + F1;` produces an error because units are not compatible. However, `bar = L1 + L2;` is acceptable. On the other hand,

multiplication, division and modulus, always work because the result produces a quantity with new units (except when the adimensional quantity is involved).

```
1 A = L1*L2;
2 Stress_n = F1 / A;
```

Units for all variables are internally converted to SI units ($kg - m - s$) and stored in that unit system. Variables can be *displayed* using different units by using the [] operator. This does not change the variable, it just displays the value of variable with required unit. For example,

```
1 print Stress_n;           //Print in base SI units.
2 print Stress_n in Pa;     //Print in Pascal
3 print Stress_n in kPa;    //Print in kilo Pascal
```

Command line output

```
Stress_n = 250000 [kg*m^-1*s^-2]

Stress_n = 250000 [Pa]

Stress_n = 250 [kPa]
```

The DSL provides functions to test the physical units of variables. For example,

```
print isForce(F1);
```

Will print an adimensional, Boolean 1 because F1 has units of force. While,

```
print isPressure(F);
```

will print an adimensional, Boolean 0. The language also provides comparison of quantities with same units (remember all values are compared in SI Units).

```
print F1 > F2;
```

will print an adimensional, Boolean 1 since F1 is greater than F2.

The program flow can be controlled with `if` and `while` statements, i.e.:

```
1 if (isForce(F1))
2 {
3   print F1;    // This will be executed
4 };
5
6 if (isForce(L1))
7 {
8   print L1;    // This will not.
9 };
```

Note the necessary semicolon (;) at the closing brace. Unlike C/C++, the braces are always necessary. Closing colon is also always necessary.

The “else” statement is also available:

```

1  if (isForce(L1))
2  {
3    print L1;      // This will not execute
4  }
5  else
6  {
7    print L2;      // This will execute instead
8  };

```

While loops are also available:

```

1  i = 0;
2  while( i < 10)
3  {
4    print i;
5    i = i +1;
6  };

```

1.3.4 Modeling

This section details ESSI modeling commands. Angle brackets $\langle \rangle$ are used for quantity or variable placeholder, that is, they indicate where user input goes. Within the angle brackets, the expected unit type is given as well, i.e., $\langle L \rangle$ means the command expects an input with a value and a length unit. The symbol $\langle . \rangle$ represents the adimensional quantity.

In addition to that, the vertical bar $|$ (“OR” sign)) is used to separate two or more keyword options, i.e. $[a|b|c]$ is used indicate keyword options a or b or c. The symbol $|\dots|$ is used to denote where several long options exist and are explained elsewhere (an example of this is available below in a material model definitions).

All commands require unit consistency. Base units, SI or other can be used as indicated below:

- length, symbol L , units [m, inch, ft]
- mass, symbol M , units [kg, lbm],
- time, symbol T , units [s]

Derived units can also be used:

- angle, symbol rad (radian), unit [*dimensionless*, L/L]
- force, symbol N (Newton), units [N , kN , MN , $M * L/T^2$],
- stress, symbol Pa (Pascal), units [Pa , kPa , MPa , N/L^2 , $M/L/T^2$]

- strain, symbol (no symbol), units $[L/L]$
- mass density, symbol (no symbol), units $[M/L^3]$
- force density, symbol (no symbol), units $[M/L^2/T^2]$

All models have to be named: `model name "model_name_string";` This is important as output files are named based on model name.

Each loading stage has to be named as well. A new loading stage¹ is defined like this:

```
new loading stage "loading stage name string";
```

In addition to model name, loading stage name is used for output file name for given loading stage.

¹See more in section 101.4.5 on page 97 in Jeremić et al. (1989-2025).

Modeling, Material Model: Adding a Material Model to the Finite Element Model

Adding constitutive material model to the finite element model/domain is done using command:

```
1  add material  # <.> type |...|  
2                mass_density = <M/L^3>  
3                (more model dependent parameters) ;
```

- Material number # (or alternatively No) is a distinct integer number used to uniquely identify this material.
- Mass density should be defined for each material (even if only static analysis is performed, for example if self weight is to be used as a loading stage).
- Depending on material model, there will be additional material parameters that are defined for each material model/type below:

Starting with version 03-NOV-2015 all elastic-plastic material models require explicit specification of the constitutive integration algorithm. More information on this can be found in [1.3.5](#). Only the material `linear_elastic_isotropic_3d_LT` ignores this option.

Choices for `material_type` are listed below.

Modeling, Material Model: Linear Elastic Isotropic Material Model

The command is:

```
1  add material # <.> type linear_elastic_isotropic_3d
2      mass_density = <M/L^3>
3      elastic_modulus = <F/L^2>
4      poisson_ratio = <.>;
```

where:

- `mass_density` is the mass density of material [M/L^3]
- `elastic_modulus` is an isotropic modulus of elasticity of a material (units: stress)
- `poisson_ratio` is a Poisson's ratio [dimensionless]

More on this material model can be found in Section [104.6.1](#) on Page [220](#) in Lecture Notes by [Jeremić et al. \(1989-2025\)](#) ([Lecture Notes URL](#)).

Modeling, Material Model: Cross Anisotropic Linear Elastic Material Model

The command is:

```
1  add material    # <.> <material_number>
2                  type linear_elastic_crossanisotropic
3                  mass_density = <M/L^3>
4                  elastic_modulus_horizontal = <F/L^2>
5                  elastic_modulus_vertical = <F/L^2>
6                  poisson_ratio_h_v = <.>
7                  poisson_ratio_h_h = <.>
8                  shear_modulus_h_v = <F/L^2>;
```

where:

- `mass_density` is the mass density of material [M/L^3]
- `elastic_modulus_horizontal` is an anisotropic modulus of elasticity for horizontal plane of a material [F/L^2]
- `elastic_modulus_vertical` is an anisotropic modulus of elasticity for vertical direction of a material [F/L^2]
- `poisson_ratio_h_v` is a Poisson's ratio for horizontal-vertical directions [dimensionless]
- `poisson_ratio_h_h` is a Poisson's ratio for horizontal-horizontal directions [dimensionless]
- `shear_modulus_h_v` is a shear modulus for horizontal-vertical directions [F/L^2]

It is assumed that vertical axes is global Z axes.

More on this material model can be found in Section 104.6.1 on Page 220 in Lecture Notes by Jeremić et al. (1989-2025) ([Lecture Notes URL](#)).

Modeling, Material Model: von Mises Associated Material Model with Linear Isotropic and/or Kinematic Hardening

Implements von Mises family of constitutive models, with linear kinematic and/or isotropic hardening.

The command is:

```
1 add material # <.> type vonMises
2   mass_density = <M/L^3>
3   elastic_modulus = <F/L^2>
4   poisson_ratio = <.>
5   von_mises_radius = <F/L^2>
6   kinematic_hardening_rate = <F/L^2>
7   isotropic_hardening_rate = <F/L^2> ;
```

where:

- mass_density is the mass density of material [M/L^3]
- elastic_modulus is the elastic modulus of material [F/L^2]
- poisson_ratio is the Poisson's ratio material []
- von_mises_radius is the radius of the deviatoric section of the von Mises yield surface [F/L^2]
- kinematic_hardening_rate is the rate of the kinematic hardening [F/L^2]
- isotropic_hardening_rate is the rate of the kinematic hardening [F/L^2]

More on this material model can be found in Section 104.6.6 on Page 226 in Lecture Notes by Jeremić et al. (1989-2025) ([Lecture Notes URL](#)).

Modeling, Material Model: von Mises Associated Material Model with Isotropic Hardening and/or Armstrong-Frederic Nonlinear Kinematic Hardening

This command is for von Mises family of constitutive models, with Armstrong-Frederick kinematic and/or isotropic hardening.

The command is:

```
1 add material # <.> type vonMisesArmstrongFrederick
2   mass_density = <M/L^3>
3   elastic_modulus = <F/L^2>
4   poisson_ratio = <.>
5   von_mises_radius = <.>
6   armstrong_frederick_ha = <F/L^2>
7   armstrong_frederick_cr = <.>
8   isotropic_hardening_rate = <F/L^2> ;
```

where:

- mass_density is the mass density of material [M/L^3]
- elastic_modulus is the elastic modulus of material [F/L^2]
- poisson_ratio is the Poisson's ratio material []
- von_mises_radius is the radius of the deviatoric section of the von Mises yield surface [F/L^2]
- armstrong_frederick_ha controls rate of the kinematic hardening [F/L^2]
- armstrong_frederick_cr controls the saturation limit for kinematic hardening [Dimensionless]
- isotropic_hardening_rate is the rate of the kinematic hardening [F/L^2]

More on this material model can be found in Section 104.6.6 on Page 226 in Lecture Notes by Jeremić et al. (1989-2025) ([Lecture Notes URL](#)).

Modeling, Material Model: Drucker-Prager Associated Material Model with Linear Isotropic and/or Kinematic Hardening

This command is for Drucker-Prager family of constitutive models, with linear kinematic and/or isotropic hardening. This material uses associate plastic flow rule.

The command is:

```
1 add material # <.> type DruckerPrager
2   mass_density = <M/L^3>
3   elastic_modulus = <F/L^2>
4   poisson_ratio = <.>
5   druckerprager_k = <.>
6   kinematic_hardening_rate = <F/L^2>
7   isotropic_hardening_rate = <F/L^2>
8   initial_confining_stress = <F/L^2> ;
```

where:

- mass_density is the mass density of material [M/L^3]
- elastic_modulus is the elastic modulus of material [F/L^2]
- poisson_ratio is the Poisson's ratio material []
- druckerprager_k slope of the Drucker-Prager yield surface in p - q m space (equivalent to M parameter) [dimensionless]
- kinematic_hardening_rate is the rate of the kinematic hardening [F/L^2]
- isotropic_hardening_rate is the rate of the isotropic hardening [F/L^2]

More on this material model can be found in Section 104.6.7 on Page 232 in Lecture Notes by Jeremić et al. (1989-2025) ([Lecture Notes URL](#)).

Modeling, Material Model: Drucker-Prager Associated Material Model with Isotropic Hardening and/or Armstrong-Frederick Nonlinear Kinematic Hardening

A Drucker-Prager constitutive model with associative plastic-flow rule, Armstrong-Frederick kinematic hardening, and linear isotropic hardening and linear elastic isotropic elasticity law.

The command is:

```

1 add material # <.> type DruckerPragerArmstrongFrederickLE
2   mass_density = <M/L^3>
3   elastic_modulus = <F/L^2>
4   poisson_ratio = <.>
5   druckerprager_k = <.>
6   armstrong_frederick_ha = <F/L^2>
7   armstrong_frederick_cr = <.>
8   isotropic_hardening_rate = <F/L^2>
9   initial_confining_stress = <F/L^2>;

```

where:

- `mass_density` is the mass density of material [M/L^3]
- `elastic_modulus` is the elastic modulus of material [F/L^2]
- `poisson_ratio` is the Poisson's ratio material []
- `druckerprager_k` slope of the Drucker-Prager yield surface in p - q m space (equivalent to M parameter) [Dimensionless]
- `armstrong_frederick_ha` controls rate of the kinematic hardening [F/L^2]
- `armstrong_frederick_cr` controls the saturation limit for kinematic hardening [Dimensionless]
- `isotropic_hardening_rate` is the rate of the isotropic hardening [F/L^2]
- `initial_confining_stress` initial confining (mean) pressure [F/L^2]

More on this material model can be found in Section 104.6.7 on Page 232 in Lecture Notes by Jeremić et al. (1989-2025) ([Lecture Notes URL](#)).

Modeling, Material Model: Drucker-Prager Associated Material Model with Isotropic Hardening and/or Armstrong-Frederick Nonlinear Kinematic Hardening and Nonlinear Duncan-Chang Elasticity

A Drucker-Prager constitutive model with associative plastic-flow rule, Armstrong-Frederick kinematic hardening, and Duncan-Chang non-linear isotropic elasticity law.

The command is:

```

1 add material # <.> type DruckerPragerArmstrongFrederickNE
2   mass_density = <M/L^3>
3   DuncanChang_K = <.>
4   DuncanChang_pa = <F/L^2>
5   DuncanChang_n = <.>
6   DuncanChang_sigma3_max = <F/L^2>
7   DuncanChang_nu = <.>
8   druckerprager_k = <.>
9   armstrong_frederick_ha = <F/L^2>
10  armstrong_frederick_cr = <.>
11  isotropic_hardening_rate = <F/L^2>
12  initial_confining_stress = <F/L^2>;

```

where:

- mass_density is the mass density of material [M/L^3]
- DuncanChang_K parameter controlling Young's modulus [$< . >$]
- DuncanChang_pa reference pressure [F/L^2]
- DuncanChang_n exponent [$< . >$]
- DuncanChang_sigma3_max maximum value for σ_3 ($\sigma_3 < 0$) elastic properties are constant for greater values of σ_3 [F/L^2]
- DuncanChang_nu Poisson's ratio [F/L^2]
- druckerprager_k slope of the Drucker-Prager yield surface in p - q m space (equivalent to M parameter) [Dimensionless]
- armstrong_frederick_ha controls rate of the kinematic hardening [F/L^2]
- armstrong_frederick_cr controls the saturation limit for kinematic hardening [Dimensionless]
- isotropic_hardening_rate is the rate of the isotropic hardening [F/L^2]
- initial_confining_stress initial confining (mean) pressure [F/L^2]

More on this material model can be found in Section 104.6.7 on Page 232 in Lecture Notes by Jeremić et al. (1989-2025) ([Lecture Notes URL](#)).

Modeling, Material Model: Drucker-Prager Nonassociated Material Model with Linear Isotropic and/or Kinematic Hardening

This command defines Drucker-Prager family of constitutive models, with linear kinematic and/or isotropic hardening. This material uses non-associate plastic flow rule.

The command is:

```

1 add material # <.> type DruckerPragerNonAssociateLinearHardening
2   mass_density = <M/L^3>
3   elastic_modulus = <F/L^2>
4   poisson_ratio = <.>
5   druckerprager_k = <.>
6   kinematic_hardening_rate = <F/L^2>
7   isotropic_hardening_rate = <F/L^2>
8   initial_confining_stress = <F/L^2>
9   plastic_flow_xi = <.>
10  plastic_flow_kd = <.>;

```

where:

- `mass_density` is the mass density of material [M/L^3]
- `elastic_modulus` is the elastic modulus of material [F/L^2]
- `poisson_ratio` is the Poisson's ratio material []
- `druckerprager_k` slope of the Drucker-Prager yield surface in p - q m space (equivalent to M parameter) [Dimensionless]
- `kinematic_hardening_rate` is the linear rate of the kinematic hardening [F/L^2]
- `isotropic_hardening_rate` is the linear rate of the isotropic hardening [F/L^2]
- `initial_confining_stress` initial confining (mean) pressure [F/L^2]
- `plastic_flow_xi` governs the amplitude of plastic volume changes. The higher ξ , the higher the dilatancy. If $\xi = 0$, the material model will only produce deviatoric plastic strains. [.]
- `plastic_flow_kd` governs the size of the dilatancy surface, a cone in the stress space on which no plastic volume changes occur. k_d governs the size of this cone: if k_d is equal to zero, the dilatancy surface shrinks to a line (the hydrostatic axis), so that only dilative soil deformation is possible. [.]

More on this material model can be found in Section 104.6.7 on Page 232 in Lecture Notes by Jeremić et al. (1989-2025) ([Lecture Notes URL](#)).

Modeling, Material Model: Drucker-Prager Nonassociated Material Model with Linear Isotropic and/or Armstrong-Frederick Nonlinear Kinematic Hardening

This command defines Drucker-Prager family of constitutive models, with nonlinear kinematic and/or linear isotropic hardening. This material uses non-associated plastic flow rule.

The command is:

```

1  add material # <.> type DruckerPragerNonAssociateArmstrongFrederick
2      mass_density = <M/L^3>
3      elastic_modulus = <F/L^2>
4      poisson_ratio = <.>
5      druckerprager_k = <.>
6      armstrong_frederick_ha = <F/L^2>
7      armstrong_frederick_cr = <.>
8      isotropic_hardening_rate = <F/L^2>
9      initial_confining_stress = <F/L^2>
10     plastic_flow_xi = <.>
11     plastic_flow_kd = <.>;

```

where:

- `mass_density` is the mass density of material [M/L^3]
- `elastic_modulus` is the elastic modulus of material [F/L^2]
- `poisson_ratio` is the Poisson's ratio material []
- `druckerprager_k` slope of the Drucker-Prager yield surface in p - q m space (equivalent to M parameter) [Dimensionless]
- `armstrong_frederick_ha` a kinematic hardening parameter, which governs the initial stiffness after the yield [F/L^2]
- `armstrong_frederick_cr` a kinematic hardening parameter. $\frac{h_a}{c_r}$ governs the limit of the back-stress [Dimensionless]
- `isotropic_hardening_rate` is the rate of the kinematic hardening [F/L^2]
- `initial_confining_stress` initial confining (mean) pressure [F/L^2]
- `plastic_flow_xi` governs the amplitude of plastic volume changes - the higher ξ , the higher the dilatancy. If $\xi = 0$, the material model will only produce deviatoric plastic strains. [.]
- `plastic_flow_kd` governs the size of the dilatancy surface, a cone in the stress space on which no plastic volume changes occur. k_d governs the size of this cone: if k_d is equal to zero, the dilatancy surface shrinks to a line (the hydrostatic axis), so that only dilative soil deformation is possible. [.]

More on this material model can be found in Section 104.6.7 on Page 232 in Lecture Notes by Jeremić et al. (1989-2025) (Lecture Notes URL).

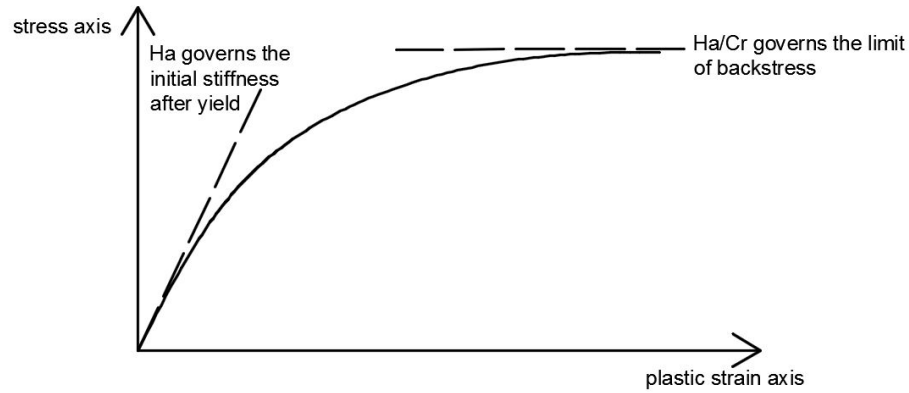


Figure 1.1: The physical meanings of h_a and c_r

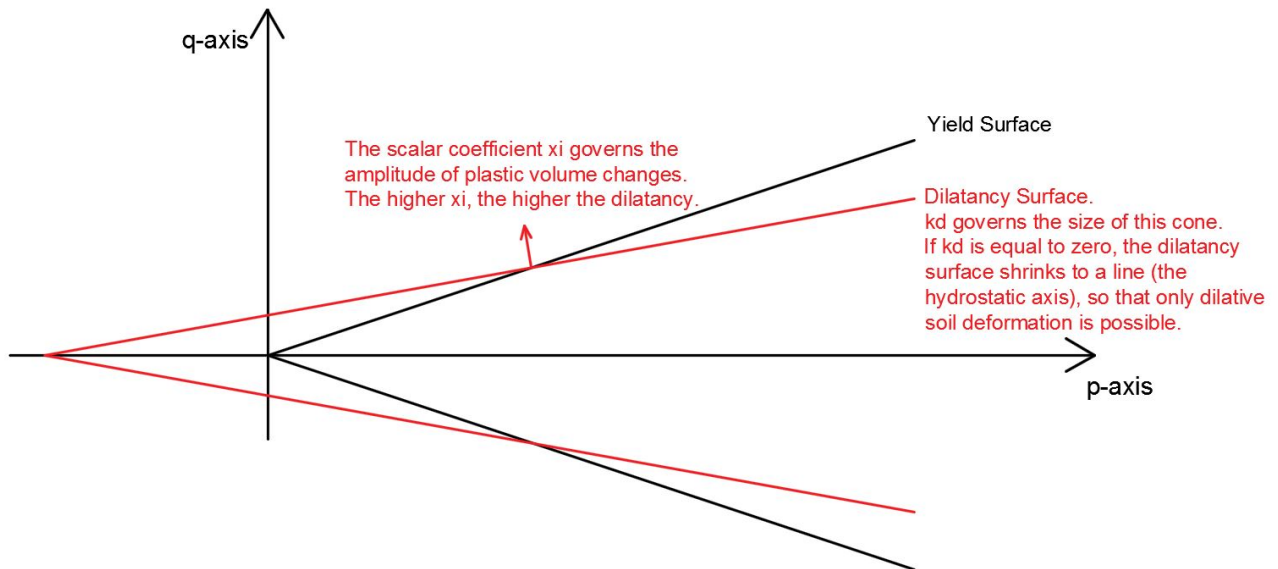


Figure 1.2: The physical meanings of ξ and k_d

The physical meanings of h_a , c_r , ξ , and k_d are shown in Figure (1.1) and Figure (1.2).

Modeling, Material Model: Hyperbolic Drucker-Prager Nonassociated Material Model with Linear Isotropic and/or Armstrong-Frederick Nonlinear Kinematic Hardening

This command defines a hyperbolic Drucker-Prager constitutive model, with nonlinear kinematic and/or linear isotropic hardening. This material uses non-associated plastic flow rule.

The command is:

```

1  add material # <.> type ↵
    HyperbolicDruckerPragerNonAssociateArmstrongFrederick
2      mass_density = <M/L^3>
3      elastic_modulus = <F/L^2>
4      poisson_ratio = <.>
5      friction_angle = <.>
6      cohesion = <F/L^2>
7      rounded_distance = <F/L^2>
8      armstrong_frederick_ha = <F/L^2>
9      armstrong_frederick_cr = <.>
10     isotropic_hardening_rate = <F/L^2>
11     initial_confining_stress = <F/L^2>
12     plastic_flow_xi = <.>
13     plastic_flow_kd = <.>;

```

where:

- `mass_density` is the mass density of material [M/L^3]
- `elastic_modulus` is the elastic modulus of material [F/L^2]
- `poisson_ratio` is the Poisson's ratio for material []
- `friction_angle` is the initial friction angle of the material. If isotropic hardening is present, friction angle will evolve. [rad]
- `cohesion` is a material constant that defines the cohesion of the material [F/L^2]
- `rounded_distance` is the parameter that controls the shape of the rounded apex of yield surface [F/L^2]
- `armstrong_frederick_ha` a kinematic hardening parameter, that governs the initial stiffness after the yield [F/L^2]
- `armstrong_frederick_cr` a kinematic hardening parameter. It is noted that ratio $\frac{h_a}{c_r}$ controls the asymptote the back-stress, that can be related to the ultimate shear strength [.]
- `isotropic_hardening_rate` is the rate of the isotropic hardening [F/L^2]
- `initial_confining_stress` initial confining (mean) pressure, a small value just get initial stress out of cone zone [F/L^2]

- `plastic_flow_xi` governs the amplitude of plastic volume changes - the higher ξ , the higher the dilatancy. If $\xi = 0$, the material model will only produce deviatoric plastic strains. [.]
- `plastic_flow_kd` governs the size of the dilatancy surface, a cone in the stress space on which no plastic volume changes occur. k_d governs the size of this cone: if k_d is equal to zero, the dilatancy surface shrinks to a line (the hydrostatic axis), so that only dilative soil deformation is possible. [.]

More on this material model can be found in Section 104.6.8.5 on Page 247 in Lecture Notes by Jeremić et al. (1989-2025) ([Lecture Notes URL](#)).

Modeling, Material Model: Rounded Mohr-Coulomb Associated Linear Isotropic Hardening Material Model

The command is:

```

1  add material # <.> type roundedMohrCoulomb
2      mass_density = <M/L^3>
3      elastic_modulus = <F/L^2>
4      poisson_ratio = <.>
5      RMC_m = <.>
6      RMC_qa = <F/L^2>
7      RMC_pc = <F/L^2>
8      RMC_e = <.>
9      RMC_eta0 = <.>
10     RMC_Heta = <F/L^2>
11     initial_confining_stress = <F/L^2>
12

```

where

- `mass_density` is the mass density of material [M/L^3]
- `elastic_modulus` is the elastic modulus of material [F/L^2]
- `poisson_ratio` is the Poisson's ratio material []
- `RMC_m` $0 < m < 1$ parameter of the RMC yield function. Controls roundness of apex in p - q space. []
- `RMC_qa` q_a parameter of the RMC yield function. Controls roundness of apex in p - q space. [F/L^2]
- `RMC_pc` p pressure offset [F/L^2]
- `RMC_e` e parameter controls roundness of the deviatoric cross-section of the yield surface. $0.5 < e \leq 1$, $e = 0.5$ results in a triangular deviatoric section while $e = 1$ is round. []
- `RMC_eta0` controls the opening of the yield surface []
- `RMC_Heta` isotropic (linear) hardening of the yield surface [F/L^2]
- `initial_confining_stress` initial confining (mean) pressure [F/L^2]

More on this material model can be found in Section 104.6.9 on Page 250 in Lecture Notes by Jeremić et al. (1989-2025) ([Lecture Notes URL](#)).

Modeling, Material Model: Cam Clay Material Model

The command is:

```
1  add material # <.> type CamClay
2    mass_density = <M/L^3>
3    M = <.>
4    lambda = <.>
5    kappa = <.>
6    e0 = <.>
7    p0 = <F/L^2>
8    poisson_ratio = <.>
9    initial_confining_stress = <F/L^2>
```

where

- mass_density is the mass density of material $[M/L^3]$
- e0 void ratio (e_0) at the reference pressure, [dimensionless]
- M Cam-Clay slope of the critical state line in stress space, [dimensionless]
- lambda Cam-Clay normal consolidation line slope, (unit: dimensionless)
- kappa Cam-Clay unload-reload line slope, (unit: dimensionless)
- poisson_ratio Constant Poisson-ratio
- p0 Cam-Clay parameter (p_0). Tip of the yield surface in q - p space. $[F/L^2]$
- initial_confining_stress initial confining (mean) pressure $[F/L^2]$

More on this material model can be found in Section [104.6.10](#) on Page [251](#) in Lecture Notes by [Jeremić et al. \(1989-2025\)](#) ([Lecture Notes URL](#)).

Modeling, Material Model: von Mises Associated Multiple Yield Surface Material Model

The command is:

```

1      add material # <.> type vonMisesMultipleYieldSurface
2      mass_density = <M/L^3>
3      elastic_modulus = <F/L^2>
4      poisson_ratio = <.>
5      total_number_of_yield_surface = <.>
6      radiuses_of_yield_surface = <string>
7      radiuses_scale_unit = <F/L^2>
8      hardening_parameters_of_yield_surfaces = <string>
9      hardening_parameters_scale_unit = <F/L^2> ;
10

```

where

- `mass_density` is the mass density of material $[M/L^3]$
- `elastic_modulus` is the elastic modulus of the material $[F/L^2]$
- `poisson_ratio` is the constant Poisson-ratio [dimensionless]
- `total_number_of_yield_surface` is the total number of yield surfaces. [dimensionless]
- `radiuses_of_yield_surface` is the radius list of multiple yield surfaces. This parameter gives the radiuses of each yield surface from the smallest to the biggest. This parameter should be a string which contains the dimensionless radiuses. The radiuses should be separated by a blank space or a comma. [string]
- `radiuses_scale_unit` is the unit of the each yield surface. This parameter also provides a method to scale up or scale down the radiuses of each yield surfaces. $[F/L^2]$
- `hardening_parameters_of_yield_surfaces` is the hardening parameters corresponding to each yield surface. This parameter should be a string which contains the dimensionless hardening parameters. The hardening parameters should be separated by a blank space or a comma. [string]
- `hardening_parameters_scale_unit` The unit of the each hardening parameter. This parameter also provides a method to scale up or scale down the hardening parameter of each yield surfaces. $[F/L^2]$

Modeling, Material Model: von Mises Associated Multiple Yield Surface Material Model that Matches G/G_{max} Curves

The command is:

```

1  add material # <.> type vonMisesMultipleYieldSurfaceGoverGmax
2  mass_density = <M/L^3>
3  initial_shear_modulus = <F/L^2>
4  poisson_ratio = <.>
5  total_number_of_shear_modulus = <.>
6  GoverGmax = <string>
7  ShearStrainGamma = <string> ;

```

Command Example is

```

1  add material # 1 type vonMisesMultipleYieldSurfaceGoverGmax
2  mass_density = 0.0*kg/m^3
3  initial_shear_modulus = 3E8 * Pa
4  poisson_ratio = 0.0
5  total_number_of_shear_modulus = 9
6  GoverGmax =
7  "1,0.995,0.966,0.873,0.787,0.467,0.320,0.109,0.063"
8  ShearStrainGamma =
9  "0,1E-6,1E-5,5E-5,1E-4, 0.0005, 0.001, 0.005, 0.01";

```

where

- `mass_density` is the mass density of material [M/L^3]
- `initial_shear_modulus` is the initial maximum shear modulus, namely, the G_{max} . [F/L^2]
- `poisson_ratio` is the constant Poisson-ratio. [dimensionless]
- `total_number_of_shear_modulus` is the total number of shear modulus, including the initial maximum shear modulus. The total number of yield surface is one less than the total number of shear modulus. Namely, $(N+1)$ areas are divided by N surfaces. [dimensionless]
- `GoverGmax` is the G/G_{max} from experiments, including the initial shear modulus. Namely, the first element should be 1.0. Each element is dimensionless. The input should be separated by a blank space or a comma. [string]
- `ShearStrainGamma` is the shear strain γ corresponding to the `GoverGmax`. Note that $\gamma = 2\varepsilon$ when the input is prepared. The first element should be 0.0 corresponding to the initial shear modulus. Each element is dimensionless. The input should be separated by a blank space or a comma. [string]

Modeling, Material Model: Drucker-Prager Nonassociated Multi-Yield Surface Material Model

The command is:

```

1  add material # <.> type DruckerPragerMultipleYieldSurface
2  mass_density = <M/L^3>
3  elastic_modulus = <F/L^2>
4  poisson_ratio = <.>
5  initial_confining_stress = <F/L^2>
6  reference_pressure = <F/L^2>
7  pressure_exponential_n = <.>
8  cohesion = <F/L^2>
9  dilation_angle_eta = <.>
10 dilation_scale = <.>
11 total_number_of_yield_surface = <.>
12 sizes_of_yield_surfaces = <string>
13 yield_surface_scale_unit = <F/L^2>
14 hardening_parameters_of_yield_surfaces = <string>
15 hardening_parameters_scale_unit = <F/L^2>;

```

where

- `mass_density` is the mass density of material [M/L^3]
- `elastic_modulus` is the elastic modulus of the material [F/L^2]
- `poisson_ratio` is the constant Poisson-ratio [dimensionless]
- `initial_confining_stress` is the initial confining (mean) pressure [F/L^2]
- `reference_pressure` is the reference pressure for the initial modulus. This parameter is usually 101kPa. [F/L^2]
- `pressure_exponential_n` is the exponential number of the pressure dependent modulus. [dimensionless]
- `cohesion` is the attraction force is the soil. [F/L^2]
- `dilation_angle_eta` controls the dilation and compaction of the material. When the stress ratio is smaller than this parameter, plastic compaction takes place. When the stress ratio is greater than this parameter, the plastic dilation takes place. [dimensionless]
- `dilation_scale` controls the rate of the dilation or compaction in the plastic flow. [dimensionless]
- `total_number_of_yield_surface` is the total number of yield surfaces. [dimensionless]

- `radiuses_of_yield_surface` is the radius list of multiple yield surfaces. This parameter gives the radiuses of each yield surface from the smallest to the biggest. This parameter should be a string which contains the dimensionless radiuses. The radiuses should be separated by a blank space or a comma. [string]
- `radiuses_scale_unit` is the unit of the each yield surface. This parameter also provides a method to scale up or scale down the radiuses of each yield surfaces. [F/L^2]
- `hardening_parameters_of_yield_surfaces` is the hardening parameters corresponding to each yield surface. This parameter should be a string which contains the dimensionless hardening parameters. The hardening parameters should be separated by a blank space or a comma. [string]
- `hardening_parameters_scale_unit` The unit of the each hardening parameter. This parameter also provides a method to scale up or scale down the hardening parameter of each yield surfaces. [F/L^2]

Modeling, Material Model: Drucker-Prager Nonassociated Material Model that Matches G/G_{max} Curves

The command is:

```

1  add material # <.> type DruckerPragerMultipleYieldSurfaceGoverGmax
2  mass_density = <M/L^3>
3  initial_shear_modulus = <F/L^2>
4  poisson_ratio = <.>
5  initial_confining_stress = <F/L^2>
6  reference_pressure = <F/L^2>
7  pressure_exponential_n = <.>
8  cohesion = <F/L^2>
9  dilation_angle_eta = <.>
10 dilation_scale = <.>
11 total_number_of_shear_modulus = <.>
12 GoverGmax = <string>
13 ShearStrainGamma = <string>

```

Command Example is

```

1  add material # 1 type DruckerPragerMultipleYieldSurfaceGoverGmax
2  mass_density = 0.0*kg/m^3
3  initial_shear_modulus = 3E8 * Pa
4  poisson_ratio = 0.0
5  initial_confining_stress = 1E5 * Pa
6  reference_pressure = 1E5 * Pa
7  pressure_exponential_n = 0.5
8  cohesion = 0. * Pa
9  dilation_angle_eta = 1.0
10 dilation_scale = 0.0
11 total_number_of_shear_modulus = 9
12 GoverGmax =
13 "1,0.995,0.966,0.873,0.787,0.467,0.320,0.109,0.063"
14 ShearStrainGamma =
15 "0,1E-6,1E-5,5E-5,1E-4, 0.0005, 0.001, 0.005, 0.01";

```

where

- `mass_density` is the mass density of material [M/L^3]
- `elastic_modulus` is the elastic modulus of the material [F/L^2]
- `poisson_ratio` is the constant Poisson-ratio [dimensionless]
- `initial_confining_stress` is the initial confining (mean) pressure [F/L^2]
- `reference_pressure` is the reference pressure for the initial modulus. This parameter is usually 101kPa. [F/L^2]

- `pressure_exponential_n` is the exponential number of the pressure dependent modulus. [dimensionless]
- `cohesion` is the attraction force is the soil. [F/L^2]
- `dilation_angle_eta` controls the dilation and compaction of the material. When the stress ratio is smaller than this parameter, plastic compaction takes place. When the stress ratio is greater than this parameter, the plastic dilation takes place. [dimensionless]
- `dilation_scale` controls the rate of the dilation or compaction in the plastic flow. For this automatic G/G_{max} match, the dilation scale has to be zero, which means only deviatoric plastic flow is allowed. If the users want to have volumetric dilation, they can match the G/G_{max} manually with the other `DruckerPragerMultipleYieldSurface` command. [dimensionless]
- `total_number_of_shear_modulus` is the total number of shear modulus, including the initial maximum shear modulus. The total number of yield surface is one less than the total number of shear modulus. Namely, $(N+1)$ areas are divided by N surfaces. [dimensionless]
- `GoverGmax` is the G/G_{max} from experiments, including the initial shear modulus. Namely, the first element should be 1.0. Each element is dimensionless. The input should be separated by a blank space or a comma. [string]
- `ShearStrainGamma` is the shear strain γ corresponding to the `GoverGmax`. Note that $\gamma = 2\varepsilon$ when the input is prepared. The first element should be 0.0 corresponding to the initial shear modulus. Each element is dimensionless. The input should be separated by a blank space or a comma. [string]

Modeling, Material Model: Rounder Mohr-Coulomb Nonassociated Multi-Yield Surface Material Model

The command is:

```

1  add material # <.> type RoundedMohrCoulombMultipleYieldSurface
2  mass_density = <M/L^3>
3  elastic_modulus = <F/L^2>
4  poisson_ratio = <.>
5  initial_confining_stress = <F/L^2>
6  reference_pressure = <F/L^2>
7  pressure_exponential_n = <.>
8  cohesion = <F/L^2>
9  RMC_shape_k = <.>
10 dilation_angle_eta = <.>
11 dilation_scale = <.>
12 total_number_of_yield_surface = <.>
13 sizes_of_yield_surfaces = <string>
14 yield_surface_scale_unit = <F/L^2>
15 hardening_parameters_of_yield_surfaces = <string>
16 hardening_parameters_scale_unit = <F/L^2>;

```

where

- `mass_density` is the mass density of material [M/L^3]
- `elastic_modulus` is the elastic modulus of the material [F/L^2]
- `poisson_ratio` is the constant Poisson-ratio [dimensionless]
- `initial_confining_stress` is the initial confining (mean) pressure [F/L^2]
- `reference_pressure` is the reference pressure for the initial modulus. This parameter is usually 101kPa. [F/L^2]
- `pressure_exponential_n` is the exponential number of the pressure dependent modulus. [dimensionless]
- `cohesion` is the attraction force is the soil. [F/L^2]
- `RMC_shape_k` controls the shape of the rounded Mohr-Coulomb yield surface. [dimensionless]
- `dilation_angle_eta` controls the dilation and compaction of the material. When the stress ratio is smaller than this parameter, plastic compaction takes place. When the stress ratio is greater than this parameter, the plastic dilation takes place. [dimensionless]
- `dilation_scale` controls the rate of the dilation or compaction in the plastic flow. [dimensionless]

- `total_number_of_yield_surface` is the total number of yield surfaces. [dimensionless]
- `radiuses_of_yield_surface` is the radius list of multiple yield surfaces. This parameter gives the radiuses of each yield surface from the smallest to the biggest. This parameter should be a string which contains the dimensionless radiuses. The radiuses should be separated by a blank space or a comma. [string]
- `radiuses_scale_unit` is the unit of the each yield surface. This parameter also provides a method to scale up or scale down the radiuses of each yield surfaces. [F/L^2]
- `hardening_parameters_of_yield_surfaces` is the hardening parameters corresponding to each yield surface. This parameter should be a string which contains the dimensionless hardening parameters. The hardening parameters should be separated by a blank space or a comma. [string]
- `hardening_parameters_scale_unit` The unit of the each hardening parameter. This parameter also provides a method to scale up or scale down the hardening parameter of each yield surfaces. [F/L^2]

Modeling, Material Model: Tsinghua Liquefaction Material Model

The command is:

```

1  add material # <.> type TsinghuaLiquefactionModel
2  mass_density = <M/L^3>
3  poisson_ratio = <.>
4  initial_confining_stress = <F/L^2>
5  liquefaction_G0 = <.>
6  liquefaction_EXP_N = <.>
7  liquefaction_c_h0 = <.>
8  liquefaction_mfc = <.>
9  liquefaction_mdc = <.>
10 liquefaction_dre1 = <.>
11 liquefaction_Dre2 = <.>
12 liquefaction_Dir = <.>
13 liquefaction_Alpha = <.>
14 liquefaction_gamar = <.>
15 liquefaction_pa = <.>
16 liquefaction_pmin = <.>

```

Command Example is

```

1  add material # 1 type TsinghuaLiquefactionModel
2  mass_density = 0.0*kg/m^3
3  poisson_ratio = 0.1
4  initial_confining_stress = 1E5 *Pa
5  liquefaction_G0 = 800
6  liquefaction_EXP_N = 0.5
7  liquefaction_c_h0 = 1.0
8  liquefaction_mfc = 1.2
9  liquefaction_mdc = 0.4
10 liquefaction_dre1 = 0.5
11 liquefaction_Dre2 = 1500
12 liquefaction_Dir = 0.1
13 liquefaction_Alpha = 0.01
14 liquefaction_gamar = 0.01
15 liquefaction_pa = 1E5
16 liquefaction_pmin = 100 ;

```

where

- mass_density is the mass density of material [M/L^3]
- poisson_ratio is the constant Poisson ratio [dimensionless]
- initial_confining_stress is the initial confining (mean) pressure [F/L^2]
- liquefaction_G0 is initial modulus scale at the reference pressure. For medium dense soil, G0 is 800. [dimensionless]

- `liquefaction_EXPN` is the exponential number of the pressure dependent modulus. [dimensionless]
- `liquefaction_c_h0` is the plastic modulus coefficient. This parameter should be determined by the G/G_{max} curve. When the G/G_{max} curve is hyperbolic, h is 1.2. The range of h is 0.7-1.2 [dimensionless].
- `liquefaction_mfc` is the slope of the failure surface in p - q plane. The range of $M_{f,c}$ is 1.4-1.8 [dimensionless].
- `liquefaction_mdc` is the slope of the phase transition surface in p - q plane. The range of $M_{d,c}$ is 0.3-1.0 [dimensionless].
- `liquefaction_dre1` is the accumulation coefficient of the reversible dilatancy. This parameter is usually 0.4 [dimensionless].
- `liquefaction_Dre2` is the release coefficient of the reversible dilatancy. This range of $d_{re,2}$ is 1000-1500 [dimensionless].
- `liquefaction_Dir` is the coefficient of irreversible dilatancy. The parameter d_{ir} controls the initial slope of the irreversible strain development with respect to the number of reversible loadings. Intuitively, when d_{ir} is bigger, the soil becomes liquefaction faster. The parameter d_{ir} can be around 0.2 [dimensionless].
- `liquefaction_Alpha` is the limit of the irreversible strain. Intuitively, α controls the maximum strain after the liquefaction. The parameter α can be around 0.03 [dimensionless].
- `liquefaction_gamar` is the maximum shear strain length in one liquefaction loading. Intuitively, this parameter controls the maximum strain size of one loop. This parameter can be around 0.05 [dimensionless].
- `liquefaction_pa` is the reference pressure. Usually, this parameter is 10000 [dimensionless].
- `liquefaction_pmin` is the minimum pressure in the calculation. If the pressure is smaller than p_{min} during the calculation, the pressure will be set to p_{min} . This parameter can be 1. Increasing this parameter can avoid the potential numerical errors on small numbers [dimensionless].

Modeling, Material Model: SANISand Material Model, version 2004

The command is:

```

1 add material # <.> type sanisand2004
2   mass_density = <M/L^3>
3   e0 = <.>
4   sanisand2004_G0 = <.>
5   poisson_ratio = <.>
6   sanisand2004_Pat = <.>
7   sanisand2004_p_cut = <.>
8   sanisand2004_Mc = <.>
9   sanisand2004_c = <.>
10  sanisand2004_lambda_c = <.>
11  sanisand2004_xi = <.>
12  sanisand2004_ec_ref = <.>
13  sanisand2004_m = <.>
14  sanisand2004_h0 = <.>
15  sanisand2004_ch = <.>
16  sanisand2004_nb = <.>
17  sanisand2004_A0 = <.>
18  sanisand2004_nd = <.>
19  sanisand2004_z_max = <.>
20  sanisand2004_cz = <.>
21  initial_confining_stress = <F/L^2>;

```

where

- MaterialNumber: Material tag
- mass_density is the mass density of material [M/L^3]
- sanisand2004_e0 initial void ratio []
- sanisand2004_G0 normalized elastic shear modulus []
- poisson_ratio Poisson's ratio []
- sanisand2004_Pat atmospheric pressure [F/L^2]
- sanisand2004_p_cut pressure cut-off ratio [F/L^2]
- sanisand2004_Mc Critical stress ratio at triaxial compression []
- sanisand2004_c tension-compression strength ratio $c = M_e/M_c$ []
- sanisand2004_lambda_c parameter for critical state line []
- sanisand2004_xi parameter for critical state line []

- `sanisand2004_ec_ref` reference void for critical state line []
- `sanisand2004_m` opening of the yield surface []
- `sanisand2004_h0` bounding surface parameter []
- `sanisand2004_ch` bounding surface parameter []
- `sanisand2004_nb` bounding surface parameter []
- `sanisand2004_A0` dilatancy parameter []
- `sanisand2004_nd` dilatancy parameter []
- `sanisand2004_z_max` maximum z fabric parameter []
- `sanisand2004_cz` fabric hardening parameter []
- `initial_confining_stress` is the initial confining stress $p = -1/3\sigma_{ii}$ and it is positive in compressions (since there is that $-$ (minus) sign in front of sum of normal stresses (σ_{ii} indicial notation summation convention applies) that are positive in tension [stress]).

More on this material model can be found in section [104.6.11](#) on Page [255](#) in Lecture Notes by [Jeremić et al. \(1989-2025\)](#) ([Lecture Notes URL](#)).

Important note: This material model should be used together with explicit constitutive algorithms, e.g. `Forward_Euler` or `Forward_Euler_Subincrement`. For better result, it is suggested to apply strain increments, or sub-increments, smaller than $1e-4$.

Modeling, Material Model: SANISand Material Model, version 2008

The command is:

```

1 add material # <.> type sanisand2008
2   mass_density = <M/L^3>
3   e0 = <.>
4   sanisand2008_G0 = <.>
5   sanisand2008_K0 = <.>
6   sanisand2008_Pat = <.>
7   sanisand2008_k_c = <.>
8   sanisand2008_alpha_cc = <.>
9   sanisand2008_c = <.>
10  sanisand2008_lambda = <.>
11  sanisand2008_ec_ref = <.>
12  sanisand2008_m = <.>
13  sanisand2008_h0 = <.>
14  sanisand2008_ch = <.>
15  sanisand2008_nb = <.>
16  sanisand2008_A0 = <.>
17  sanisand2008_nd = <.>
18  sanisand2008_p_r = <.>
19  sanisand2008_rho_c = <.>
20  sanisand2008_theta_c = <.>
21  sanisand2008_X = <.>
22  sanisand2008_z_max = <.>
23  sanisand2008_cz = <.>
24  sanisand2008_p0 = <F/L^3>
25  sanisand2008_p_in = <F/L^3>
26  algorithm = explicit (or) implicit
27  number_of_subincrements = <.>
28  maximum_number_of_iterations = <.>
29  tolerance_1 = <.>
30  tolerance_2 = <.>;

```

where

- MaterialNumber: Number of the ND material to be used ;
- Algorithm: Explicit (=0) or Implicit (=1) ;
- rho: density ;
- e0: initial void ratio at zero strain ;
- G0: Reference elastic shear modulus [stress];
- K0: Reference elastic bulk modulus [stress];
- sanisand2008_Pat: atmospheric pressure for critical state line ;

- `sanisand2008_k_c`: cut-off factor; for $p < k_c P_{at}$, use $p = k_c P_{at}$ for calculation of G ; (a default value of $k_c = 0.01$ should work fine) ;
- `sanisand2008_alpha_cc`: critical state stress ratio ;
- `sanisand2008_c`: tension-compression strength ratio ;
- `sanisand2008_lambda`: parameter for critical state line ;
- `sanisand2008_xi`: parameter for critical state line ;
- `sanisand2008_ec_ref`: reference void for critical state line, ; $e_c = e_r \lambda (p_c / P_{at})^{\xi}$;
- `sanisand2008_m`: opening of the yield surface ;
- `sanisand2008_h0`: bounding surface parameter ;
- `sanisand2008_ch`: bounding surface parameter ;
- `sanisand2008_nb`: bounding surface parameter ;
- `sanisand2008_A0`: dilatancy parameter ;
- `sanisand2008_nd`: dilatancy parameter ;
- `sanisand2008_p_r`: LCC parameter ;
- `sanisand2008_rho_c`: LCC parameter ;
- `sanisand2008_theta_c`: LCC parameter ;
- `sanisand2008_X`: LCC parameter ;
- `sanisand2008_z_max`: fabric parameter ;
- `sanisand2008_cz`: fabric parameter ;
- `sanisand2008_p0`: yield surface size ;
- `sanisand2008_p_in` ;
- `number_of_subincrements` number of subincrements in constitutive simulation
- `maximum_number_of_iterations` maximum number of iterations
- `tolerance_1` Explicit: tolerance for intersection point (distance between two consecutive points) Implicit: yield function tolerance

- `tolerance_2` Implicit: residual tolerance

More on this material model can be found in Section [104.6.12](#) on Page [262](#) in Lecture Notes by [Jeremić et al. \(1989-2025\)](#) ([Lecture Notes URL](#)).

Modeling, Material Model: Cosserat Linear Elastic Material Model

The command is:

```

1 add material # <.> type Cosserat_linear_elastic_isotropic_3d
2   mass_density = <M/L^3>
3   lambda = <F/L^2>
4   mu = <F/L^2>
5   chi = <F/L^2>
6   pi1 = <F>
7   pi2 = <F>
8   pi3 = <F>
9   ;

```

- MaterialNumber unique material Number.
- mass_density the density of the material.
- lambda, mu, chi, pi1, pi2, pi3 are the 6 Cosserat elastic constants ([Eringen, 2012](#)).

The relations between elastic constants is as follows [Eringen \(2012\)](#). Note the Young's modulus and the Poisson's ratio are different from the classical elasticity:

- Young's modulus $E = (2\mu + \chi)(3\lambda + 2\mu + \chi)$.
- Shear modulus $G = \mu + 1/2\chi$.
- Poisson's ratio $\nu = \lambda/(2\lambda + 2\mu + \chi)$.
- Characteristic length for torsion $l_t = ((\pi_2 + \pi_3)/(2\mu + \chi))^{1/2}$.
- Characteristic length for bending $l_b = (\pi_3/2(2\mu + \chi))^{1/2}$.
- Coupling number $N = (\chi/2(\mu + \chi))$
- Polar ratio $\Phi = (\pi_2 + \pi_3)/(\pi_1 + \pi_2 + \pi_3)$

According to Eringen [Eringen \(2012\)](#), the 6 elastic constants should satisfy the following conditions

$$\begin{aligned}
 3\lambda + 2\mu + \chi &\geq 0, & 2\mu + \chi &\geq 0, & \chi &\geq 0, \\
 3\pi_1 + \pi_2 + \pi_3 &\geq 0, & \pi_3 + \pi_2 &\geq 0, & \pi_3 - \pi_2 &\geq 0.
 \end{aligned} \tag{1.1}$$

Modeling, Material Model: von Mises Cosserat Material Model

The command is:

```
1 add material # <.> type Cosserat_von_Mises
2   mass_density = <M/L^3>
3   lambda = <F/L^2>
4   mu = <F/L^2>
5   chi = <F/L^2>
6   pi1 = <F>
7   pi2 = <F>
8   pi3 = <F>
9   plastic_internal_length = <L>
10  von_mises_radius = <F/L^2>
11  isotropic_hardening_rate = <F/L^2>
12  ;
```

- MaterialNumber unique material Number.
- mass_density the density of the material.
- lambda, mu, chi, pi1, pi2, pi3 are the 6 Cosserat elastic constants [Eringen \(2012\)](#).
- plastic_internal_length is the characteristic length in the plasticity.
- von_mises_radius is radius of the unified yield surface of force-stress and couple-stress.
- isotropic_hardening_rate is the rate of isotropic hardening.

Modeling, Material Model: Uniaxial Linear Elastic, Fiber Material Model

The command is:

```
1 add material # <.> type uniaxial_elastic
2   elastic_modulus = <F/L^2>
3   viscoelastic_modulus = <mass / length / time> ;
```

where

- MaterialNumber unique material Number.
- elastic_modulus elastic modulus of the material.
- viscoelastic_modulus damping tangent.

More on this material model can be found in Section ?? on Page ?? in Lecture Notes by [Jeremić et al. \(1989-2025\)](#) ([Lecture Notes URL](#)).

As the name implies, uniaxiale_elastic material model works with uniaxial element only. For 3D elements, for example solid brick elements, please use 3D material models, for example, linear_elastic_isotropic_3d.

Modeling, Material Model: Stochastic Uniaxial Linear Elastic Model

The command is:

```
1 add material # <.> type stochastic_uniaxial_elastic ↔  
    uncertain_elastic_modulus = random variable # <.> ↔  
    elastic_modulus_scale_unit = <F/L^2>;
```

where

- `uncertain_elastic_modulus` specify uncertain elastic modulus of the material through a defined random variable.
- `elastic_modulus_scale_unit` specify the unit scale factor that would be multiplied with the polynomial chaos coefficients of the random variable.

As the name implies, `stochastic_uniaxial_elastic` material model works with stochastic uniaxial element only.

For example:

```
1 add material # 1 type stochastic_uniaxial_elastic ↔  
    uncertain_elastic_modulus = random variable # 1 ↔  
    elastic_modulus_scale_unit = 1*Pa;
```

Add material #1 as `stochastic_uniaxial_elastic` material with uncertain elastic modulus characterized by the polynomial chaos coefficients of random variable 1 and scale factor $1 * Pa$.

Modeling, Material Model: Stochastic Uniaxial Nonlinear Armstrong Frederick Model

The command is:

```
1 add material # <.> type stochastic_uniaxial_Armstrong_Frederick
2 constitutive triple product # <.>
3 armstrong_frederick_ha = random variable # <.>
4 armstrong_frederick_ha_scale_unit = <F/L^2>
5 armstrong_frederick_cr = random variable # <.>
```

or

```
1 add material # <.> type stochastic_uniaxial_Armstrong_Frederick
2 constitutive triple product # <.>
3 armstrong_frederick_ha = random variable # <.>
4 armstrong_frederick_ha_scale_unit = <F/L^2>
5 armstrong_frederick_cr = random variable # <.>
6 polynomial_chaos_terms_ha = <.>
7 polynomial_chaos_terms_cr = <.>
8 polynomial_chaos_terms_incremental_strain = <.>;
```

Note that the difference between these two commands is that the first command would by default use the full polynomial chaos (PC) bases defined in the provided constitutive triple product for probabilistic constitutive modeling. The second command would support user-specified number of polynomial chaos terms for uncertain armstrong_frederick_ha, armstrong_frederick_cr and incremental_strain. This enables users to perform truncation of PC bases for probabilistic constitutive modeling.

The command input parameters are:

- `constitutive triple product #` specifies the ID of the triple product, that would be used in probabilistic constitutive updating. In stochastic finite element method (FEM), the first and second PC basis for this triple product should come from the joint PC representation of uncertain parameters `armstrong_frederick_ha` and `armstrong_frederick_cr`. The third PC basis for this triple product should come from the PC representation of uncertain FEM system response, e.g., uncertain structural displacement.
- `armstrong_frederick_ha = random variable #` specifies the uncertain Armstrong Frederick parameter *ha* through a defined random variable.
- `armstrong_frederick_ha_scale_unit` specifies the unit scale factor that would be multiplied with the polynomial chaos coefficients of the random variable of uncertain Armstrong Frederick parameter *ha*.
- `armstrong_frederick_cr = random variable #` specifies the uncertain Armstrong Frederick parameter *cr* through a defined random variable.

- `polynomial_chaos_terms_ha` specifies the number of polynomial chaos basis of uncertain ha involved in the probabilistic constitutive updating.
- `polynomial_chaos_terms_cr` specifies the number of polynomial chaos basis of uncertain cr involved in the probabilistic constitutive updating.
- `polynomial_chaos_terms_incremental_strain` specifies the number of polynomial chaos basis of uncertain incremental strain $d\epsilon$ involved in the probabilistic constitutive updating.

As the name implies, `stochastic_uniaxial_Armstrong_Frederick` material model works with stochastic uniaxial element only.

For example:

```
1 add material # 1 type stochastic_uniaxial_Armstrong_Frederick
2 constitutive triple product # 1
3 armstrong_frederick_ha = random variable # 1
4 armstrong_frederick_ha_scale_unit = 1*Pa
5 armstrong_frederick_cr = random variable # 2
6 polynomial_chaos_terms_ha = 10
7 polynomial_chaos_terms_cr = 10
8 polynomial_chaos_terms_incremental_strain = 30;
```

Add material # 1 as `stochastic_uniaxial_Armstrong_Frederick` material with triple product # 1 for probabilistic constitutive updating.

Uncertain parameter ha is characterized by random variable # 1 using scale unit 1*Pa.

Uncertain parameter cr is characterized by random variable # 2. The number of polynomial chaos basis for uncertain parameters ha , cr and incremental strain in probabilistic constitutive updating are 10, 10 and 30, respectively.

Modeling, Material Model: Uniaxial Nonlinear Concrete, Fiber Material Model, version 02

The command is:

```
1 add material # <.> type uniaxial_concrete02
2   compressive_strength = <F/L^2>
3   strain_at_compressive_strength = <.>
4   crushing_strength = <F/L^2>
5   strain_at_crushing_strength = <.>
6   lambda = <.>
7   tensile_strength = <F/L^2>
8   tension_softening_stiffness = <F/L^2>;
```

- compressive_strength compressive strength.
- strain_at_compressive_strength strain at compressive strength.
- crushing_strength crushing strength.
- strain_at_crushing_strength strain at crushing strength.
- lambda ratio between unloading slope at epscu and initial slope.
- tensile_strength tensile strength.
- tension_softening_stiffness tension softening stiffness (absolute value) (slope of the tension softening branch).

More on this material model can be found in Section ?? on Page ?? in Lecture Notes by [Jeremić et al. \(1989-2025\)](#) ([Lecture Notes URL](#)).

Modeling, Material Model: Faria-Oliver-Cervera Concrete Material

The command is:

```
1 add material No (or #) <material_number>
2   type FariaOliverCerveraConcrete
3   elastic_modulus      = <F/L^2>
4   poisson_ratio        = <.>
5   tensile_yield_strength = <F/L^2>
6   compressive_yield_strength = <F/L^2>
7   plastic_deformation_rate = <.>
8   damage_parameter_Ap   = <.>
9   damage_parameter_An   = <.>
10  damage_parameter_Bn   = <.>
```

where

- No (or #) <material_number> is a unique material integer number (does not have to be sequential, any unique positive integer number can be used).
- type FariaOliverCerveraConcrete is the material type.
- elastic_modulus is the elastic modulus of material $[F/L^2]$
- poisson_ratio is the Poisson's ratio material.
- tensile_yield_strength is the tensile yield strength $[F/L^2]$
- compressive_yield_strength is the compressive yield strength $[F/L^2]$

Modeling, Material Model: Plane Stress Layered Material

The command is:

```
1 add material No (or #) <element_number>
2   type PlaneStressLayeredMaterial
3   number_of_layers = <.>
4   thickness_array = <string>
5   thickness_scale_unit = <L>
6   with material # <string>
7   ;
```

where

- No (or #) <material_number> is a unique material integer number (does not have to be sequential, any unique positive integer number can be used).
- type PlaneStressLayeredMaterial is the material type.
- number_of_layers is the number of layers in this layered material. For reinforced concrete wall element, this will be just 3 layers, inside/confined concrete, reinforcement, and outside/unconfined concrete.
- thickness_array is the thickness ratio of each individual material.
- thickness_scale_unit set the length unit and the scale factor for the thickness of the layered material.
- material # <string> is the string of predefined individual material tags.

Modeling, Material Model: Uniaxial Nonlinear Steel, Fiber Material Model, version 01

The command is:

```
1 add material # <.> type uniaxial_steel01
2   yield_strength = <F/L^2>
3   elastic_modulus = <F/L^2>
4   strain_hardening_ratio = <.>
5   a1 = <.>
6   a2 = <.>
7   a3 = <>
8   a4 = <.> ;
```

- yield_strength yield strength.
- elastic_modulus initial elastic tangent.
- strain_hardening_ratio strain-hardening ratio (ratio between post-yield tangent and initial elastic tangent).
- a1, a2, a3, a4 isotropic hardening parameters
 - a1: isotropic hardening parameter, increase of compression yield envelope as proportion of yield strength after a plastic strain of $a2 \cdot (f_y/E_p)$. ;
 - a2: isotropic hardening parameter (see explanation under a1) ;
 - a3: isotropic hardening parameter, increase of tension yield envelope as proportion of yield strength after a plastic strain of $a4 \cdot (f_y/E_p)$;
 - a4: isotropic hardening parameter (see explanation under a3) ;

More on this material model can be found in Section ?? on Page ?? in Lecture Notes by [Jeremić et al. \(1989-2025\)](#) ([Lecture Notes URL](#)).

Modeling, Material Model: Uniaxial Nonlinear Steel, Fiber Material Model, version 02

The command is:

```

1 add material # <.> type uniaxial_steel02
2   yield_strength = <F/L^2>
3   elastic_modulus = <F/L^2>
4   strain_hardening_ratio = <.>
5   R0 = <.>
6   cR1 = <.>
7   cR2 = <.>
8   a1 = <.>
9   a2 = <.>
10  a3 = <>
11  a4 = <.> ;

```

- yield_strength: yield strength ;
- elastic_modulus: initial elastic tangent ;
- strain_hardening_ratio: strain-hardening ratio (ratio between post-yield tangent and initial elastic tangent) ;
- R0, cR1, cR2: control the transition from elastic to plastic branches. Recommended values: R0=between 10 and 20, cR1=0.925, cR2=0.15 ;
- a1, a2, a3, a4: isotropic hardening parameters ;
 - a1: isotropic hardening parameter, increase of compression yield envelope as proportion of yield strength after a plastic strain of $a2 \cdot (F_y/E)$. ;
 - a2: isotropic hardening parameter (see explanation under a1) ;
 - a3: isotropic hardening parameter, increase of tension yield envelope as proportion of yield strength after a plastic strain of $a4 \cdot (F_y/E)$;
 - a4: isotropic hardening parameter (see explanation under a3) ;

More on this material model can be found in Section ?? on Page ?? in Lecture Notes by [Jeremić et al. \(1989-2025\)](#) ([Lecture Notes URL](#)).

Modeling, Material Model: Plane Stress Plastic Damage Concrete Material

This is a plane stress version of the plastic damage concrete model developed by [Faria et al. \(1998\)](#). This material was implemented as part of the endeavor to model reinforced concrete shells, plates and shear walls. It should only be used together with Inelastic Layered Shell Section and 4 Node Shell NLDKGQ/Xin-Zheng-Lu, see page [100](#).

The command is:

```

1 add material No (or #) <material_number> type ←
   PlasticDamageConcretePlaneStress
2   elastic_modulus           = <F/L^2>
3   poisson_ratio             = <.>
4   tensile_yield_strength     = <F/L^2>
5   compressive_yield_strength = <F/L^2>
6   plastic_deformation_rate   = <.>
7   damage_parameter_Ap        = <.>
8   damage_parameter_An        = <.>
9   damage_parameter_Bn        = <.>

```

where

- No (or #) <material_number> is a unique material integer number (does not have to be sequential, any unique positive integer number can be used).
- elastic_modulus is the elastic modulus of material $[F/L^2]$
- poisson_ratio is the Poisson's ratio material.
- tensile_yield_strength is the tensile yield strength $[F/L^2]$
- compressive_yield_strength is the compressive yield strength $[F/L^2]$
- plastic_deformation_rate governs the post-yield hardening modulus in the effective (undamaged) space and the plastic strain rate
- damage_parameter_Ap governs the tensile fracture energy and affects the ductility of the tensile response
- damage_parameter_An governs the softening behavior of concrete in compression, it changes the ductility but does not alter the peak strength
- damage_parameter_Bn governs the softening behavior of concrete in compression, it changes both the ductility and the peak strength

Modeling, Material Model: Plane Stress Rebar Material

This is a plane stress version of the Uniaxial Nonlinear Steel material. This material was implemented as part of the endeavor to model reinforced concrete shells, plates and shear walls. This model should be used together with Inelastic Layered Shell Section and 4 Node Shell NLDKGQ/Xin-Zheng-Lu, see page [100](#).

The command is:

```
1 add material No (or #) <material_number> type PlaneStressRebarMaterial
2   with uniaxial_material # <.>
3   angle = <degree> ;
```

where

- No (or #) <material_number> is a unique material integer number (does not have to be sequential, any unique positive integer number can be used).
- with uniaxial_material # is the material tag of predefined uniaxial steel material
- angle is the angle of uniaxial steel rebars. The angle is 0 along the direction formed by the first two nodes of a 4 Node Shell element.

Modeling, Nodes: Adding Nodes

Nodes can be added to the finite element model.

The command is:

```
1 add node # <.> at (<L>,<L>,<L>) with <.> dofs;
```

For example:

```
1 add node No 1 at (1.0*m, 2.5*m, 3.33*m) with 3 dofs;
```

adds a node number 1 at coordinates $x = 1.0m$, $y = 2.5m$ and $z = 3.33m$ with 3 dofs. The nodes can be of 3dofs $[u_x, u_y, u_z]$, 4dofs $[u_x, u_y, u_z, p]$ (u-p elements), 6dofs $[u_x, u_y, u_z, r_x, r_y, r_z]$ (beams and shells) and 7 dofs $[u_x, u_y, u_z, p, U_x, U_y, U_z]$ (upU element) types. Description of output for nodes of different dof types can be found in section [206.6](#)

Modeling, Nodes: Adding Stochastic Nodes

Nodes can be added to the stochastic finite element model. Different from deterministic finite element analysis, nodes in stochastic FEM should specify the number of polynomial chaos (PC) terms for each physical nodal degree of freedom (dof).

The command is:

```
1 add node # <.> at (<L>,<L>,<L>) with <.> dofs polynomial_chaos_terms ↵
    = <.>;
```

Where:

- `polynomial_chaos_terms` specifies the number of polynomial chaos terms for each physical nodal dof.

The stochastic nodes can also be added as:

```
1 add node # <.> at (<L>,<L>,<L>) with <.> dofs polynomial_chaos_terms ↵
    as random field # <.>;
```

Which specifies the number of polynomial chaos terms for each physical nodal dof using the number of Hermite PC basis of a defined random field.

For example:

```
1 add node # 1 at (1.0*m, 0.0*m, 0.0*m) with 3 dofs ↵
    polynomial_chaos_terms = 10;
```

Add a node # 1 at coordinates $x = 1.0m$, $y = 0.0m$ and $z = 0.0m$ with 3 physical dofs. For each physical dof, the number of terms for polynomial chaos expansion is 10.

```
1 add node # 1 at (1.0*m, 0.0*m, 0.0*m) with 3 dofs ↵
    polynomial_chaos_terms as random field # 2;
```

Add a node # 1 at coordinates $x = 1.0m$, $y = 0.0m$ and $z = 0.0m$ with 3 physical dofs. For each physical dof, the number of terms for polynomial chaos expansion is equal to the number of PC basis of random field # 2.

Modeling, Nodes: Define Nodal Physical Group

Physical Group for nodes can be defined as well.

The command is:

```
1 define physical_node_group "string";
```

For example:

```
1 define physical_node_group "my_new_node_group";
```

this would create a new `physical_node_group` with name "my_new_node_group".

Description of output for physical groups can be found in section [206.5.5](#)

Modeling, Nodes: Adding Nodes to Nodal Physical Group

Already created nodes can be added to the (any) `physical_node_group`.

The command is:

```
1 add nodes (<.>,<.>,...) to physical_node_group "string";
```

For example:

```
1 add nodes (1,2,3) to physical_node_group "my_new_node_group";
```

this would add node tag (1,2 and 3) to already created `physical_node_group` "my_new_node_group". Please note that the nodes (1,2 and 3) must be added to the model before they are added to the `physical_node_group`.

Description of output for physical groups can be found in section [206.5.5](#)

Modeling, Nodes: Removing Nodal Physical Group

Already defined node physical group `physical_node_group` can be removed.

The command is

```
1 remove physical_node_group "string";
```

For example:

```
1 remove physical_node_group "my_new_node_group";
```

this would delete the `physical_node_group` "my_new_node_group".

Modeling, Nodes: Print Nodal Physical Group

Printing already defined nodal physical group `physical_node_group` is possible too.

The command is:

```
1 print physical_node_group "string";
```

For example:

```
1 print physical_node_group "my_new_node_group";
```

this would print the information about `physical_node_group` "my_new_node_group".

```
1 PHYSICAL_NODE_GROUP my_new_node_group  
2 [1 2 3]
```

Modeling, Nodes: Removing Nodes

Nodes can be removed from the finite element model, for example during excavation, removal of finite elements.

The command is:

```
1 remove node No (or #) <.>;
```

For example:

```
1 remove node # 1;
```

Modeling, Nodes: Adding Nodal Mass, for 3DOFs and/or 6DOFs

Nodal mass can be added to nodes with 3 DOFs and/or 6DOFs. This is in addition to nodal mass that is obtained from finite elements.

The command for 3DOFs nodes (truss, solids, wall) is:

```
1 add mass to node # <.>
2 mx = <M>
3 my = <M>
4 mz = <M>;
```

Similarly, the command for 6DOFs nodes (beams and shells) is:

```
1 add mass to node # <.>
2 mx = <M>
3 my = <M>
4 mz = <M>
5 Imx = <M*L^2>
6 Imy = <M*L^2>
7 Imz = <M*L^2>;
```


Modeling, Finite Element: Adding Finite Elements

The basic structure for adding any finite element is:

```
1 add element No (or #)
2   type <finite_element_type>
3   with nodes (<.>, ..., <.>)|
4   {element dependent parameters};
```

Choices for `finite_element_type` are listed below

Modeling, Finite Element: Define Finite Element Physical Group

Physical group for finite elements can be defined.

The command is:

```
1 define physical_element_group "string";
```

For example:

```
1 define physical_element_group "my_new_element_group";
```

this would create a new `physical_element_group` with name `"my_new_element_group"`.

Description of output for physical groups can be found in [Section 206.5.5](#).

Modeling, Finite Element: Adding Elements to Physical Element Group

Finite elements, that already exist in the finite element domain, can be added to the `physical_element_group`.

The command is:

```
1 add elements (<.>,<.>,...) to physical_node_group "string";
```

For example:

```
1 add elements (1,2,3) to physical_node_group "my_new_node_group";
```

this would add elements with tags/numbers (1,2 and 3) to already created `physical_element_group` "my_new_element_group". Please note that the elements (1,2 and 3) must be added to the model before they are added to the `physical_element_group`.

Description of output for physical groups can be found in [Section 206.5.5](#).

Modeling, Finite Element: Remove Physical Finite Element Group

Finite elements can also be removed from the `physical_element_group`.

The command is:

```
1 remove physical_element_group "string";
```

For example:

```
1 remove physical_element_group "my_new_element_group";
```

this would delete the `physical_element_group` "my_new_element_group".

Modeling, Finite Element: Print Physical Finite Element Group

Details of the `physical_element_group` can be printed.

The commands is:

```
1 print physical_element_group "string";
```

For example:

```
1 print physical_element_group "my_new_element_group";
```

this would print the information about `physical_element_group "my_new_element_group"`.

```
1 PHYSICAL_ELEMENT_GROUP my_new_element_group  
2 [1 2 3]
```

Modeling, Finite Element: Remove Finite Element

Finite elements can be removed, for example if modeling requires excavation, removal of finite elements and nodes.

The command is:

```
1 remove element # <.>;
```

For example,

```
1 remove element # 1;
```

Modeling, Finite Element: Truss Element

The command is:

```
1 add element No (or #) <element_number> type truss
2     with nodes (n1, n2)
3     use material No (or #) <material_number>
4     section_area <section_area> [unit];
5     mass_density <mass_density> [unit];
```

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 truss is the element type
- with nodes (n1, n2) are the 2 nodes (node numbers) defining this element
- use material No (or #) is the material number which makes up the element. Material has to be a uniaxial material, and it can be either elastic or one of the elastic-plastic materials defined for uniaxial behavior.
- section_area is the cross section area [L^2]

Description of output by this element can be found in Section [206.8.1](#). more on this finite element can be found in Section [102.6](#) on Page [126](#) in Lecture Notes by [Jeremić et al. \(1989-2025\)](#) ([Lecture Notes URL](#)).

Modeling, Finite Element: Kelvin-Voigt Element

The command is:

```
1 add element # <.> type Kelvin_Voigt
2   with nodes (<.>, <.>)
3   axial_stiffness = <F/L>
4   axial_viscous_damping = <F/L*T>;
```

where

- No (or #) <element_number> is a unique element integer number, that does not have to be sequential, any unique positive integer number can be used
- type Kelvin_Voigt is the element type
- with nodes (n1, n2) are the 2 nodes (node numbers) defining this element
- axial_stiffness represents the stiffness in the axial direction, $[F/L]$
- axial_viscous_damping represents the viscosity, or viscous damping coefficient, in the axial direction, $[F/L * T]$

Note: Nodes defining this element cannot be at the same location, that is, this is a two node element and direction of this element is calculated from two distinct locations/coordinates of nodes.

Modeling, Finite Element: Inerter Element

The command is:

```
1 add element # <.> type Inerter
2   with nodes (<.>, <.>)
3   inertance = <M>;
```

where

- No (or #) <element_number> is a unique element integer number, that does not have to be sequential, any unique positive integer number can be used
- type Inerter is the element type
- with nodes (n1, n2) are the 2 nodes (node numbers) defining this element
- inertance represents the inertance in the axial direction, $[M]$

Note: Nodes defining this element cannot be at the same location, that is, this is a two node element and direction of this element is calculated from two distinct locations/coordinates of nodes.

Modeling, Finite Element: Shear Beam Element

The command is:

```
1 add element # <.> type ShearBeam
2   with nodes (<.>, <.>)
3   cross_section = <l^2>
4   use material # <.>;
```

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)
- with nodes (n1, n2) are the 2 nodes (node numbers) defining this element. NOTE: element is supposed to be alligned along vertical, Z direction !!
- use material No (or #) is the material (LT-based material) number which makes up the element.
- section_area is the cross section area [L^2]

Description of output by this element can be found in Section [206.8.3](#). more on this finite element can be found in Section [102.9](#) on page [135](#) in Lecture Notes by [Jeremić et al. \(1989-2025\)](#) ([Lecture Notes URL](#)).

Modeling, Finite Element: Stochastic Shear Beam Element

Add stochastic shear beam element for stochastic finite element analysis.

```

1 add element # <.> type stochastic_shear_beam with nodes (<.>, <.>)
2   use material # <.>
3   triple product # <.>
4   cross_section = <L^2>
5   mass_density = <M/L^3>;

```

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).
- with nodes (n1, n2) are the 2 nodes (node numbers) defining this element.
- use material No (or #) is the stochastic uniaxial material number that makes up the element.
- triple product # specifies the ID of the triple product, that would be used in the formation of elemental stochastic stiffness matrix. In stochastic finite element method (FEM), the first PC basis for this triple product should come from the PC representation of uncertain element stiffness. The second and third PC basis for this triple product should come from the PC representation of uncertain FEM system response, e.g., uncertain structural displacement.
- section_area is the cross section area [L^2].
- mass_density is the density [M/L^3].

For example:

```

1 add element # 1 type stochastic_shear_beam with nodes (1, 2) use ↵
   material # 1 triple product # 1 cross_section = 1*m^2 mass_density ↵
   = 2000*kg/m^3;

```

Add a stochastic shear beam element # 1 with stochastic nodes 1 and 2 using stochastic uniaxial material # 1.

The cross section of the element is 1 m^2 and mass density is 2000 kg/m^3 .

Modeling, Finite Element: Elastic Beam–Column Element

The command is:

```

1 add element # <.> type beam_elastic with nodes (<.>, <.>)
2   cross_section = <L^2>
3   elastic_modulus = <F/L^2>
4   shear_modulus = <F/L^2>
5   torsion_Jx = <length^4>
6   bending_Iy = <length^4>
7   bending_Iz = <length^4>
8   mass_density = <M/L^3>
9   xz_plane_vector = (<.>, <.>, <.> )
10  joint_1_offset = (<L>, <L>, <L> )
11  joint_2_offset = (<L>, <L>, <L> );

```

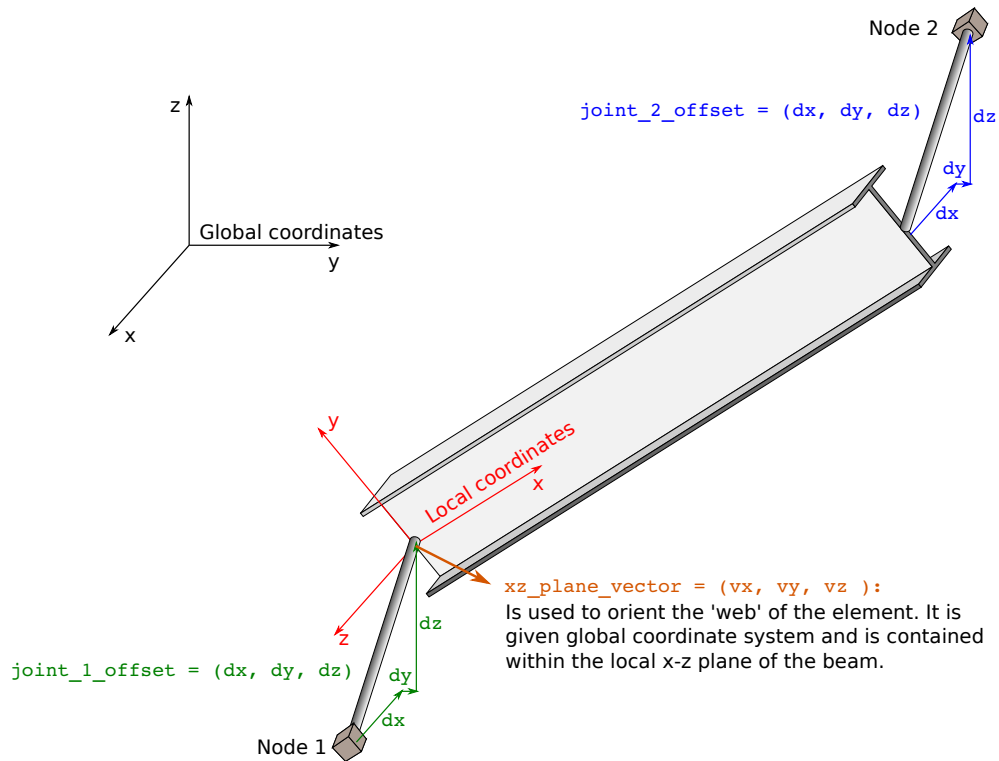


Figure 1.3: Beam Element, sketch of main geometric components.

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. **NOTE:** Please make sure that your `xz_plane_vector` is a bit away from the actual local x axes, the axes that runs along the beam element, in order to prevent numerical problems that might appear when vector cross products are performed inside the program... It is suggested that your `xz_plane_vector` be closer to local z axes... See Figure 1.4 on Page 86 for more in depth explanation of `xz_plane_vector`.
- `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](#)).

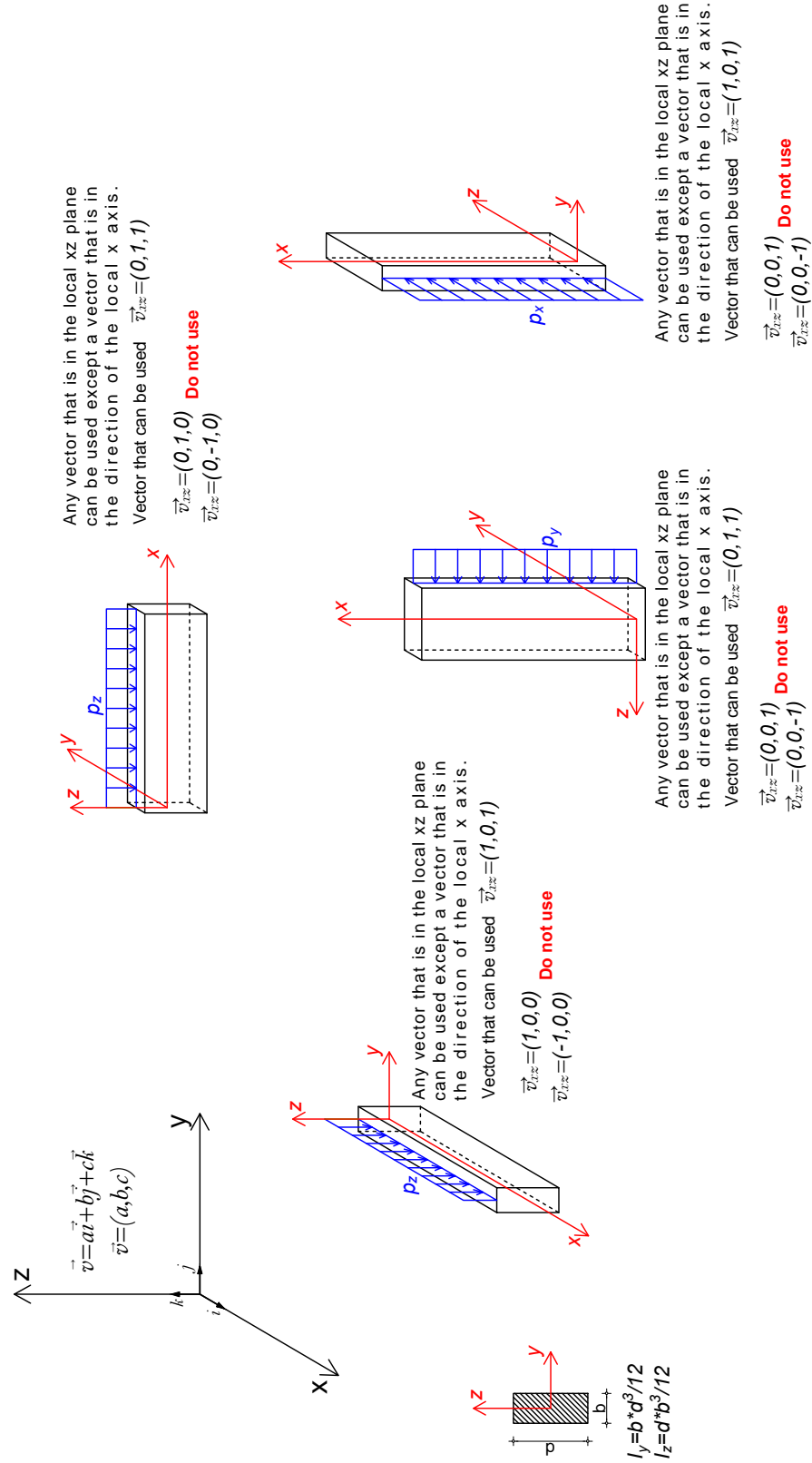


Figure 1.4: Beam Element, details of vector in X-Z plane.

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

The command is:

```

1 add element # <.> type beam_elastic_corotational with nodes (<.>, <.>)
2   cross_section = <L^2>
3   elastic_modulus = <F/L^2>
4   shear_modulus = <F/L^2>
5   torsion_Jx = <length^4>
6   bending_Iy = <length^4>
7   bending_Iz = <length^4>
8   mass_density = <M/L^3>
9   xz_plane_vector = (<.>, <.>, <.> )
10  joint_1_offset = (<L>, <L>, <L> )
11  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, $[L]$

Modeling, Finite Element: Timoshenko Elastic Beam–Column Element

The command is:

```

1 add element # <.> type beam_elastic_Timoshenko with nodes (<.>, <.>)
2   cross_section = <L^2>
3   elastic_modulus = <F/L^2>
4   shear_modulus = <F/L^2>
5   torsion_Jx = <length^4>
6   bending_Iy = <length^4>
7   bending_Iz = <length^4>
8   mass_density = <M/L^3>
9   shear_correction_coefficient = <.>
10  xz_plane_vector = (<.>, <.>, <.>)
11  joint_1_offset = (<L>, <L>, <L>)
12  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](#)).

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

The command is:

```

1  add element # <.> type beam_elastic_Timoshenko_directional with nodes <->
    (<.>, <.>)
2      cross_section = <L^2>
3      elastic_modulus = <F/L^2>
4      shear_modulus = <F/L^2>
5      torsion_Jx = <length^4>
6      bending_Iy = <length^4>
7      bending_Iz = <length^4>
8      mass_density = <M/L^3>
9      shear_correction_coefficient_y = <.>
10     shear_correction_coefficient_z = <.>
11     xz_plane_vector = (<.>, <.>, <.>)
12     joint_1_offset = (<L>, <L>, <L>)
13     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](#)).

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

Fibers can be added to the fiber beam cross section.

The command is:

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

For example:

```
1 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`.

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 1.3.4 on page 92.

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.

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

```

1 add element # <.> type BeamColumnDispFiber3d with nodes (<.>, <.>)
2   number_of_integration_points = <.>
3   section_number = <.>
4   mass_density = <M/L^3>
5   xz_plane_vector = (<.>, <.>, <.>)
6   joint_1_offset = (<L>, <L>, <L>)
7   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](#),

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

```

1 add element # <.> type BeamColumnDispFiber3d_Corotational with nodes ↵
  (<.>, <.>)
2   number_of_integration_points = <.>
3   section_number = <.>
4   mass_density = <M/L^3>
5   xz_plane_vector = (<.>, <.>, <.>)
6   joint_1_offset = (<L>, <L>, <L>)
7   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 \(1990\)](#).

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

```

1 add element # <.> type beam_9dof_elastic
2   with nodes (<.>, <.>)
3   cross_section = <L^2>
4   elastic_modulus = <F/L^2>
5   shear_modulus = <F/L^2>
6   torsion_Jx = <length^4>
7   bending_Iy = <length^4>
8   bending_Iz = <length^4>
9   mass_density = <M/L^3>
10  xz_plane_vector = (<.>, <.>, <.> )
11  joint_1_offset = (<L>, <L>, <L> )
12  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. If this 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.

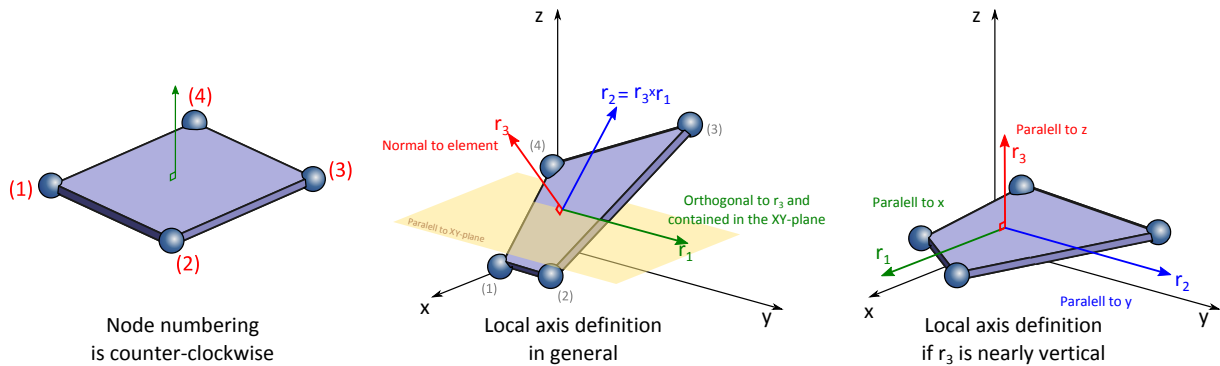
More on this finite element can be found in Section 102.8 on Page 129 in Lecture Notes by Jeremić et al. (1989-2025) ([Lecture Notes URL](#)).

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:

```
1 add element # <.> type 4NodeShell_ANDES
2   with nodes (<.>, <.>, <.>)
3   use material # <.>
4   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)
- 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](#)).

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

```
1 add element # <.> type 3NodeShell_ANDES
2   with nodes (<.>, <.>, <.>)
3   use material # <.>
4   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)

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:

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

It can also be called using the alternative command:

```
1 add element # <.> type 4NodeShell_XinZhengLu_Tsinghua
2   with nodes (<.>, <.>, <.>, <.>)
3   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 [101](#).

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:

```
1 add section # <.> type LayeredShellFiber
2   number_of_layers = <.>
3   thickness_array = "<.>,<.>..."
4   with material # "<.>,<.>..."
5   thickness_scale_unit = <L>
6   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 [63](#) and [64](#).
- `thickness_scale_unit` is the total thickness of the section
- `outofplane_shear_modulus` is the out-of-plane shear modulus of the section

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:

```
1 add element No (or #) <element_number>
2   type ElasticMembranePlaneStress
3   with nodes (n1, n2, n3, n4)
4   use material No (or #) <material_number>
5   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.

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:

```
1 add element No (or #) <element_number>
2   type InelasticMembranePlaneStress
3   with nodes (n1, n2, n3, n4)
4   use material No (or #) <material_number>
5   ;
```

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.

Modeling, Finite Element: SuperElementLinearElasticImport

The command is:

```
1 add element No (or #) <element_number>
2   type SuperElementLinearElasticImport
3   with hdf5_file = <string>
4   ;
```

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 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).

Modeling, Finite Element: 8 Node Brick Element

The command is:

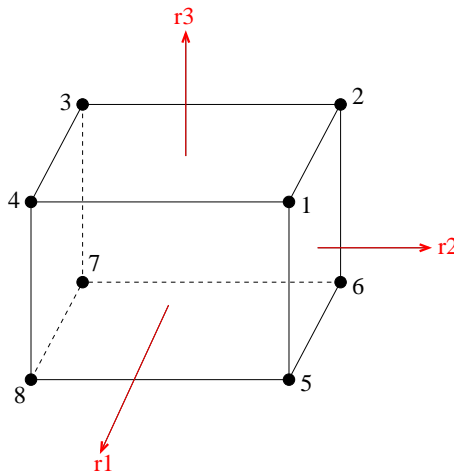
```
1 add element # <element_number> type 8NodeBrick
2   using <.> Gauss points each direction
3   with nodes (n1, n2, n3, n4, n5, n6, n7, n8)
4   use material No (or #) <material_number>;
```

and/or;

```
1 add element # <element_number> type 8NodeBrick
2   with nodes (n1, n2, n3, n4, n5, n6, n7, n8)
3   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](#)).

Modeling, Finite Element: 20 Node Brick Element

The command is:

```

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

```

and/or

```

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

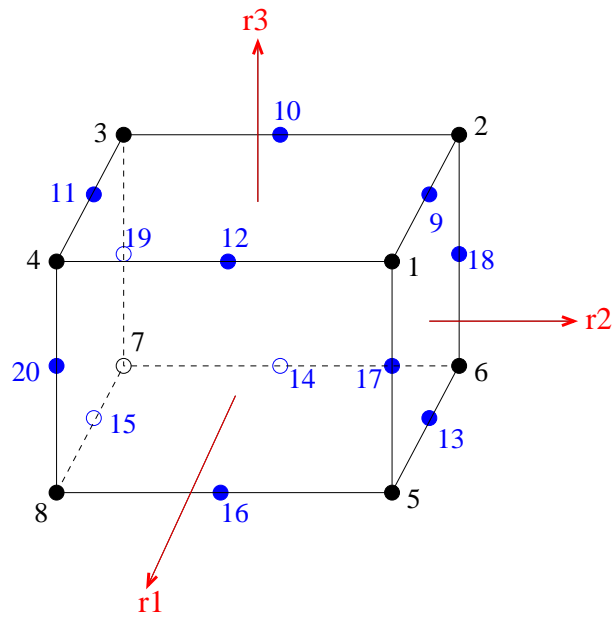
```

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. In 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](#)).



Modeling, Finite Element: 27 Node Brick Element

The command is:

```

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

```

and/or

```

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

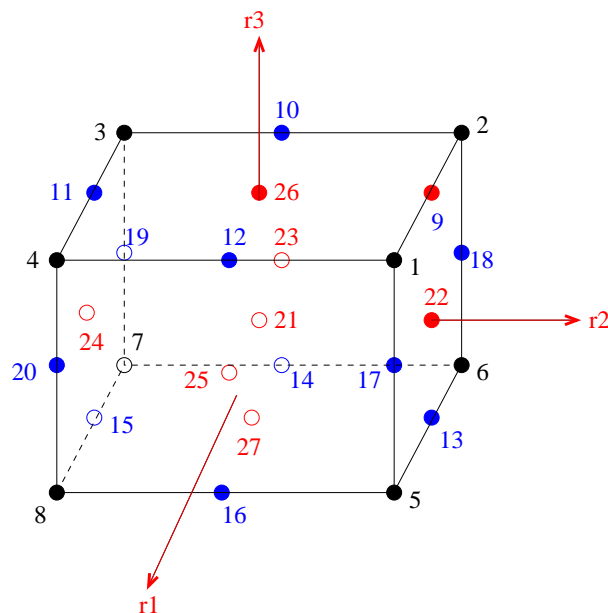
```

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](#)).



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

The command is:

```

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

```

and/or

```

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

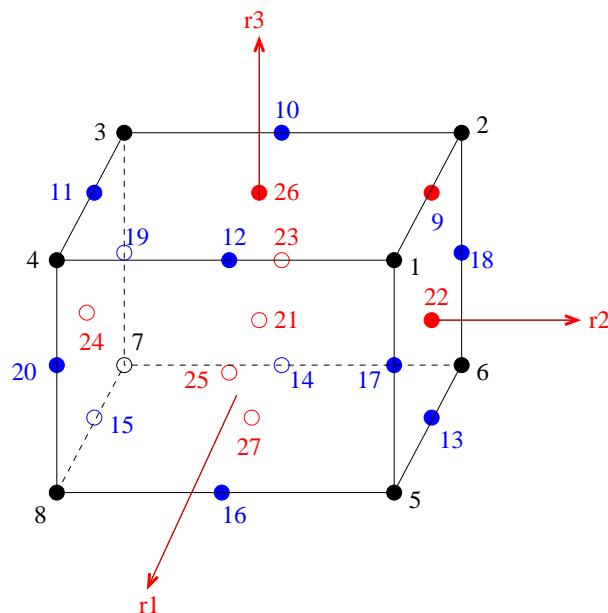
```

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](#).



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

The command is:

```

1 add element # <.> type 8NodeBrick_up
2   using <.> Gauss points each direction
3   with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>)
4   use material # <.>
5   porosity = <.>
6   alpha = <.>
7   rho_s = <M/L^3>
8   rho_f = <M/L^3>
9   k_x = <L^3*T/M>
10  k_y = <L^3*T/M>
11  k_z = <L^3*T/M>
12  K_s = <F/L^2>
13  K_f = <F/L^2>;

```

and/or

```

1 add element # <.> type 8NodeBrick_up
2   with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>)
3   use material # <.>
4   porosity = <.>
5   alpha = <.>
6   rho_s = <M/L^3>
7   rho_f = <M/L^3>
8   k_x = <L^3*T/M>
9   k_y = <L^3*T/M>
10  k_z = <L^3*T/M>
11  K_s = <F/L^2>
12  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](#)).

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

The command is:

```

1 add element # <.> type 20NodeBrick_up
2   using <.> Gauss points each direction
3   with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>)
4   use material # <.>
5   porosity = <.>
6   alpha = <.>
7   rho_s = <M/L^3>
8   rho_f = <M/L^3>
9   k_x = <L^3*T/M>
10  k_y = <L^3*T/M>
11  k_z = <L^3*T/M>
12  K_s = <F/L^2>
13  K_f = <F/L^2>;

```

and/or

```

1 add element # <.> type 20NodeBrick_up
2   with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>)
3   use material # <.>
4   porosity = <.>
5   alpha = <.>
6   rho_s = <M/L^3>
7   rho_f = <M/L^3>
8   k_x = <L^3*T/M>
9   k_y = <L^3*T/M>
10  k_z = <L^3*T/M>
11  K_s = <F/L^2>
12  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](#)).

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

The command is:

```

1 add element # <.> type 27NodeBrick_up
2   using <.> Gauss points each direction
3   with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>,<.>, <.>, <.>, ←
<.>, <.>, <.>, <.>, <.>,<.>, <.>, <.>, <.>, <.>, <.>, <.>, ←
<.>, <.>, <.>)
4   use material # <.>
5   porosity = <.>
6   alpha = <.>
7   rho_s = <M/L^3>
8   rho_f = <M/L^3>
9   k_x = <L^3*T/M>
10  k_y = <L^3*T/M>
11  k_z = <L^3*T/M>
12  K_s = <F/L^2>
13  K_f = <F/L^2>;

```

and/or

```

1 add element # <.> type 27NodeBrick_up
2   with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>,<.>, <.>, <.>, ←
   <.>, <.>, <.>, <.>, <.>,<.>, <.>, <.>, <.>, <.>, <.>, <.>,<.>, ←
   <.>, <.>, <.>)
3   use material # <.>
4   porosity = <.>
5   alpha = <.>
6   rho_s = <M/L^3>
7   rho_f = <M/L^3>
8   k_x = <L^3*T/M>
9   k_y = <L^3*T/M>
10  k_z = <L^3*T/M>
11  K_s = <F/L^2>
12  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 `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](#).

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

The command is:

```

1 add element # <.> type 8NodeBrick_upU
2   using <.> Gauss points each direction
3   with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>)
4   use material # <.>
5   porosity = <.>
6   alpha = <.>
7   rho_s = <M/L^3>
8   rho_f = <M/L^3>
9   k_x = <L^3*T/M>
10  k_y = <L^3*T/M>
11  k_z = <L^3*T/M>
12  K_s = <F/L^2>
13  K_f = <F/L^2>;

```

and/or

```

1 add element # <.> type 8NodeBrick_upU
2   with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>)
3   use material # <.>
4   porosity = <.>
5   alpha = <.>
6   rho_s = <M/L^3>
7   rho_f = <M/L^3>
8   k_x = <L^3*T/M>
9   k_y = <L^3*T/M>
10  k_z = <L^3*T/M>
11  K_s = <F/L^2>
12  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 ($3 \times 3 \times 3$).

- `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 `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](#)).

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

The command is:

```

1 add element # <.> type 20NodeBrick_upU
2   using <.> Gauss points each direction
3   with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>)
4   use material # <.>
5   porosity = <.>
6   alpha = <.>
7   rho_s = <M/L^3>
8   rho_f = <M/L^3>
9   k_x = <L^3*T/M>
10  k_y = <L^3*T/M>
11  k_z = <L^3*T/M>
12  K_s = <F/L^2>
13  K_f = <F/L^2>;

```

and/or

```

1 add element # <.> type 20NodeBrick_upU
2   with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>)
3   use material # <.>
4   porosity = <.>
5   alpha = <.>
6   rho_s = <M/L^3>
7   rho_f = <M/L^3>
8   k_x = <L^3*T/M>
9   k_y = <L^3*T/M>
10  k_z = <L^3*T/M>
11  K_s = <F/L^2>
12  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 `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](#).

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

The command is:

```

1 add element # <.> type 27NodeBrick_upU
2   using <.> Gauss points each direction
3   with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>,<.>, <.>, <.>, ←
<.>, <.>, <.>, <.>, <.>,<.>, <.>, <.>, <.>, <.>, <.>, <.>,<.>, ←
<.>, <.>, <.>)
4   use material # <.>
5   porosity = <.>
6   alpha = <.>
7   rho_s = <M/L^3>
8   rho_f = <M/L^3>
9   k_x = <L^3*T/M>
10  k_y = <L^3*T/M>
11  k_z = <L^3*T/M>
12  K_s = <F/L^2>
13  K_f = <F/L^2>;

```

and/or

```

1 add element # <.> type 27NodeBrick_upU
2   with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, ←
   <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, ←
   <.>, <.>, <.>)
3   use material # <.>
4   porosity = <.>
5   alpha = <.>
6   rho_s = <M/L^3>
7   rho_f = <M/L^3>
8   k_x = <L^3*T/M>
9   k_y = <L^3*T/M>
10  k_z = <L^3*T/M>
11  K_s = <F/L^2>
12  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 `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](#).

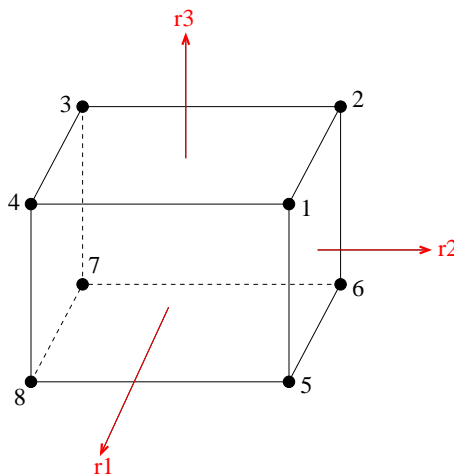
Modeling, Finite Element: 8 Node Cosserat Brick Element

The command is:

```
1 add element # <element_number> type Cosserat8NodeBrick
2   with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>)
3   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.

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

The command is:

```
1 add element # <.> type BondedContact
2   with nodes (<.>, <.>)
3   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.

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

The command is:

```
1 add element # <.> type CoupledBondedContact
2   with nodes (<.>, <.>)
3   penalty_stiffness = <F/L>
4   axial_viscous_damping = <F/L>
5   shear_viscous_damping = <F/L>
6   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.

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

The command is:

```

1 add element # <.> type ForceBasedHardContact
2   with nodes (<.>, <.>)
3   axial_stiffness = <F/L>
4   shear_stiffness = <F/L>
5   axial_viscous_damping = <F/L>
6   shear_viscous_damping = <F/L>
7   friction_ratio = <.>
8   contact_plane_vector = (<.>, <.>, <.> );

```

The axial force F_a and axial stiffness E_a in defined as

$$F_a = E_a * \delta_a \quad (1.2)$$

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](#).

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

The command is:

```

1 add element # <.> type ForceBasedSoftContact
2   with nodes (<.>, <.>)
3   initial_axial_stiffness = <F/L>
4   stiffening_rate = <1/m>
5   max_axial_stiffness = <F/L>
6   shear_stiffness = <F/L>
7   axial_viscous_damping = <F/L>
8   shear_viscous_damping = <F/L>
9   friction_ratio = <.>
10  contact_plane_vector = (<.>, <.>, <.> );

```

The axial force F_a and axial stiffness E_a in defined as

$$\begin{aligned}
 F_a &= b * \exp(a * \delta_a) * \delta_a \\
 E_a &= \max(b * \exp(a * \delta_a) * (1 + a * \delta_a), E_{max})
 \end{aligned}
 \tag{1.3}$$

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](#).

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

The command is:

```

1 add element # <.> type ForceBasedCoupledHardContact
2   with nodes (<.>, <.>)
3   axial_stiffness = <F/L>
4   axial_penalty_stiffness = <F/L>
5   shear_stiffness = <F/L>
6   axial_viscous_damping = <F/L>
7   shear_viscous_damping = <F/L>
8   friction_ratio = <.>
9   contact_plane_vector = (<.>, <.>, <.> );

```

The axial force F_a and axial stiffness E_a in defined as

$$\begin{aligned}
 F_a &= b * \delta_a \\
 E_a &= b
 \end{aligned}
 \tag{1.4}$$

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](#).

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

The command is:

```

1 add element # <.> type ForceBasedCoupledSoftContact
2   with nodes (<.>, <.>)
3   initial_axial_stiffness = <F/L>
4   stiffening_rate = <1/m>
5   max_axial_stiffness = <F/L>
6   axial_penalty_stiffness = <F/L>
7   shear_stiffness = <F/L>
8   axial_viscous_damping = <F/L>
9   shear_viscous_damping = <F/L>
10  friction_ratio = <.>
11  contact_plane_vector = (<.>, <.>, <.> );

```

The axial force F_a and axial stiffness E_a in defined as

$$\begin{aligned}
 F_a &= b * \exp(a * \delta_a) * \delta_a \\
 E_a &= \max(b * \exp(a * \delta_a) * (1 + a * \delta_a), E_{max})
 \end{aligned}
 \tag{1.5}$$

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](#).

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

The command is:

```

1 add element # <.> type StressBasedHardContact_ElPP1Shear
2   with nodes (<.>, <.>)
3   axial_stiffness = <Pa>
4   initial_shear_stiffness = <Pa>
5   axial_viscous_damping = <Pa*s>
6   shear_viscous_damping = <Pa*s>
7   residual_friction_coefficient = <.>
8   shear_zone_thickness = <m>
9   contact_plane_vector = (<.>, <.>, <.> );

```

and/or;

```

1 add element # <.> type StressBasedHardContact_ElPP1Shear
2   with nodes (<.>, <.>)
3   axial_stiffness = <Pa>
4   initial_shear_stiffness = <Pa>
5   axial_viscous_damping = <Pa*s>
6   shear_viscous_damping = <Pa*s>
7   residual_friction_coefficient = <.>
8   shear_zone_thickness = <m>
9   surface_vector_relative_tolerance = <.>;

```

The axial stress σ_a and axial stiffness E_a in defined as

$$\begin{aligned}\sigma_a &= b * \epsilon_a \\ E_a &= b\end{aligned}\tag{1.6}$$

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](#).

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

The command is:

```

1 add element # <.> type StressBasedHardContact_NonLinHardShear
2   with nodes (<.>, <.>)
3   axial_stiffness = <Pa>
4   initial_shear_stiffness = <Pa>
5   axial_viscous_damping = <Pa*s>
6   shear_viscous_damping = <Pa*s>
7   residual_friction_coefficient = <.>
8   shear_zone_thickness = <m>
9   contact_plane_vector = (<.>, <.>, <.> );

```

and/or;

```

1 add element # <.> type StressBasedHardContact_NonLinHardShear
2   with nodes (<.>, <.>)
3   axial_stiffness = <Pa>
4   initial_shear_stiffness = <Pa>
5   axial_viscous_damping = <Pa*s>
6   shear_viscous_damping = <Pa*s>
7   residual_friction_coefficient = <.>
8   shear_zone_thickness = <m>
9   surface_vector_relative_tolerance = <.>;

```

The axial stress σ_a and axial stiffness E_a in defined as

$$\begin{aligned}\sigma_a &= b * \epsilon_a \\ E_a &= b\end{aligned}\tag{1.7}$$

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](#).

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

The command is:

```

1 add element # <.> type StressBasedHardContact_NonLinHardSoftShear
2   with nodes (<.>, <.>)
3   axial_stiffness = <Pa>
4   initial_shear_stiffness = <Pa>
5   rate_of_softening = <>
6   size_of_peak_plateau = <>
7   axial_viscous_damping = <Pa*s>
8   shear_viscous_damping = <Pa*s>
9   peak_friction_coefficient_limit = <>
10  peak_friction_coefficient_rate_of_decrease = <.>
11  residual_friction_coefficient = <.>
12  shear_zone_thickness = <m>
13  contact_plane_vector = (<.>, <.>, <.> );

```

and/or;

```

1 add element # <.> type StressBasedHardContact_NonLinHardSoftShear
2   with nodes (<.>, <.>)
3   axial_stiffness = <Pa>
4   initial_shear_stiffness = <Pa>
5   rate_of_softening = <>
6   size_of_peak_plateau = <>
7   axial_viscous_damping = <Pa*s>
8   shear_viscous_damping = <Pa*s>
9   peak_friction_coefficient_limit = <>
10  peak_friction_coefficient_rate_of_decrease = <.>
11  residual_friction_coefficient = <.>
12  shear_zone_thickness = <m>
13  surface_vector_relative_tolerance = <.>;

```

The axial stress σ_a and axial stiffness E_a in defined as

$$\begin{aligned}\sigma_a &= b * \epsilon_a \\ E_a &= b\end{aligned}\tag{1.8}$$

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

$$\Delta\mu = -\frac{n * R_s(\mu_p - \mu_r)}{(\pi/2)^n \theta^{1/n-1}} * \cos^2\theta \Delta\gamma^p \quad (1.9)$$

$$\theta = \frac{\mu_p - \mu}{\mu_p - \mu_r} (\pi/2)^n \quad (1.10)$$

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.

$$\Delta\gamma^p = \sqrt{\Delta\epsilon_{ij}^p \Delta\epsilon_{ij}^p} \quad (1.11)$$

- `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 1.17.
- `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

$$\mu_p = \max(\mu_{p0}, \mu_{p0} - k * \log(\sigma_a/P_0)) \quad (1.12)$$

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](#).

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

The command is:

```

1 add element # <.> type StressBasedSoftContact_ElPP1Shear
2   with nodes (<.>, <.>)
3   initial_axial_stiffness = <Pa>
4   stiffening_rate = <>
5   max_axial_stiffness = <Pa>
6   initial_shear_stiffness = <Pa>
7   axial_viscous_damping = <Pa*s>
8   shear_viscous_damping = <Pa*s>
9   residual_friction_coefficient = <.>
10  shear_zone_thickness = <m>
11  contact_plane_vector = (<.>, <.>, <.> );

```

and/or;

```

1 add element # <.> type StressBasedSoftContact_ElPP1Shear
2   with nodes (<.>, <.>)
3   initial_axial_stiffness = <Pa>
4   stiffening_rate = <>
5   max_axial_stiffness = <Pa>
6   initial_shear_stiffness = <Pa>
7   axial_viscous_damping = <Pa*s>
8   shear_viscous_damping = <Pa*s>
9   residual_friction_coefficient = <.>
10  shear_zone_thickness = <m>
11  surface_vector_relative_tolerance = <.>;

```

The axial stress σ_a and axial stiffness E_a in defined as

$$\begin{aligned}
 \sigma_a &= b * \exp(a * \epsilon_a) * \epsilon_a \\
 E_a &= \max(b * \exp(a * \epsilon_a) * (1 + a * \epsilon_a), E_{max})
 \end{aligned}
 \tag{1.13}$$

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.

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

The command is:

```

1 add element # <.> type StressBasedSoftContact_NonLinHardShear
2   with nodes (<.>, <.>)
3   initial_axial_stiffness = <Pa>
4   stiffening_rate = <>
5   max_axial_stiffness = <Pa>
6   initial_shear_stiffness = <Pa>
7   axial_viscous_damping = <Pa*s>
8   shear_viscous_damping = <Pa*s>
9   residual_friction_coefficient = <.>
10  shear_zone_thickness = <m>
11  contact_plane_vector = (<.>, <.>, <.> );

```

and/or;

```

1 add element # <.> type StressBasedSoftContact_NonLinHardShear
2   with nodes (<.>, <.>)
3   initial_axial_stiffness = <Pa>
4   stiffening_rate = <>
5   max_axial_stiffness = <Pa>
6   initial_shear_stiffness = <Pa>
7   axial_viscous_damping = <Pa*s>
8   shear_viscous_damping = <Pa*s>
9   residual_friction_coefficient = <.>
10  shear_zone_thickness = <m>
11  surface_vector_relative_tolerance = <.>;

```

The axial stress σ_a and axial stiffness E_a in defined as

$$\begin{aligned}\sigma_a &= b * \exp(a * \epsilon_a) * \epsilon_a \\ E_a &= \max(b * \exp(a * \epsilon_a) * (1 + a * \epsilon_a), E_{max})\end{aligned}\tag{1.14}$$

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](#).

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

The command is:

```

1 add element # <.> type StressBasedSoftContact_NonLinHardSoftShear
2   with nodes (<.>, <.>)
3   initial_axial_stiffness = <Pa>
4   stiffening_rate = <>
5   max_axial_stiffness = <Pa>
6   initial_shear_stiffness = <Pa>
7   rate_of_softening = <>
8   size_of_peak_plateau = <>
9   axial_viscous_damping = <Pa*s>
10  shear_viscous_damping = <Pa*s>
11  peak_friction_coefficient_limit = <>
12  peak_friction_coefficient_rate_of_decrease = <.>
13  residual_friction_coefficient = <.>
14  shear_zone_thickness = <m>
15  contact_plane_vector = (<.>, <.>, <.> );

```

and/or;

```

1 add element # <.> type StressBasedSoftContact_NonLinHardSoftShear
2   with nodes (<.>, <.>)
3   initial_axial_stiffness = <Pa>
4   stiffening_rate = <>
5   max_axial_stiffness = <Pa>
6   initial_shear_stiffness = <Pa>
7   rate_of_softening = <>
8   size_of_peak_plateau = <>
9   axial_viscous_damping = <Pa*s>
10  shear_viscous_damping = <Pa*s>
11  peak_friction_coefficient_limit = <>
12  peak_friction_coefficient_rate_of_decrease = <.>
13  residual_friction_coefficient = <.>
14  shear_zone_thickness = <m>
15  surface_vector_relative_tolerance = <.>;

```

The axial stress σ_a and axial stiffness E_a in defined as

$$\begin{aligned}
 \sigma_a &= b * \exp(a * \epsilon_a) * \epsilon_a \\
 E_a &= \max(b * \exp(a * \epsilon_a) * (1 + a * \epsilon_a), E_{max})
 \end{aligned}
 \tag{1.15}$$

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

$$\Delta\mu = -\frac{n * R_s(\mu_p - \mu_r)}{(\pi/2)^n \theta^{1/n-1}} * \cos^2\theta \Delta\gamma^p \quad (1.16)$$

$$\theta = \frac{\mu_p - \mu}{\mu_p - \mu_r} (\pi/2)^n \quad (1.17)$$

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.

$$\Delta\gamma^p = \sqrt{\Delta\epsilon_{ij}^p \Delta\epsilon_{ij}^p} \quad (1.18)$$

- `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 1.17.
- `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

$$\mu_p = \max(\mu_{p0}, \mu_{p0} - k * \log(\sigma_a/P_0)) \quad (1.19)$$

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](#).

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

The command is:

```

1 add element # <.> type StressBasedCoupledHardContact_ElPP1Shear
2   with nodes (<.>, <.>)
3   axial_stiffness = <Pa>
4   axial_penalty_stiffness = <Pa>
5   initial_shear_stiffness = <Pa>
6   axial_viscous_damping = <Pa*s>
7   shear_viscous_damping = <Pa*s>
8   residual_friction_coefficient = <.>
9   shear_zone_thickness = <m>
10  contact_plane_vector = (<.>, <.>, <.> );

```

and/or;

```

1 add element # <.> type StressBasedCoupledHardContact_ElPP1Shear
2   with nodes (<.>, <.>)
3   axial_stiffness = <Pa>
4   axial_penalty_stiffness = <Pa>
5   initial_shear_stiffness = <Pa>
6   axial_viscous_damping = <Pa*s>
7   shear_viscous_damping = <Pa*s>
8   residual_friction_coefficient = <.>
9   shear_zone_thickness = <m>
10  surface_vector_relative_tolerance = <.>;

```

The axial stress σ_a and axial stiffness E_a in defined as

$$\begin{aligned}\sigma_a &= b * \epsilon_a \\ E_a &= b\end{aligned}\tag{1.20}$$

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](#).

Modeling, Finite Element: Stress Based Coupled Hard Contact/Interface/Joint Element with Non-linear Hardening Shear Behavior

The command is:

```

1 add element # <.> type StressBasedCoupledHardContact_NonLinHardShear
2   with nodes (<.>, <.>)
3   axial_stiffness = <Pa>
4   axial_penalty_stiffness = <Pa>
5   initial_shear_stiffness = <Pa>
6   axial_viscous_damping = <Pa*s>
7   shear_viscous_damping = <Pa*s>
8   residual_friction_coefficient = <.>
9   shear_zone_thickness = <m>
10  contact_plane_vector = (<.>, <.>, <.> );

```

and/or;

```

1 add element # <.> type StressBasedCoupledHardContact_NonLinHardShear
2   with nodes (<.>, <.>)
3   axial_stiffness = <Pa>
4   axial_penalty_stiffness = <Pa>
5   initial_shear_stiffness = <Pa>
6   axial_viscous_damping = <Pa*s>
7   shear_viscous_damping = <Pa*s>
8   residual_friction_coefficient = <.>
9   shear_zone_thickness = <m>
10  surface_vector_relative_tolerance = <.>;

```

The axial stress σ_a and axial stiffness E_a in defined as

$$\begin{aligned}\sigma_a &= b * \epsilon_a \\ E_a &= b\end{aligned}\tag{1.21}$$

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](#).

Modeling, Finite Element: Stress Based Coupled Hard Contact/Interface/Joint Element with Non-linear Hardening and Softening Shear Behavior

The command is:

```

1 add element # <.> type StressBasedCoupledHardContact_NonLinHardSoftShear
2   with nodes (<.>, <.>)
3   axial_stiffness = <Pa>
4   axial_penalty_stiffness = <Pa>
5   initial_shear_stiffness = <Pa>
6   rate_of_softening = <>
7   size_of_peak_plateau = <>
8   axial_viscous_damping = <Pa*s>
9   shear_viscous_damping = <Pa*s>
10  peak_friction_coefficient_limit = <>
11  peak_friction_coefficient_rate_of_decrease = <.>
12  residual_friction_coefficient = <.>
13  shear_zone_thickness = <m>
14  contact_plane_vector = (<.>, <.>, <.> );

```

and/or;

```

1 add element # <.> type StressBasedCoupledHardContact_NonLinHardSoftShear
2   with nodes (<.>, <.>)
3   axial_stiffness = <Pa>
4   axial_penalty_stiffness = <Pa>
5   initial_shear_stiffness = <Pa>
6   rate_of_softening = <>
7   size_of_peak_plateau = <>
8   axial_viscous_damping = <Pa*s>
9   shear_viscous_damping = <Pa*s>
10  peak_friction_coefficient_limit = <>
11  peak_friction_coefficient_rate_of_decrease = <.>
12  residual_friction_coefficient = <.>
13  shear_zone_thickness = <m>
14  surface_vector_relative_tolerance = <.>;

```

The axial stress σ_a and axial stiffness E_a in defined as

$$\begin{aligned}\sigma_a &= b * \epsilon_a \\ E_a &= b\end{aligned}\tag{1.22}$$

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

$$\Delta\mu = -\frac{n * R_s(\mu_p - \mu_r)}{(\pi/2)^n \theta^{1/n-1}} * \cos^2\theta \Delta\gamma^p \quad (1.23)$$

$$\theta = \frac{\mu_p - \mu}{\mu_p - \mu_r} (\pi/2)^n \quad (1.24)$$

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.

$$\Delta\gamma^p = \sqrt{\Delta\epsilon_{ij}^p \Delta\epsilon_{ij}^p} \quad (1.25)$$

- `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 1.24.
- `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

$$\mu_p = \max(\mu_{p0}, \mu_{p0} - k * \log(\sigma_a/P_0)) \quad (1.26)$$

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](#).

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

The command is:

```

1 add element # <.> type StressBasedCoupledSoftContact_ElPP1Shear
2   with nodes (<.>, <.>)
3   initial_axial_stiffness = <Pa>
4   stiffening_rate = <>
5   max_axial_stiffness = <Pa>
6   axial_penalty_stiffness = <Pa>
7   initial_shear_stiffness = <Pa>
8   axial_viscous_damping = <Pa*s>
9   shear_viscous_damping = <Pa*s>
10  residual_friction_coefficient = <.>
11  shear_zone_thickness = <m>
12  contact_plane_vector = (<.>, <.>, <.> );

```

and/or;

```

1 add element # <.> type StressBasedCoupledSoftContact_ElPP1Shear
2   with nodes (<.>, <.>)
3   initial_axial_stiffness = <Pa>
4   stiffening_rate = <>
5   max_axial_stiffness = <Pa>
6   axial_penalty_stiffness = <Pa>
7   initial_shear_stiffness = <Pa>
8   axial_viscous_damping = <Pa*s>
9   shear_viscous_damping = <Pa*s>
10  residual_friction_coefficient = <.>
11  shear_zone_thickness = <m>
12  surface_vector_relative_tolerance = <.>;

```

The axial stress σ_a and axial stiffness E_a in defined as

$$\begin{aligned}\sigma_a &= b * \exp(a * \epsilon_a) * \epsilon_a \\ E_a &= \max(b * \exp(a * \epsilon_a) * (1 + a * \epsilon_a), E_{max})\end{aligned}\tag{1.27}$$

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 ??.

Modeling, Finite Element: Stress Based Coupled Soft Contact/Interface/Joint Element with Non-linear Hardening Shear Behavior

The command is:

```

1 add element # <.> type StressBasedCoupledSoftContact_NonLinHardShear
2   with nodes (<.>, <.>)
3   initial_axial_stiffness = <Pa>
4   stiffening_rate = <>
5   max_axial_stiffness = <Pa>
6   axial_penalty_stiffness = <Pa>
7   initial_shear_stiffness = <Pa>
8   axial_viscous_damping = <Pa*s>
9   shear_viscous_damping = <Pa*s>
10  residual_friction_coefficient = <.>
11  shear_zone_thickness = <m>
12  contact_plane_vector = (<.>, <.>, <.> );

```

and/or;

```

1 add element # <.> type StressBasedCoupledSoftContact_NonLinHardShear
2   with nodes (<.>, <.>)
3   initial_axial_stiffness = <Pa>
4   stiffening_rate = <>
5   max_axial_stiffness = <Pa>
6   axial_penalty_stiffness = <Pa>
7   initial_shear_stiffness = <Pa>
8   axial_viscous_damping = <Pa*s>
9   shear_viscous_damping = <Pa*s>
10  residual_friction_coefficient = <.>
11  shear_zone_thickness = <m>
12  surface_vector_relative_tolerance = <.>;

```

The axial stress σ_a and axial stiffness E_a in defined as

$$\begin{aligned}\sigma_a &= b * \exp(a * \epsilon_a) * \epsilon_a \\ E_a &= \max(b * \exp(a * \epsilon_a) * (1 + a * \epsilon_a), E_{max})\end{aligned}\tag{1.28}$$

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](#).

Modeling, Finite Element: Stress Based Coupled Soft Contact/Interface/Joint Element with Non-linear Hardening and Softening Shear Behavior

The command is:

```

1 add element # <.> type StressBasedCoupledSoftContact_NonLinHardSoftShear
2   with nodes (<.>, <.>)
3   initial_axial_stiffness = <Pa>
4   stiffening_rate = <>
5   max_axial_stiffness = <Pa>
6   axial_penalty_stiffness = <Pa>
7   initial_shear_stiffness = <Pa>
8   rate_of_softening = <>
9   size_of_peak_plateau = <>
10  axial_viscous_damping = <Pa*s>
11  shear_viscous_damping = <Pa*s>
12  peak_friction_coefficient_limit = <>
13  peak_friction_coefficient_rate_of_decrease = <.>
14  residual_friction_coefficient = <.>
15  shear_zone_thickness = <m>
16  contact_plane_vector = (<.>, <.>, <.> );

```

and/or;

```

1 add element # <.> type StressBasedCoupledSoftContact_NonLinHardSoftShear
2   with nodes (<.>, <.>)
3   initial_axial_stiffness = <Pa>
4   stiffening_rate = <>
5   max_axial_stiffness = <Pa>
6   axial_penalty_stiffness = <Pa>
7   initial_shear_stiffness = <Pa>
8   rate_of_softening = <>
9   size_of_peak_plateau = <>
10  axial_viscous_damping = <Pa*s>
11  shear_viscous_damping = <Pa*s>
12  peak_friction_coefficient_limit = <>
13  peak_friction_coefficient_rate_of_decrease = <.>
14  residual_friction_coefficient = <.>
15  shear_zone_thickness = <m>
16  surface_vector_relative_tolerance = <.>;

```

The axial stress σ_a and axial stiffness E_a in defined as

$$\begin{aligned}\sigma_a &= b * \exp(a * \epsilon_a) * \epsilon_a \\ E_a &= \max(b * \exp(a * \epsilon_a) * (1 + a * \epsilon_a), E_{max})\end{aligned}\tag{1.29}$$

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

$$\Delta\mu = -\frac{n * R_s(\mu_p - \mu_r)}{(\pi/2)^n \theta^{1/n-1}} * \cos^2\theta \Delta\gamma^p \quad (1.30)$$

$$\theta = \frac{\mu_p - \mu}{\mu_p - \mu_r} (\pi/2)^n \quad (1.31)$$

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.

$$\Delta\gamma^p = \sqrt{\Delta\epsilon_{ij}^p \Delta\epsilon_{ij}^p} \quad (1.32)$$

- **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 1.31.

- `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

$$\mu_p = \max(\mu_{p0}, \mu_{p0} - k * \log(\sigma_a/P_0)) \quad (1.33)$$

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.

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](#)).

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](#)).

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](#)).

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](#)).

Modeling, Damping: Adding Rayleigh Damping

First, define the Rayleigh damping.

```
1 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.

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

```
1 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.

Modeling, Damping: Adding 3rd Order Caughey Damping

First, define the 3rd-order Caughey damping

```
1 add damping # <.> type Caughey3rd
2 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.

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

```
1 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.

Modeling, Damping: Adding 4th Caughey Damping

First, define the 4th-order Caughey damping

```
1 add damping # <.> type Caughey4th
2 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.

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

```
1 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.

Modeling, Constraints and Supports: Adding Constraints or Supports

```
1 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;`

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.

```
1 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,

```
1 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.

Modeling, Constraints and Supports: Free Constraint or Support

Free the specified DOFs on a designated node.

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

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

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

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

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.

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

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

Remove the tied/connected nodes equal_dofs.

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

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.

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

Modeling, Acceleration Field: Adding Acceleration/Inertia Field

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

Example adding acceleration induced loading field for (some) elements

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

NOTE: see note on page [189](#) for command

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


Modeling, Loads: Nodal Loads

The general signature to add loads is

```
1 add load # <.> to node # <.>
2   type <load type> <direction> = <force_amplitude>
3   {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:

¹Applies to solid phase only when connected to coupled elements

²Applies 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](#)).

³For elements with rotational DOFs, i.e. beams, shells

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

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:

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

```
1 add load # <.> to node # <.> type path_time_series
2   FORCETYPE = <force or moment scale factor>
3   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`).

Modeling, Loads: Nodal Loads From Reactions

Loads can be added from reactions.

The command is:

```
1 add load # <.> to node # <.> type from_reactions;  
2 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:

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

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

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

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

Modeling, Loads: Selfweight Element Load

```
1 add load # <.>
2 to element # <.>
3 type self_weight
4 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 [184](#).

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.

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

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

```
1 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...

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>);
```


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

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

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

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

Surface of 20 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>);
```

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>;
```

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>);
```

Modeling, Loads: Removing Loads

Loads can be removed using:

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

Modeling, Loads: Domain Reduction Method, DRM

```

1 add load # <.> type domain reduction method hdf5_file = <string>;
2 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.(1.5), eight components are required for the DRM input.

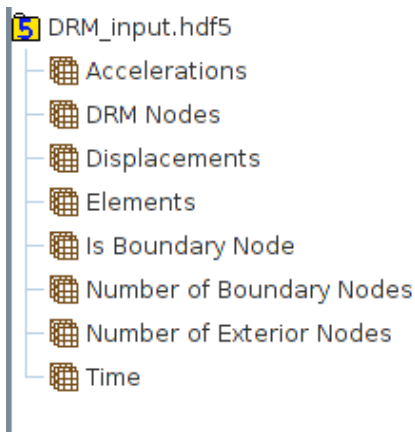


Figure 1.5: Components of DRM input in HDF5 format.

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 u_x , u_y , and u_z), the first three rows represent the input displacements on first DRM node in directions of u_x , u_y , and u_z . 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.

```

1 # Created by Jose Antonio Abell Mena
2 # This file reads old-format DRM input files and translates ↵
   them into new
3 # HDF5-based format.
4 #
5
6 # This file produces a rigid body input to the DRM layer. That ↵
   is, all DRM
7 # nodes have same X-direction displacement and acceleration. In ↵
   this case a
8 # sine wave is used. This is not realistic, its just for ↵
   demonstration
9 # purposes. DRM won't work in this case but can be used to verify ↵
   input if a
10 # pseudo-static analysis is done (zero density on all elements and ↵
    apply loads
11 # with transient analysis.)
12 # For real input motions, produced using some other means, say ↵
    SW4, fk,
13 # Hisada or MS ESSI program, this simple program can be used as an ↵
    example,
14 # where DRM input file format used here can be (re) used, while ↵
    those real
15 # motions are read form output of above mentioned programs.
16
```

```

17 import scipy as sp
18 import h5py
19 import time
20
21 # Write elements and nodes data
22 elements = sp.loadtxt("DRMelements.txt",dtype=sp.int32)
23 exterior_nodes = sp.loadtxt("DRMexterior.txt",dtype=sp.int32)
24 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
76 acc = h5file.create_dataset("Accelerations", (3*Nt,len(t)), ←
    dtype=sp.double)
77 dis = h5file.create_dataset("Displacements", (3*Nt,len(t)), ←
    dtype=sp.double)
78
79 for node_index in range(Nt):
80     acc[3*node_index,:] = a
81     acc[3*node_index+1,:] = 0*a #Zero acceleration in y and z
82     acc[3*node_index+2,:] = 0*a
83     dis[3*node_index,:] = d
84     dis[3*node_index+1,:] = 0*d #Zero displacement in y and z
85     dis[3*node_index+2,:] = 0*d
86
87 h5file.close()
```

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

```

1 add wave field # <.> with
2   acceleration_filename = <string>
3   unit_of_acceleration = <L/T^2>
4   displacement_filename = <string>
5   unit_of_displacement = <L>
6   add_compensation_time = <T>
7   motion_depth = <L>
8   monitoring_location = ←
   <within_soil_layer|equivalent_rock_outcropping>
9   soil_profile_filename = <string>
10  unit_of_Vs = <L/T>
11  unit_of_rho = <M/L^3>
12  unit_of_damping = <absolute|percent>
13  unit_of_thickness = <L>
14  ;

```

Example adding a wave field

```

1 add wave field # 1 with
2   acceleration_filename = "acc.txt"
3   unit_of_acceleration = 1 * m/s^2
4   displacement_filename = "dis.txt"
5   unit_of_displacement = 1 * m
6   add_compensation_time = 0.5 * s
7   motion_depth = 0 * m
8   monitoring_location = within_soil_layer
9   soil_profile_filename = "soil_profile.txt"
10  unit_of_Vs = 1 * m/s
11  unit_of_rho = 1 * kg/m^3
12  unit_of_damping = absolute
13  unit_of_thickness = 1*m
14  ;

```

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.

1	// Vs	rho	damp	thickness
2	200	2000	0.03	35
3	250	2000	0.04	35
4	2000	2400	0.05	
5				

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.

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

One example of deconvolution is

```
1 generate wave propagation results of wave field
2 # 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

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.

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

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

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

One example of deconvolution to DRM is

```
1 generate DRM motion file from wave field
2   # 1 in direction ux
3   soil_surface at z = 0*m
4   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.

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.

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

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

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

One example of deconvolution to DRM is

```
1 generate DRM force file from wave field
2   # 1 in direction ux
3   soil_surface at z = 0*m
4   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.

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

```

1 add wave field # <.> type inclined_plane_wave with
2   anticlockwise_angle_of_SV_wave_plane_from <x|y|z> = <degrees>
3   SV_incident_magnitude = <L^2>
4   SV_incident_angle = <degrees>
5   SV_incident_frequency = <1/T>
6   motion_time_step = <T>
7   number_of_time_steps = <.>
8   soil_profile_filename = <string>
9   soil_surface at <x|y|z> = <L>
10  unit_of_vs_and_vp = <L/T>
11  unit_of_rho = <M/L^3>
12  unit_of_damping = <absolute|percent>
13  unit_of_thickness = <L>
14 ;

```

Example of adding an inclined plane wave field

```

1 add wave field # 1 type inclined_plane_wave with
2   anticlockwise_angle_of_SV_wave_plane_from x= 30
3   SV_incident_magnitude = 2*m^2
4   SV_incident_angle = 60
5   SV_incident_frequency = 5/s
6   motion_time_step = 0.01*s
7   number_of_time_steps = 600
8   soil_profile_filename = "soil.txt"
9   soil_surface at z = 0*m
10  unit_of_vs_and_vp = 1*m/s
11  unit_of_rho = 1*kg/m^3
12  unit_of_damping = absolute
13  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 1.7, 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 $|u|$ is the displacement magnitude, ϕ is potential magnitude, ω is incident angular frequency and V_s is the

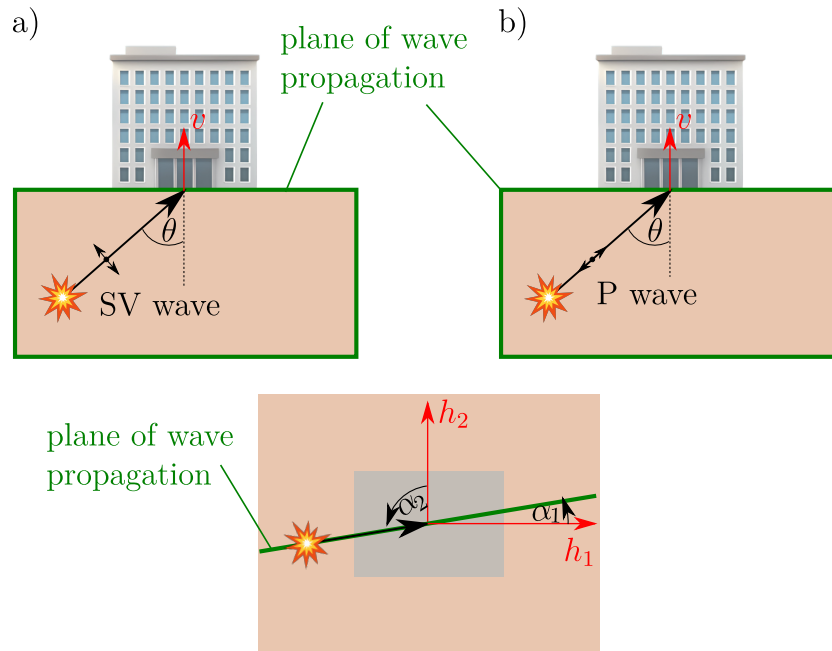


Figure 1.6: Orientation of an inclined plane wave with respect to the vertical, v , and horizontal, h_1, h_2 , directions: a) a SV wave and b) a P-wave. The user defines v, h_1, h_2 directions. Either α_1 or α_2 can be input into the DSL command.

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 1.7, 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 five 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.

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	
5						

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

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

```

1 add wave field # <.> type inclined_plane_wave with
2   anticlockwise_angle_of_SV_wave_plane_from <x|y|z> = <degrees>
3   SV_incident_acceleration_filename = <string>
4   unit_of_acceleration = <L/T^2>
5   SV_incident_displacement_filename = <string>
6   unit_of_displacement = <L>
7   SV_incident_angle = <degrees>
8   add_compensation_time = <T>
9   source_location = (<T>, <T>, <T>)
10  soil_profile_filename = <string>
11  soil_surface at <x|y|z> = <L>
12  unit_of_vs_and_vp = <L/T>
13  unit_of_rho = <M/L^3>
14  unit_of_damping = <absolute|percent>
15  unit_of_thickness = <L>
16 ;

```

Example of adding an inclined plane wave field

```

1 add wave field # 1 type inclined_plane_wave with
2   anticlockwise_angle_of_SV_wave_plane_from x = 0
3   SV_incident_acceleration_filename = "Kobe_acc.txt"
4   unit_of_acceleration = 1*m/s^2
5   SV_incident_displacement_filename = "Kobe_disp.txt"
6   unit_of_displacement = 1*m
7   SV_incident_angle = 15
8   add_compensation_time = 0.5*s
9   source_location = (-150*m, 0*m, -100*m)
10  soil_profile_filename = "soil_profile.txt"
11  soil_surface at z = 0*m
12  unit_of_vs_and_vp = 1*m/s
13  unit_of_rho = 1*kg/m^3
14  unit_of_damping = absolute
15  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 1.7, the anticlockwise_angle_of_SV_wave_plane_from x axis is α .
- SV_incident_acceleration_filename is the filename of a plain text file, which contains the acceler-

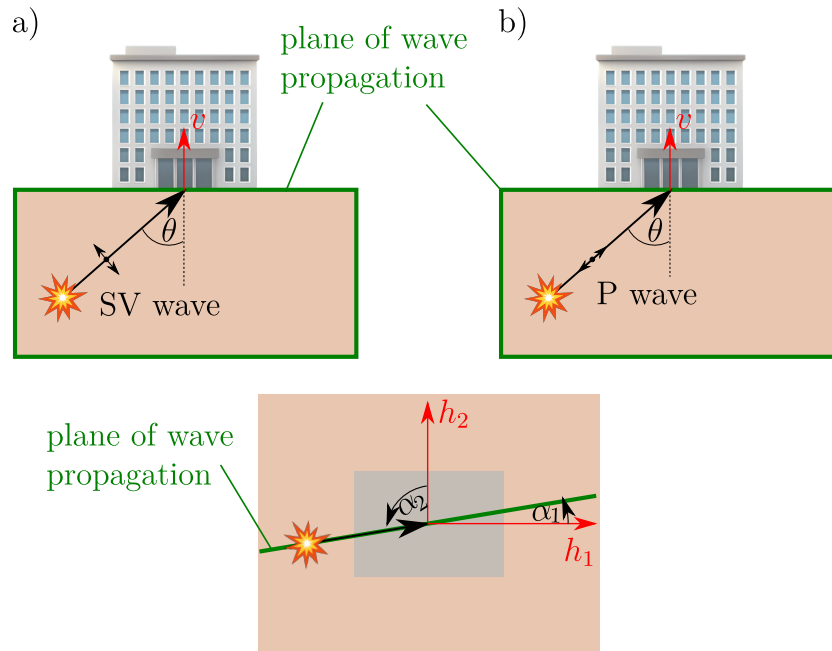


Figure 1.7: Orientation of an inclined plane wave with respect to the vertical, v , and horizontal, h_1, h_2 , directions: a) a SV wave and b) a P-wave. The user defines v, h_1, h_2 directions. Either α_1 or α_2 can be input into the DSL command.

ation 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 1.7, 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 five 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	
5						

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

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

```

1 add wave field # <.> type inclined_plane_wave with
2   anticlockwise_angle_of_P_wave_plane_from <x|y|z> = <degrees>
3   P_incident_magnitude = <L^2>
4   P_incident_angle = <degrees>
5   P_incident_frequency = <1/T>
6   motion_time_step = <T>
7   number_of_time_steps = <.>
8   soil_profile_filename = <string>
9   soil_surface at <x|y|z> = <L>
10  unit_of_vs_and_vp = <L/T>
11  unit_of_rho = <M/L^3>
12  unit_of_damping = <absolute|percent>
13  unit_of_thickness = <L>
14  ;

```

Example adding an inclined plane wave field

```

1 add wave field # 1 type inclined_plane_wave with
2   anticlockwise_angle_of_P_wave_plane_from x= 30
3   P_incident_magnitude = 2*m^2
4   P_incident_angle = 60
5   P_incident_frequency = 5/s
6   motion_time_step = 0.01*s
7   number_of_time_steps = 600
8   soil_profile_filename = "soil.txt"
9   soil_surface at z = 0*m
10  unit_of_vs_and_vp = 1*m/s
11  unit_of_rho = 1*kg/m^3
12  unit_of_damping = absolute
13  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 1.7, 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 1.7, 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 five 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.

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	
5						

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

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

```

1 add wave field # <.> type inclined_plane_wave with
2   anticlockwise_angle_of_P_wave_plane_from <x|y|z> = <degrees>
3   P_incident_acceleration_filename = <string>
4   unit_of_acceleration = <L/T^2>
5   P_incident_displacement_filename = <string>
6   unit_of_displacement = <L>
7   P_incident_angle = <degrees>
8   add_compensation_time = <T>
9   source_location = (<T>, <T>, <T>)
10  soil_profile_filename = <string>
11  soil_surface at <x|y|z> = <L>
12  unit_of_vs_and_vp = <L/T>
13  unit_of_rho = <M/L^3>
14  unit_of_damping = <absolute|percent>
15  unit_of_thickness = <L>
16 ;

```

Example of adding an inclined plane wave field

```

1 add wave field # 1 type inclined_plane_wave with
2   anticlockwise_angle_of_P_wave_plane_from x = 0
3   P_incident_acceleration_filename = "Kobe_acc.txt"
4   unit_of_acceleration = 1*m/s^2
5   P_incident_displacement_filename = "Kobe_disp.txt"
6   unit_of_displacement = 1*m
7   P_incident_angle = 15
8   add_compensation_time = 0.5*s
9   source_location = (-150*m, 0*m, -100*m)
10  soil_profile_filename = "soil_profile.txt"
11  soil_surface at z = 0*m
12  unit_of_vs_and_vp = 1*m/s
13  unit_of_rho = 1*kg/m^3
14  unit_of_damping = absolute
15  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 1.7, 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 acceler-

ation 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 1.7, 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 five 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	
5						

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

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

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

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

One example of generating inclined DRM motion is:

```
1 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.

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.

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

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

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.

```

1 add load # <.> type imposed random motions to node # <.> dof DOFTYPE
2 time_step = <T>
3 displacement_scale_unit = <L>
4 displacement_file = "filename"
5 velocity_scale_unit = <L/T>
6 velocity_file = "filename"
7 acceleration_scale_unit = <L/L^2>
8 acceleration_file = "filename"
9 penalty_stiffness = <N/L>
10 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.

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.

```
1 add load # <.> type imposed random motions to node # <.> dof DOFTYPE
2 time_step = <T>
3 uncertain_displacement = random field # <.>
4 displacement_scale_unit = <L>
5 penalty_stiffness = <N/L>
6 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.

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.

```
1 add load # <.> type imposed motion to node # <.> dof DOFTYPE
2 displacement_scale_unit = <displacement>
3 displacement_file = "filename"
4 velocity_scale_unit = <velocity>
5 velocity_file = "filename"
6 acceleration_scale_unit = <acceleration>
7 acceleration_file = "filename";
```

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

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:

```
1 add load # <.> type uniform acceleration to all nodes dof <.>
2   time_step = <T>
3   scale_factor = <L/T^2>
4   initial_velocity = <L/T>
5   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.

Modeling, Imposed Motions: Remove Imposed Motions

Motions can be removed using:

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

Modeling, Random Variable: Adding Gaussian Random Variables

Gaussian random variable can be added for probabilistic analysis.

The command is:

```
1 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:

```
1 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.

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:

```
1 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:

```
1 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 $x = 3\text{m}$, $y = 0\text{m}$ and $z = 0\text{m}$.

Modeling, Random Variable: Adding Lognormal Random Variables

Lognormal random variable can be added for probabilistic analysis.

The command is:

```
1 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:

```
1 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.

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:

```
1 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:

```
1 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 $x = 3\text{m}$, $y = 0\text{m}$ and $z = 0\text{m}$.

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

Lognormal random variable can be added for probabilistic analysis.

The command is:

```
1 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:

$$\ln(X) \sim N(\mu, \sigma) \quad (1.34)$$

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

$$m = e^{\mu + \sigma^2/2} \quad (1.35)$$

$$v = e^{2\mu + \sigma^2}(e^{\sigma^2} - 1) \quad (1.36)$$

For example:

```
1 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.

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:

```
1 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:

$$\ln(X) \sim N(\mu, \sigma) \quad (1.37)$$

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

$$m = e^{\mu + \sigma^2/2} \quad (1.38)$$

$$v = e^{2\mu + \sigma^2}(e^{\sigma^2} - 1) \quad (1.39)$$

For example:

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

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:

```
1 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:

```
1 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.

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:

```
1 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:

```
1 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 $x = 3\text{m}$, $y = 0\text{m}$ and $z = 0\text{m}$.

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:

```
1 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:

```
1 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.

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:

```
1 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:

```
1 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 $z = 0\text{m}$.

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:

```
1 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:

```
1 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.

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:

```
1 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:

```
1 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 $x = 3\text{m}$, $y = 0\text{m}$ and $z = 0\text{m}$.

Modeling, Random Variable: Remove Random Variables

Remove random variables.

The command is:

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

For example:

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

Remove random variable 2 from the analysis.

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:

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

where:

- order specifies the order of Hermite polynomial chaos expansion

For example:

```
1 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.

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:

```
1 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. PC_{ij} denotes the order of polynomial chaos dimension ξ_j that contributes to the i^{th} multidimensional 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:

```
1 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.

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:

```
1 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. PC_{ij} denotes the order of polynomial chaos dimension ξ_j that contributes to the i^{th} multidimensional 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:

```
1 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.

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:

```
1 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:

```
1 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.

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:

```
1 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:

```
1 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.

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:

```
1 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:

```
1 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:

```
1 //=====
2 // This file specify the global dimension index of local dimensions
3 // of Hermite PC basis
4 // File should have two columns separated by spaces:
5 // The first column is local KL dimension ID
6 // The second column is the global KL dimension ID
7 //=====
8 1 10
9 2 11
10 3 12
11 4 13
```

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:

```
1 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:

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

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

Modeling, Random Field: Adding Random Field with Zero Correlation

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

The command is:

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

Modeling, Random Field: Adding Random Field with Exponential Correlation

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

The command is:

```
1 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:

$$\rho(RV_i, RV_j) = \exp(-d/l_c) \quad (1.40)$$

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:

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

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

Modeling, Random Field: Adding Random Field with Triangular Correlation

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

The command is:

```
1 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:

$$\rho(RV_i, RV_j) = \max\{1 - d/l_c, 0\} \quad (1.41)$$

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:

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

Adds an triangular correlated random field number 1 with correlation length 10m.

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:

```
1 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:

$$\rho(RV_i, RV_j) = \exp(-d/l_c) * \cos(d/l_c) \quad (1.42)$$

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:

```
1 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.

Modeling, Random Field: Adding Random Field with Gaussian Correlation

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

The command is:

```
1 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:

$$\rho(RV_i, RV_j) = \exp(-d^2/l_c^2) \quad (1.43)$$

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:

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

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

Modeling, Random Field: Remove Random Fields

Remove random Fields.

The command is:

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

For example:

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

Remove random field 2 from the analysis.

Modeling, Random Field: Adding Random Variable to Random Field

Add random variable to random field.

The command is:

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

For example:

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

Adds random variable 2 to random field 1.

Modeling, Random Field: Remove Random Variable From Random Field

Remove random variable from random field.

The command is:

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

For example:

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

Remove random variable 2 from random field 1.

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:

```
1 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:

```
1 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.

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:

```
1 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:

```
1 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 2nd order Gaussian correlation kernel inversion is adopted.

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:

```
1 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:

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


and:

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

For example:

```
1 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:

```
1 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

```
1 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.

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:

```
1 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:

```
1 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".

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:

```
1 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 3rd order Gaussian correlation kernel inversion is adopted. The input marginal distribution and correlation information are defined in HDF5 file "PC_RF1.hdf5".

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:

```
1 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. PC_{ij} denotes the order of polynomial chaos dimension ξ_j that contributes to the i^{th} multidimensional 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:

```
1 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.

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:

```
1 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. PC_{ij} denotes the order of polynomial chaos dimension ξ_j that contributes to the i^{th} multidimensional 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:

```
1 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".

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:

```
1 add random field # <.> with <distribution_type> distribution
2 marginal_mean_file = "file_name"
3 marginal_standard_deviation_file = "file_name"
4 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.

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:

```
1 add triple product # <.> with Hermite polynomial chaos from random ↵  
   field (<.>, <.>, <.>);
```

For example:

```
1 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;

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:

```
1 add double product # <.> with Hermite polynomial chaos from random ↵  
   field (<.>, <.>);
```

For example:

```
1 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;

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:

```
1 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. $PC1_{ij}$ 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. $PC2_{ij}$ 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. $PC3_{ij}$ 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 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";

Modeling, Random Field: Generate Double Product of Hermite Polynomial Chaos Basis

The computation of double product of Hermite polynomial 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:

```
1 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. $PC1_{ij}$ 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. $PC2_{ij}$ 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";

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:

```
1 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. $PC1_{ij}$ 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. $PC2_{ij}$ 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. $PC3_{ij}$ 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".

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:

```
1 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. $PC1_{ij}$ 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. $PC2_{ij}$ 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 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".

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:

```
1 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:

```
1 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.

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:

```
1 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):
 - 1st entry: Real-ESSI node ID;
 - 2nd entry: x coordinate of Real-ESSI interface node;
 - 3rd entry: y coordinate of Real-ESSI interface node;
 - 4th entry: z coordinate of Real-ESSI interface node;

For example:

```
1 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:

```
1 //=====
2 // Files contains information about ESSI nodes at solid fluid ↵
   interface, have 4 columns
3 // 1st column: ESSI node ID
4 // 2nd column: coordinate x
5 // 3rd column: coordinate y
6 // 4th column: coordinate z
7 //=====
8 1 0.00 0.00 0.00
9 12 30.00 0.00 0.00
10 33 0.00 0.00 10.00
11 ...
```

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:
 - 1st entry: Real-ESSI interface face ID;
 - 2nd entry: Real-ESSI element ID that contains the Real-ESSI interface face;
 - 3rd entry: Real-ESSI node ID for Real-ESSI interface face, i.e., a brick face, a quad, vertex 1;
 - 4th entry: Real-ESSI node ID for Real-ESSI interface face, i.e., a brick face, a quad, vertex 2;
 - 5th entry: Real-ESSI node ID for Real-ESSI interface face, i.e., a brick face, a quad, vertex 3;
 - 6th 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:

```
1 //=====
2 // Files contains information about ESSI faces (quads) at solid fluid ←
  interface, have 6 columns
3 // 1st column: ESSI face ID
```

```
4 // 2nd column: ESSI Element ID that ESSI face belongs to
5 // 3rd column: ESSI node ID for quad vertex 1
6 // 4th column: ESSI node ID for quad vertex 2
7 // 5th column: ESSI node ID for quad vertex 3
8 // 6th column: ESSI node ID for quad vertex 4
9 //=====
10 1 276 1 25 272 5
11 2 277 25 26 273 272
12 3 278 26 27 274 273
13 4 279 27 28 275 274
14 ...
```


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:

```
1 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):
 - 1st entry: OpenFOAM node ID;
 - 2nd entry: x coordinate of OpenFOAM interface node;
 - 3rd entry: y coordinate of OpenFOAM interface node;
 - 4th entry: z coordinate of OpenFOAM interface node;

For example:

```
1 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:

```
1 //=====
2 // Files contains information about Foam nodes at solid fluid ↔
   interface, have 4 columns
3 // 1st column: Foam node ID
4 // 2nd column: coordinate x
5 // 3rd column: coordinate y
6 // 4th column: coordinate z
7 //=====
8 1 0.00 0.00 0.00
9 12 30.00 0.00 0.00
10 33 0.00 0.00 10.00
11 ...
```

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):
 - 1st entry: OpenFOAM face ID;
 - 2nd entry: OpenFOAM node ID for OpenFOAM interface face, i.e., a quad, vertex 1;
 - 3rd entry: OpenFOAM node ID for OpenFOAM interface face, i.e., a quad, vertex 2;
 - 4th entry: OpenFOAM node ID for OpenFOAM interface face, i.e., a quad, vertex 3;
 - 5th 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 ...
```

1.3.5 Simulation

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 are 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...

Simulation, Solvers: Parallel Solvers

```
1 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:

```
1 define solver parallel petsc "-ksp_type preonly -pc_type lu" ;
2 define solver parallel petsc "-pc_type lu ←
  -pc_factor_mat_solver_package mumps" ;
3 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:

```
1 define solver parallel petsc "-ksp_type gmres -pc_type jacobi";
2 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 1.1 and 1.2 on next pages present a full set of options for iterative solvers and preconditioners.

Table 1.1: 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 1.2: 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>).

Simulation: Static Solution Advancement

```
1 simulate <.> steps using static algorithm;
```

Simulation: Dynamic Solution Advancement with the Constant Time Step

```
1 simulate <.> steps using transient algorithm time_step = <T>;
```

Simulation: Dynamic Solution Advancement with Variable Time Step

```
1  simulate <.> steps using variable transient algorithm
2      time_step = <T>
3      minimum_time_step = <.>
4      maximum_time_step = <.>
5      number_of_iterations = <.>
```

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](#).

Simulation: Displacement Control

```
1 define static integrator displacement_control using node # <.> dof ↔  
    DOFTYPE increment <L>;
```

Simulation: Load, Control, Factor Increment

```
1 define load factor increment <.>;
```

Simulation: Dynamic Integrator, Newmark Method

```
1 define dynamic integrator Newmark with gamma = <.> beta = <.>;
```

See also the list of the reserved keywords from Section [1.7](#) on page [347](#).

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 [1.7](#) on page [347](#).

Simulation: Absolute Convergence Criteria

```

1 define convergence test
2   < Absolute_Norm_Unbalanced_Force | ↔
   Absolute_Norm_Displacement_Increment >
3   tolerance = <.>
4   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.

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.

Simulation: Relative Convergence Criteria

```

1 define convergence test ⇐
    <Relative_Norm_Unbalanced_Force|Relative_Norm_Displacement_Increment>
2 tolerance = <.>
3 minimum_absolute_tolerance = <.>
4 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.

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.

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:

```
1 define NDMaterial constitutive integration algorithm Forward_Euler;
```

```
1 define NDMaterial constitutive integration algorithm ←
  Forward_Euler_Subincrement
2   number_of_subincrements = <.> ;
```

```
1 define NDMaterial constitutive integration algorithm Backward_Euler
2   yield_function_relative_tolerance = <.>
3   stress_relative_tolerance = <.>
4   maximum_iterations = <.>;
```

```
1 define NDMaterial constitutive integration algorithm ←
  Backward_Euler_Subincrement
2   yield_function_relative_tolerance = <.>
3   stress_relative_tolerance = <.>
4   maximum_iterations = <.>
5   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 $1E-4$.

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.

```

1 atmospheric_pressure = 101325*Pa;
2
3 pstart = 3000*kPa;
4
5 //SAniSand 2004 calibration for Toyoura Sand.
6 add material # 1 type sanisand2004
7     mass_density = 2100.0 * kg / m^3
8     e0 = 0.735
9     sanisand2004_G0 = 125.                                poisson_ratio = 0.05
10    sanisand2004_Pat = atmospheric_pressure
11    sanisand2004_p_cut = 0.1*atmospheric_pressure
12    sanisand2004_Mc = 1.25                                sanisand2004_c = 0.712
13    sanisand2004_lambda_c = 0.019                        sanisand2004_xi = 0.7
14    sanisand2004_ec_ref = 0.934                          sanisand2004_m = 0.01
15    sanisand2004_h0 = 7.05                                sanisand2004_ch = 0.968
16    sanisand2004_nb = 1.1                                sanisand2004_A0 = 0.704
17    sanisand2004_nd = 3.5                                sanisand2004_z_max = 4.
18    sanisand2004_cz = 600.
19    initial_confining_stress = 1*Pa ;
20
21 simulate constitutive testing DIRECT_STRAIN
22     use material # 1
23     scale_factor = 1.
24     series_file = "increments.txt"
25     sigma0 = ( -pstart*kPa , -pstart*kPa , -pstart*kPa , 0*Pa , 0*Pa ←
, 0*Pa )
26     verbose_output = 1;
27
28
29 if(SIMULATE_EXIT_FLAG == 0)
30 {
31     print "All Good!";
32 }
33 else
34 {
35     print "Something went wrong. Error code = ";
36     print SIMULATE_EXIT_FLAG;
37 }
38
39 bye;
```

The above simulation fails because the integration method for the constitutive model is not set (see 1.3.5). Therefore, the second branch of the 'if' statement will execute.

Simulation: Save State

Save ESSI system state to a file, to prepare for a restart.

```
1 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:

```
1 model name "ESSI_model";  
2 . . .  
3 new loading stage "Loading_stage_2";  
4 . . .  
5 simulate 100 steps using transient algorithm time_step = 0.005*s;  
6  
7 save model;
```

will be saved in file with the following name:

```
1 ESSI_model_Loading_stage_2_at_time_0.5second_RESTART.essi
```

since there were 100 steps with $\Delta t = 0.005s$, and that advances the solution to $t = 0.5s$.

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 [304](#).

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.

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 1.8, 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.

```

1  step=0;
2  Nsteps = 100;
3  define load factor increment 0.01; // Start with load-control
4
5  simulation_status=simulate 1 steps using static algorithm;
6
7  while (step<(Nsteps-1))
8  {
9      if(simulation_status>=0) // Converged, continue using load-control
10     {
11         simulation_status=simulate 1 steps using static algorithm;
12     }
13     else // Not converged, so change to displacement-control
14     {
15         define static integrator displacement_control using node # 1 ←
16         dof ux increment 1E-3*m;
17         simulate 1 steps using static algorithm;
18     }
19     step=step+1;
20 }
```

Figure 1.8: 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, 1991](#)) 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

```

1  model name "vm";
2  add material # 1 type vonMises
3      mass_density = 0.0*kg/m^3
4      elastic_modulus = 2E7*N/m^2
5      poisson_ratio = 0.0
6      von_mises_radius = 1E5*Pa
7      kinematic_hardening_rate = 0*Pa
8      isotropic_hardening_rate = 0*Pa;
9  // define the node:
10 add node # 1 at (0*m,0*m,1*m) with 3 dofs;
11 add node # 2 at (1*m,0*m,1*m) with 3 dofs;
12 add node # 3 at (1*m,1*m,1*m) with 3 dofs;
13 add node # 4 at (0*m,1*m,1*m) with 3 dofs;
14 add node # 5 at (0*m,0*m,0*m) with 3 dofs;
15 add node # 6 at (1*m,0*m,0*m) with 3 dofs;
16 add node # 7 at (1*m,1*m,0*m) with 3 dofs;
17 add node # 8 at (0*m,1*m,0*m) with 3 dofs;
18 // Define the element.
19 add element # 1 type 8NodeBrick using 2 Gauss points each direction ←
    with nodes (1, 2, 3, 4, 5, 6, 7, 8) use material # 1;
20
21 new loading stage "shearing";
22 //fix the bottom totally
23 fix node # 5 dofs all;
24 fix node # 6 dofs all;
25 fix node # 7 dofs all;
26 fix node # 8 dofs all;
27 // Fix the other 2 directions on the top.
28 fix node # 1 dofs uy uz ;
29 fix node # 2 dofs uy uz ;
30 fix node # 3 dofs uy uz ;
31 fix node # 4 dofs uy uz ;
32 add load # 101 to node # 1 type linear Fx = 40 * kN;
33 add load # 102 to node # 2 type linear Fx = 40 * kN;
34 add load # 103 to node # 3 type linear Fx = 40 * kN;
35 add load # 104 to node # 4 type linear Fx = 40 * kN;
36 define solver UMFpack;
37 //define algorithm With_no_convergence_check ←
    ;Norm_Displacement_Increment;Norm_Unbalance
38 define convergence test Absolute_Norm_Displacement_Increment
39     tolerance = 1E-3
40     maximum_iterations = 5
41     ;
42 define algorithm Newton;
43 define NDMaterial constitutive integration algorithm Backward_Euler
44     yield_function_relative_tolerance = 1E-7
45     stress_relative_tolerance = 1E-7
46     maximum_iterations = 100;
47
48 // ←
    *****
49 step=0;

```

```

50 Nsteps = 10;
51 define load factor increment 1/Nsteps; // Start with load-control
52 // ↵
53 // Simulate with status check:
54 // ↵
55 mystatus=simulate 1 steps using static algorithm;
56 while (step<(Nsteps-1)){
57     step=step+1;
58     if(mystatus>=0){ // Converged, so continue using load-control
59         mystatus=simulate 1 steps using static algorithm;
60     }
61     else{ // Not converged, so change to displacement-control
62         define static integrator displacement_control using node # 1 ↵
63         dof ux increment 1E-3*m ;
64         simulate 1 steps using static algorithm;
65     }
66 }
67 bye;

```

Resulting terminal output, showing a switch between two solution control mechanisms is provided below:

```

1
2
3
4 The Finite Element Interpreter
5
6 MS ESSI
7 Earthquake Soil Structure Interaction Simulator
8
9 Sequential processing mode.
10
11 Version Name : Academic Version. Release date: Apr 14 2017 at ↵
12 17:19:55. Tag: c24e557a56
13 Version Branch : yuan
14 Compile Date : Apr 15 2017 at 20:28:11
15 Compile User : yuan
16 Compile Sysinfo: cml01 4.4.0-72-generic x86_64 GNU/Linux
17 Runtime User : Runtime Sysinfo: Time Now : Apr 15 2017 at ↵
18 21:15:40
19 Static startup tips:
20 * Remember: Every command ends with a semicolon ';'.
21 * Type 'quit;' or 'exit;' to finish.
22 * Run 'essi -h' to see available command line options.
23 Including: "main.fei"
24
25
26 Model name is being set to "vm"

```

```

27
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.....
38 Checking constraint ←
    handler.....P
39 Checking ←
    numberer.....
40 Checking analysis ←
    algorithm.....
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.....
68 Checking constraint ←

```

```

        handler..... P
69 Checking ↵
        numberer.....
70 Checking analysis ↵
        algorithm.....
71 Checking system of equation ↵
        handler..... Pass!
72 Checking static integration ↵
        handler..... Pass!
73
74 Static Analysis: [1      /1      ]
75 [iteration 1      /5      ] Convergence Test: Absolute Norm ↵
        Displacement Increment::(tol: 0.001)
76             Absolute Norm deltaF: 2.7285e-12
77             Absolute Norm deltaU: 0.0032
78             Average Norm deltaF: 1.3642e-12
79             Average Norm deltaU: 0.0016
80             Relative Norm deltaF: 3.4106e-16
81             Relative Norm deltaU: 0.0032
82 [iteration 2      /5      ] Convergence Test: Absolute Norm ↵
        Displacement Increment::(tol: 0.001)
83             Absolute Norm deltaF: 3.5225e-12
84             Absolute Norm deltaU: 7.8429e-19
85             Average Norm deltaF: 1.7612e-12
86             Average Norm deltaU: 3.9214e-19
87             Relative Norm deltaF: 4.4031e-16
88             Relative Norm deltaU: 2.4509e-16
89
90 > Analysis End ↵
        -----
91
92 Starting sequential static multi-step analysis
93 =====
94 Creating analysis ↵
        model.....
95 Checking constraint ↵
        handler..... P
96 Checking ↵
        numberer.....
97 Checking analysis ↵
        algorithm.....
98 Checking system of equation ↵
        handler..... Pass!
99 Checking static integration ↵
        handler..... Pass!
100
101 Static Analysis: [1      /1      ]
102 [iteration 1      /5      ] Convergence Test: Absolute Norm ↵
        Displacement Increment::(tol: 0.001)
103             Absolute Norm deltaF: 1.819e-12
104             Absolute Norm deltaU: 0.0032
105             Average Norm deltaF: 9.0949e-13

```

```

106             Average Norm deltaU: 0.0016
107             Relative Norm deltaF: 2.2737e-16
108             Relative Norm deltaU: 0.0032
109 [iteration 2      /5      ] Convergence Test: Absolute Norm ↔
            Displacement Increment::(tol: 0.001)
110             Absolute Norm deltaF: 2.5724e-12
111             Absolute Norm deltaU: 5.0807e-19
112             Average Norm deltaF: 1.2862e-12
113             Average Norm deltaU: 2.5403e-19
114             Relative Norm deltaF: 3.2155e-16
115             Relative Norm deltaU: 1.5877e-16
116
117 > Analysis End ↔
            -----
118
119 Starting sequential static multi-step analysis
120 =====
121 Creating analysis ↔
            model.....
122 Checking constraint ↔
            handler.....
123 Checking ↔
            numberer.....
124 Checking analysis ↔
            algorithm.....
125 Checking system of equation ↔
            handler..... Pass!
126 Checking static integration ↔
            handler..... Pass!
127
128 Static Analysis: [1      /1      ]
129 [iteration 1      /5      ] Convergence Test: Absolute Norm ↔
            Displacement Increment::(tol: 0.001)
130             Absolute Norm deltaF: 3132.5
131             Absolute Norm deltaU: 0.0032
132             Average Norm deltaF: 1566.2
133             Average Norm deltaU: 0.0016
134             Relative Norm deltaF: 0.39156
135             Relative Norm deltaU: 0.0032
136 [iteration 2      /5      ] Convergence Test: Absolute Norm ↔
            Displacement Increment::(tol: 0.001)
137             Absolute Norm deltaF: 3132.5
138             Absolute Norm deltaU: 0.001253
139             Average Norm deltaF: 1566.2
140             Average Norm deltaU: 0.0006265
141             Relative Norm deltaF: 0.39156
142             Relative Norm deltaU: 0.39156
143 [iteration 3      /5      ] Convergence Test: Absolute Norm ↔
            Displacement Increment::(tol: 0.001)
144             Absolute Norm deltaF: 3132.5
145             Absolute Norm deltaU: 0.001253
146             Average Norm deltaF: 1566.2

```

```

147         Average Norm deltaU: 0.0006265
148         Relative Norm deltaF: 0.39156
149         Relative Norm deltaU: 0.39156
150     [iteration 4      /5      ] Convergence Test: Absolute Norm ↵
        Displacement Increment::(tol: 0.001)
151         Absolute Norm deltaF: 3132.5
152         Absolute Norm deltaU: 0.001253
153         Average Norm deltaF: 1566.2
154         Average Norm deltaU: 0.0006265
155         Relative Norm deltaF: 0.39156
156         Relative Norm deltaU: 0.39156
157     [iteration 5      /5      ] Convergence Test: Absolute Norm ↵
        Displacement Increment::(tol: 0.001)
158         Absolute Norm deltaF: 3132.5
159         Absolute Norm deltaU: 0.001253
160         Average Norm deltaF: 1566.2
161         Average Norm deltaU: 0.0006265
162         Relative Norm deltaF: 0.39156
163         Relative Norm deltaU: 0.39156
164     [iteration      5/5      ] Convergence Test: Absolute Norm ↵
        Displacement Increment::(tol: 0.001)      !!!FAILED TO CONVERGE!!! ↵
        [EXITING...]
165         Absolute Norm deltaF: 3132.5
166         Absolute Norm deltaU: 0.001253
167         Average Norm deltaF: 1566.2
168         Average Norm deltaU: 0.0006265
169         Relative Norm deltaF: 0.39156
170         Relative Norm deltaU: 0.39156
171     NewtonRaphson::solveCurrentStep() -the ConvergenceTest object failed ↵
        in test()
172
173     Static Analysis: [1      /1      ] The Algorithm failed at load factor 0.4
174 > Analysis End ↵
        -----
175
176     Starting sequential static multistep analysis
177     =====
178     Creating analysis ↵
        model.....
179     Checking constraint ↵
        handler.....P
180     Checking ↵
        numberer.....
181     Checking analysis ↵
        algorithm.....
182     Checking system of equation ↵
        handler.....Pass!
183     Checking static integration ↵
        handler.....Pass!
184
185     Static Analysis: [1      /1      ]
186     [iteration 1      /5      ] Convergence Test: Absolute Norm ↵

```



```

    Displacement Increment::(tol: 0.001)
187         Absolute Norm deltaF: 9310.6
188         Absolute Norm deltaU: 2.6478e-19
189         Average Norm deltaF: 4655.3
190         Average Norm deltaU: 1.3239e-19
191         Relative Norm deltaF: 0.42857
192         Relative Norm deltaU: 2.6478e-19
193
194 > Analysis End ↵
    -----
195
196 Starting sequential static multistep analysis
197 =====
198 Creating analysis ↵
    model.....
199 Checking constraint ↵
    handler..... P
200 Checking ↵
    numberer.....
201 Checking analysis ↵
    algorithm.....
202 Checking system of equation ↵
    handler..... Pass!
203 Checking static integration ↵
    handler..... Pass!
204
205 Static Analysis: [1 /1 ]
206 [iteration 1 /5 ] Convergence Test: Absolute Norm ↵
    Displacement Increment::(tol: 0.001)
207         Absolute Norm deltaF: 1437.5
208         Absolute Norm deltaU: 1.0012e-18
209         Average Norm deltaF: 718.76
210         Average Norm deltaU: 5.006e-19
211         Relative Norm deltaF: 0.10425
212         Relative Norm deltaU: 1.0012e-18
213
214 > Analysis End ↵
    -----
215
216 Starting sequential static multistep analysis
217 =====
218 Creating analysis ↵
    model.....
219 Checking constraint ↵
    handler..... P
220 Checking ↵
    numberer.....
221 Checking analysis ↵
    algorithm.....
222 Checking system of equation ↵
    handler..... Pass!
223 Checking static integration ↵

```

```

224     handler..... Pass !
225 Static Analysis: [1      /1      ]
226 [iteration 1      /5      ] Convergence Test: Absolute Norm ↵
      Displacement Increment::(tol: 0.001)
227         Absolute Norm deltaF: 112.08
228         Absolute Norm deltaU: 5.1789e-19
229         Average Norm deltaF: 56.038
230         Average Norm deltaU: 2.5895e-19
231         Relative Norm deltaF: 0.017652
232         Relative Norm deltaU: 5.1789e-19
233
234 > Analysis End ↵
      -----

235
236 Starting sequential static multistep analysis
237 =====
238 Creating analysis ↵
      model.....
239 Checking constraint ↵
      handler..... P
240 Checking ↵
      numberer.....
241 Checking analysis ↵
      algorithm.....
242 Checking system of equation ↵
      handler..... Pass !
243 Checking static integration ↵
      handler..... Pass !
244
245 Static Analysis: [1      /1      ]
246 [iteration 1      /5      ] Convergence Test: Absolute Norm ↵
      Displacement Increment::(tol: 0.001)
247         Absolute Norm deltaF: 7.142
248         Absolute Norm deltaU: 8.6792e-19
249         Average Norm deltaF: 3.571
250         Average Norm deltaU: 4.3396e-19
251         Relative Norm deltaF: 0.001399
252         Relative Norm deltaU: 8.6792e-19
253
254 > Analysis End ↵
      -----

255
256 Starting sequential static multistep analysis
257 =====
258 Creating analysis ↵
      model.....
259 Checking constraint ↵
      handler..... P
260 Checking ↵
      numberer.....
261 Checking analysis ↵

```

```

algorithm.....
262 Checking system of equation ↵
    handler..... Pass!
263 Checking static integration ↵
    handler..... Pass!
264
265 Static Analysis: [1      /1      ]
266 [iteration 1      /5      ] Convergence Test: Absolute Norm ↵
    Displacement Increment::(tol: 0.001)
267             Absolute Norm deltaF: 0.44693
268             Absolute Norm deltaU: 1.0155e-18
269             Average Norm deltaF: 0.22347
270             Average Norm deltaU: 5.0774e-19
271             Relative Norm deltaF: 8.9267e-05
272             Relative Norm deltaU: 1.0155e-18
273
274 > Analysis End ↵
    -----

275
276 Starting sequential static multistep analysis
277 =====
278 Creating analysis ↵
    model.....
279 Checking constraint ↵
    handler..... P
280 Checking ↵
    numberer.....
281 Checking analysis ↵
    algorithm.....
282 Checking system of equation ↵
    handler..... Pass!
283 Checking static integration ↵
    handler..... Pass!
284
285 Static Analysis: [1      /1      ]
286 [iteration 1      /5      ] Convergence Test: Absolute Norm ↵
    Displacement Increment::(tol: 0.001)
287             Absolute Norm deltaF: 0.027935
288             Absolute Norm deltaU: 2.1658e-19
289             Average Norm deltaF: 0.013968
290             Average Norm deltaU: 1.0829e-19
291             Relative Norm deltaF: 5.5866e-06
292             Relative Norm deltaU: 2.1658e-19
293
294 > Analysis End ↵
    -----

295 How polite! Bye, have a nice day!

```

Simulation: New Elastic Loading Case

For design applications, linear elastic analysis cases are performed and later combined, using factors of safety (see section 1.3.5 on page 317) to obtain sectional forces for design.

The command for elastic analysis is:

```
1 new elastic loading case <string> ;
```

One example is

```
1 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.

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

```
1 combine elastic load cases
2   hdf5_filenames_list = <string>
3   load_factors_list = <string>
4   output_filename = <string>
5   ;
```

One example is

```
1 combine elastic load cases
2   hdf5_filenames_list = "test_case1.h5.feiooutput ↵
3   test_case2.h5.feiooutput"
4   load_factors_list = "1.2 1.5"
5   output_filename = "combine.h5.feiooutput"
6   ;
```

- `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`.

Simulation, Dynamic Solution Advancement for Solid-Fluid Interaction

Dynamic analysis of solid-fluid interaction can be performed using:

```
1 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:

```
1 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.

Simulation, Dynamic Solution Advancement for Stochastic Finite Element Method

Dynamic analysis of stochastic finite element modeling can be performed using:

```
1 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 1.3.5 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:

```
1 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.

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:

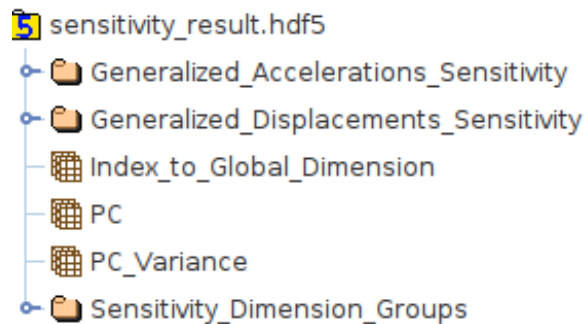


Figure 1.9: Overall data structure of output hdf5 file for sensitivity analysis.

Figure 1.9 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 1.10.

- **Component_Sobol_Indexes** dataset:

Is a column vector containing the computed Sobol Indexes for each component of polynomial chaos (PC) bases.

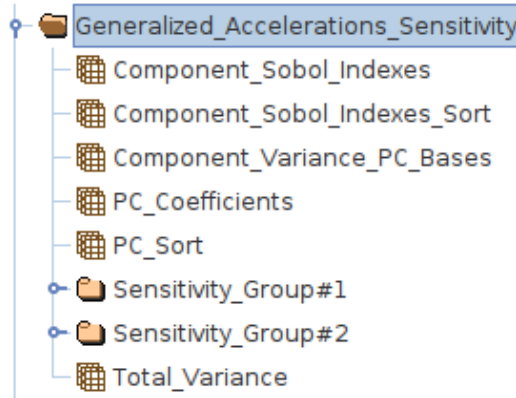


Figure 1.10: Datasets and data groups in **Generalized_Accelerations_Sensitivity** data group.

– **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 1.11.

It includes the following datasets:

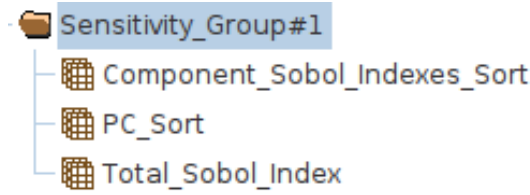


Figure 1.11: Datasets in **Sensitivity_Group#1**.

* **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. 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 1.12. Each dataset contains a column vector of integers, specifying the dimension IDs in specific sensitivity dimension group.

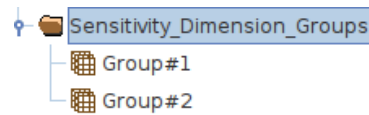


Figure 1.12: Datasets in **Sensitivity_Dimension_Groups** data group.

Simulation: 3D 3C Wave Field Inversion

```

1 generate domain reduction method motion file from ↵
    3D_3C_wave_field_inversion
2     target_nodes_filename = <string>
3     target_motions_filename = <string>
4     time_step = <T>
5     hdf5_file = <string>;

```

Example:

```

1 generate domain reduction method motion file from ↵
    3D_3C_wave_field_inversion
2     target_nodes_filename = "Target_Nodes.txt"
3     target_motions_filename = "Target_Motions.txt"
4     time_step = 0.01*s
5     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.

```

1     5
2     15
3     25
4     35
5     46
6     48
7     50
8     52
9

```

- `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.

```

1     Displacement_1000steps_with_CT.txt
2     Displacement_1000steps_with_CT.txt
3     Displacement_1000steps_with_CT.txt
4     Displacement_1000steps_with_CT.txt
5     Zero_Displacement_1000steps_with_CT.txt
6     Zero_Displacement_1000steps_with_CT.txt
7     Zero_Displacement_1000steps_with_CT.txt
8     Zero_Displacement_1000steps_with_CT.txt

```

9

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.

1.3.6 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](#).

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

```
1 disable all output;
```

Command to enable output is

```
1 enable all output;
```

Output Options: Enable/Disable Element Output

This option is used to enable or disable the outputting of element results from all elements to HDF5 (.feiooutput) 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;
```


Output Options: Enable/Disable Displacement Output

This option is used to enable or disable the displacement output at nodes to HDF5 (.feiooutput) file.

Note:- By default displacement output is enabled.

Command to disable displacement output is

```
1  disable displacement output;
```

Command to enable displacement output is

```
1  enable displacement output;
```

Output Options: Enable/Disable Acceleration Output

This option is used to enable or disable the acceleration output at nodes to HDF5 (.feiooutput) file.

Note:- By default acceleration output is disabled.

Command to disable acceleration output is

```
1 disable acceleration output;
```

Command to enable acceleration output is

```
1 enable acceleration output;
```

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;
```

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

```
1 output every <.> steps;
```

For example: To output only at interval of two time steps for a simulation of 100 steps. One can write

```
1 output every 2 steps;
```

This will only output for steps 2,4,6,... until 100th step.

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;
```

1.4 Checking the Model

Real-ESSI provides model check capability:

```
1 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 and 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.

1.5 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.

```

1  simulate constitutive testing BARDETMETHOD use material # <.>
2  scale_factor = <.>
3  series_file = <string>
4  sigma0 = ( <F/L^2> , <F/L^2> , <F/L^2> , <F/L^2> , <F/L^2> , ↵
<F/L^2> )
5  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: $(\sigma_{11}, \sigma_{22}, \sigma_{33}, \sigma_{12}, \sigma_{13}, \sigma_{23})$.
- `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.

```

1  simulate constitutive testing DIRECT_STRAIN use material # <.>
2  scale_factor = <.>
3  series_file = <string>
4  sigma0 = ( <F/L^2> , <F/L^2> , <F/L^2> , <F/L^2> , <F/L^2> , ↵
   <F/L^2> )
5  verbose_output = <.>
```

Where all 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 $d\epsilon_{11}$ $d\epsilon_{22}$ $d\epsilon_{33}$ $d\epsilon_{12}$ $d\epsilon_{13}$ $d\epsilon_{23}$.

1.6 List of Available Commands (tentative, not up to date)

```

1 add acceleration field # <.> ax = <accel> ay = <accel> az = <aaccel> ;
2 add constraint equal_dof with master node # <.> and slave node # <.> ←
  dof to constrain <.>;
3 add constraint equal_dof with node # <.> dof <.> master and node # ←
  <.> dof <.> slave;
4 add damping # <.> to element # <.>;
5 add damping # <.> to node # <.>;
6 add damping # <.> type Caughey3rd with a0 = <1/time> a1 = <time> a2 = ←
  <time^3> stiffness_to_use = ←
  <Initial_Stiffness|Current_Stiffness|Last_Committed_Stiffness>;
7 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>;
8 add damping # <.> type Rayleigh with a0 = <1/time> a1 = <time> ←
  stiffness_to_use = ←
  <Initial_Stiffness|Current_Stiffness|Last_Committed_Stiffness>;
9 add domain reduction method loading # <.> hdf5_file = <string> ←
  scale_factor = <.>;
10 add domain reduction method loading # <.> hdf5_file = <string>;
11 add element # <.> type 20NodeBrick using <.> Gauss points each ←
  direction with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, ←
  <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>) use ←
  material # <.>;
12 add element # <.> type 20NodeBrick with nodes (<.>, <.>, <.>, <.>, ←
  <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, ←
  <.>, <.>, <.>) use material # <.>;
13 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>;
14 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>;
15 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>;
16 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>;
17 add element # <.> type 27NodeBrick using <.> Gauss points each <.>
    direction with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>,
    <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>,
    <.>, <.>, <.>, <.>) use material # <.>;
18 add element # <.> type 27NodeBrick with nodes (<.>, <.>, <.>, <.>, <.>,
    <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>,
    <.>, <.>, <.>, <.>) use material # <.>;
19 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>;
20 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>;
21 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>;
22 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>;
23 add element # <.> type 3NodeShell_ANDES with nodes (<.>, <.>, <.>) <.>
    use material # <.> thickness = <l> ;
24 add element # <.> type 4NodeShell_ANDES with nodes (<.>, <.>, <.>, <.>)
    use material # <.> thickness = <l> ;
25 add element # <.> type 4NodeShell_MITC4 with nodes (<.>, <.>, <.>, <.>)
    use material # <.> thickness = <L>;
26 add element # <.> type 4NodeShell_NewMITC4 with nodes (<.>, <.>, <.>, <.>)
    use material # <.> thickness = <L>;
27 add element # <.> type 8_27_NodeBrick using <.> Gauss points each <.>
    direction with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>,
    <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>,
    <.>, <.>, <.>, <.>)

```

[illegible]

```

    <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>;
39 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>);
40 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>);
41 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>);
42 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>);
43 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>);
44 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 = (<.>, <.>, <.>);
45 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 = (<.>, <.>, <.>);
46 add element # <.> type ShearBeam with nodes (<.>, <.>) cross_section = <l^2> use material # <.>;
47 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 = (<.>, <.>, <.>);

```

```

    <.> );
48 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 = (<.>, <.>, ←
    <.> );
49 add element # <.> type truss with nodes (<.>, <.>) use material # <.> ←
    cross_section = <length^2> mass_density = <M/L^3> ;
50 add element # <.> type variable_node_brick_8_to_27 using <.> Gauss ←
    points each direction with nodes (<.>, <.>, <.>, <.>, <.>, <.>, ←
    <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, ←
    <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>) use material # <.>;
51 add elements (<.>) to physical_element_group "string";
52 add fiber # <.> using material # <.> to section # <.> ←
    fiber_cross_section = <area> fiber_location = (<L>,<L>);
53 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";
54 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";
55 add load # <.> to all elements type self_weight use acceleration ←
    field # <.>;
56 add load # <.> to element # <.> type self_weight use acceleration ←
    field # <.>;
57 add load # <.> to element # <.> type surface at nodes (<.> , <.> , ←
    <.> , <.>) with magnitude <.>;
58 add load # <.> to element # <.> type surface at nodes (<.> , <.> , ←
    <.> , <.>) with magnitudes (<.> , <.> , <.> , <.>);
59 add load # <.> to element # <.> type surface at nodes (<.> , <.> , ←
    <.> , <.>, <.>, <.>, <.>, <.>) with magnitude <.>;
60 add load # <.> to element # <.> type surface at nodes (<.> , <.> , ←
    <.> , <.>, <.>, <.>, <.>, <.>) with magnitudes (<.> , <.> , <.> , ←
    <.>, <.>, <.>, <.>, <.>);
61 add load # <.> to element # <.> type surface at nodes (<.> , <.> , ←
    <.> , <.>, <.>, <.>, <.>, <.>, <.>) with magnitude <.>;
62 add load # <.> to element # <.> type surface at nodes (<.> , <.> , ←
    <.> , <.>, <.>, <.>, <.>, <.>, <.>) with magnitudes (<.> , <.> , ←
    <.> , <.>, <.>, <.>, <.>, <.>, <.>);
63 add load # <.> to node # <.> type from_reactions;

```

```

64 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
65 add load # <.> to node # <.> type path_series FORCETYPE = <force or ←
    moment> time_step = <time> series_file = "filename";
66 add load # <.> to node # <.> type path_time_series FORCETYPE = <force ←
    or moment> series_file = "filename";
67 add load # <.> to node # <.> type self_weight use acceleration field ←
    # <.>;
68 add mass to node # <.> mx = <mass> my = <mass> mz = <mass> Imx = ←
    <mass*length^2> Imy = <mass*length^2> Imz = <mass*length^2>;
69 add mass to node # <.> mx = <mass> my = <mass> mz = <mass>;
70 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";
92 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> ;
93 add section # <.> type Elastic_Membrane_Plate elastic_modulus = <←
    <F/L^2> poisson_ratio = <.> thickness = <length> mass_density = <←
    <M/L^3>;
94 add section # <.> type FiberSection TorsionConstant_GJ = <F*L^2>
95 add section # <.> type Membrane_Plate_Fiber thickness = <length> use <←
    material # <.>;
96 add single point constraint to node # <.> dof to constrain <dof_type> <←
    constraint value of <corresponding unit>;
97 add uniform acceleration # <.> to all nodes dof <.> time_step = <T> <←
    scale_factor = <.> initial_velocity = <L/S> acceleration_file = <←
    <string>;
98 check mesh filename;
99 compute reaction forces;
100 define algorithm With_no_convergence_check / Newton / Modified_Newton;
101 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>;

```



```

102 define dynamic integrator Hilber_Hughes_Taylor with alpha = <.>;
103 define dynamic integrator Newmark with gamma = <.> beta = <.>;
104 define load factor increment <.>;
105 define NDMaterial constitutive integration algorithm Forward_Euler;
106 define NDMaterial constitutive integration algorithm ↵
    Forward_Euler_Subincrement number_of_subincrements =<.>;
107 define NDMaterial constitutive integration algorithm ↵
    Forward_Euler|Forward_Euler_Subincrement|Backward_Euler|Backward_Euler_Subin
    yield_function_relative_tolerance = <.> stress_relative_tolerance ↵
    = <.> maximum_iterations = <.>;
108 define physical_element_group "string";
109 define physical_node_group "string";
110 define solver ProfileSPD / UMFPack;
111 define static integrator displacement_control using node # <.> dof ↵
    DOFTYPE increment <length>;
112 disable asynchronous output;
113 disable element output;
114 disable output;
115 enable asynchronous output;
116 enable element output;
117 enable output;
118 fix node # <.> dofs <.>;
119 fix node # <.> dofs all;
120 free node # <.> dofs <.>;
121 help;
122 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> ) 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> ) 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 = <.>;
154 simulate constitutive testing undrained triaxial use material # <.> ←
    strain_increment_size = <.> maximum_strain = <.> ←
    number_of_times_reaching_maximum_strain = <.>;
155 simulate using eigen algorithm number_of_modes = <.>;
156 ux uy uz Ux Uy Uz rx ry rz;
157 while (.) { };
158 whos;

```

1.7 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)

```
1 a0
2 a1
3 a2
4 a3
5 a4
6 acceleration
7 acceleration_depth
8 acceleration_file
9 acceleration_filename
10 acceleration_scale_unit
11 add
12 algorithm
13 algorithm
14 all
15 all
16 allowed_subincrement_strain
17 alpha
18 alpha1
19 alpha2
20 and
21 angle
22 armstrong_frederick_cr
23 armstrong_frederick_ha
24 asynchronous
25 at
26 ax
27 axial_penalty_stiffness
28 axial_stiffness
29 axial_viscous_damping
30 ay
31 az
32 bending_Iy
33 bending_Iz
34 beta
35 Beta
36 beta_min
37 case
38 cases
39 characteristic_strength
40 check
41 chi
42 cohesion
43 combine
44 compression
45 compressive_strength
```

```
46 compressive_yield_strength
47 compute
48 confinement
49 confinement_strain
50 constitutive
51 constrain
52 constraint
53 contact_plane_vector
54 control
55 convergence
56 cR1
57 cR2
58 cross_section
59 crushing_strength
60 Current_Stiffness
61 cyclic
62 damage_parameter_An
63 damage_parameter_Ap
64 damage_parameter_Bn
65 damping
66 define
67 depth
68 dilatancy_angle
69 dilation_angle_eta
70 dilation_scale
71 direction
72 disable
73 displacement
74 displacement_file
75 displacement_scale_unit
76 dof
77 dofs
78 dofs
79 domain
80 druckerprager_k
81 DuncanChang_K
82 DuncanChang_n
83 DuncanChang_nu
84 DuncanChang_pa
85 DuncanChang_sigma3_max
86 DYNAMIC_DOMAIN_PARTITION
87 e0
88 each
89 elastic
90 elastic_modulus
91 elastic_modulus_horizontal
92 elastic_modulus_vertical
93 element
94 elements
95 else
96 enable
97 every
```

```
98 factor
99 fiber
100 fiber_cross_section
101 fiber_location
102 field
103 file
104 fix
105 fluid
106 free
107 friction_angle
108 friction_ratio
109 from
110 gamma
111 Gamma
112 Gauss
113 generate
114 GoverGmax
115 h_in
116 hardening_parameters_of_yield_surfaces
117 hardening_parameters_scale_unit
118 hdf5_file
119 hdf5_filenames_list
120 help
121 if
122 imposed
123 Imx
124 Imy
125 Imz
126 in
127 inclined
128 increment
129 initial_axial_stiffness
130 initial_confining_stress
131 initial_elastic_modulus
132 initial_shear_modulus
133 initial_shear_stiffness
134 initial_velocity
135 integration
136 integration_points
137 IntegrationRule
138 integrator
139 interface
140 isotropic_hardening_rate
141 joint_1_offset
142 joint_2_offset
143 K_f
144 K_s
145 k_x
146 k_y
147 k_z
148 kappa
149 kd_in
```

```
150 kinematic_hardening_rate
151 lambda
152 level
153 line_search_beta
154 line_search_eta
155 line_search_max_iter
156 liquefaction_Alpha
157 liquefaction_c_h0
158 liquefaction_Dir
159 liquefaction_dre1
160 liquefaction_Dre2
161 liquefaction_EXP_N
162 liquefaction_gamar
163 liquefaction_mdc
164 liquefaction_mfc
165 liquefaction_pa
166 liquefaction_pmin
167 load
168 load_factors_list
169 loading
170 local_y_vector
171 local_z_vector
172 M
173 M_in
174 magnitude
175 magnitudes
176 mass
177 mass_density
178 master
179 material
180 max_axial_stiffness
181 maximum_iterations
182 maximum_number_of_iterations
183 maximum_strain
184 maximum_stress
185 maximum_time_step
186 method
187 minimal
188 minimum_time_step
189 model
190 model
191 moment_x_stiffness
192 moment_y_stiffness
193 monotonic
194 motion
195 mu
196 mx
197 my
198 mz
199 name
200 NDMaterial
201 new
```

```
202 newton_with_subincrement
203 node
204 nodes
205 number_of_cycles
206 number_of_files
207 number_of_increment
208 number_of_integration_points
209 number_of_iterations
210 number_of_layers
211 number_of_modes
212 number_of_subincrements
213 number_of_times_reaching_maximum_strain
214 of
215 output
216 output
217 output_filename
218 p0
219 parallel
220 peak_friction_coefficient_limit
221 peak_friction_coefficient_rate_of_decrease
222 penalty_stiffness
223 pi1
224 pi2
225 pi3
226 plastic_deformation_rate
227 plastic_flow_kd
228 plastic_flow_xi
229 plot
230 point
231 points
232 poisson_ratio
233 poisson_ratio_h_h
234 poisson_ratio_h_v
235 porosity
236 print
237 propagation
238 pure
239 R0
240 radiuses_of_yield_surface
241 radiuses_scale_unit
242 rate_of_softening
243 reduction
244 reference_pressure
245 remove
246 rempve
247 residual_friction_coefficient
248 restart
249 restart_files
250 results
251 rho_a
252 rho_f
253 rho_s
```

```
254 rho_w
255 RMC_e
256 RMC_eta0
257 RMC_Heta
258 RMC_m
259 RMC_pc
260 RMC_qa
261 RMC_shape_k
262 rounded_distance
263 runTest
264 sanisand2004_A0
265 sanisand2004_c
266 sanisand2004_ch
267 sanisand2004_cz
268 sanisand2004_ec_ref
269 sanisand2004_G0
270 sanisand2004_h0
271 sanisand2004_lambda_c
272 sanisand2004_m
273 sanisand2004_Mc
274 sanisand2004_nb
275 sanisand2004_nd
276 sanisand2004_p_cut
277 sanisand2004_Pat
278 sanisand2004_xi
279 sanisand2004_z_max
280 sanisand2008_A0
281 sanisand2008_alpha_cc
282 sanisand2008_c
283 sanisand2008_ch
284 sanisand2008_cz
285 sanisand2008_ec_ref
286 sanisand2008_G0
287 sanisand2008_h0
288 sanisand2008_K0
289 sanisand2008_k_c
290 sanisand2008_lambda
291 sanisand2008_m
292 sanisand2008_nb
293 sanisand2008_nd
294 sanisand2008_p0
295 sanisand2008_p_in
296 sanisand2008_p_r
297 sanisand2008_Pat
298 sanisand2008_rho_c
299 sanisand2008_theta_c
300 sanisand2008_X
301 sanisand2008_xi
302 sanisand2008_z_max
303 save
304 scale_factor
305 SCOTCHGRAPHPARTITIONER
```



```
306 section
307 section_number
308 sequential
309 series_file
310 set
311 shear
312 shear_length_ratio
313 shear_modulus
314 shear_modulus_h_v
315 shear_stiffness
316 shear_viscous_damping
317 shear_zone_thickness
318 ShearStrainGamma
319 sigma0
320 simulate
321 single
322 size_of_peak_plateau
323 sizes_of_yield_surfaces
324 slave
325 slave
326 soil
327 soil_profile_filename
328 soil_surface
329 solid
330 solver
331 stage
332 steps
333 steps
334 stiffening_rate
335 stiffness_to_use
336 strain
337 strain_at_compressive_strength
338 strain_at_crushing_strength
339 strain_hardening_ratio
340 strain_increment_size
341 stress
342 stress_increment_size
343 stress_relative_tolerance
344 sub-stepping
345 surface
346 surface_vector_relative_tolerance
347 tensile_strength
348 tensile_yield_strength
349 tension_softening_stiffness
350 test
351 test
352 testing
353 thickness
354 time_step
355 to
356 tolerance_1
357 tolerance_2
```

```

358 torsion_Jx
359 torsional_stiffness
360 TorsionConstant_GJ
361 total_number_of_shear_modulus
362 total_number_of_yield_surface
363 triaxial
364 type
365 uniaxial
366 uniaxial_material
367 uniform
368 unit_of_acceleration
369 unit_of_damping
370 unit_of_rho
371 unit_of_vs
372 use
373 using
374 value
375 velocity_file
376 velocity_scale_unit
377 verbose_output
378 viscoelastic_modulus
379 von_mises_radius
380 wave
381 wave1c
382 wave3c
383 while
384 whos
385 with
386 xi_in
387 xz_plane_vector
388 yield_function_relative_tolerance
389 yield_strength
390 yield_surface_scale_unit
391 x
392 y
393 z

```

Second Order (inside commands)

```

1 20NodeBrick
2 20NodeBrick_up
3 20NodeBrick_upU
4 27NodeBrick
5 27NodeBrick_up
6 27NodeBrick_upU
7 3NodeShell_ANDES
8 4NodeShell_ANDES
9 4NodeShell_MITC4
10 4NodeShell_NewMITC4
11 8_27_NodeBrick
12 8_27_NodeBrick_up
13 8_27_NodeBrick_upU

```

```
14 8NodeBrick
15 8NodeBrick_fluid_incompressible_up
16 8NodeBrick_up
17 8NodeBrick_upU
18 Absolute_Norm_Displacement_Increment
19 Absolute_Norm_Unbalanced_Force
20 arclength_control
21 Average_Norm_Displacement_Increment
22 Average_Norm_Unbalanced_Force
23 Backward_Euler
24 BARDETMETHOD
25 beam_9dof_elastic
26 beam_displacement_based
27 beam_elastic
28 beam_elastic_lumped_mass
29 BeamColumnDispFiber3d
30 BearingElastomericPlasticity3d
31 BFGS
32 BondedContact
33 CamClay
34 Caughey3rd
35 Caughey4th
36 constant mean pressure triaxial strain control
37 Cosserat8NodeBrick
38 Cosserat_linear_elastic_isotropic_3d
39 Cosserat_von_Mises
40 DIRECT_STRAIN
41 displacement_control
42 DOFTYPE
43 domain reduction method
44 drained triaxial strain control
45 DruckerPrager
46 DruckerPragerArmstrongFrederickLE
47 DruckerPragerArmstrongFrederickNE
48 DruckerPragerMultipleYieldSurface
49 DruckerPragerMultipleYieldSurfaceGoverGmax
50 DruckerPragerNonAssociateArmstrongFrederick
51 DruckerPragerNonAssociateLinearHardening
52 DruckerPragervonMises
53 dynamic
54 eigen
55 elastic3d
56 Elastic_Membrane_Plate
57 ElasticFourNodeQuad
58 Energy_Increment
59 equal_dof
60 F_fluid_x
61 F_fluid_y
62 F_fluid_z
63 FiberSection
64 ForceBasedCoupledHardContact
65 ForceBasedCoupledSoftContact
```

```
66 ForceBasedElasticContact
67 ForceBasedHardContact
68 ForceBasedSoftContact
69 FORCETYPE
70 Forward_Euler
71 Forward_Euler_Subincrement
72 from_reactions
73 Fx
74 Fy
75 Fz
76 Hilber_Hughes_Taylor
77 HyperbolicDruckerPragerArmstrongFrederick
78 HyperbolicDruckerPragerLinearHardening
79 HyperbolicDruckerPragerNonAssociateArmstrongFrederick
80 HyperbolicDruckerPragerNonAssociateLinearHardening
81 linear
82 linear_elastic_crossanisotropic
83 linear_elastic_isotropic_3d
84 linear_elastic_isotropic_3d_LT
85 Membrane_Plate_Fiber
86 Modified_Newton
87 Mx
88 My
89 Mz
90 Newmark
91 Newton
92 non_converged_iterations
93 NonlinearFourNodeQuad
94 Norm_Displacement_Increment
95 Norm_Unbalance
96 Parallel
97 path_series
98 path_time_series
99 petsc
100 petsc_options_string
101 physical_element_group
102 physical_node_group
103 Pisano
104 PlaneStressLayeredMaterial
105 PlaneStressRebarMaterial
106 PlasticDamageConcretePlaneStress
107 pressure
108 ProfileSPD
109 Rayleigh
110 reaction_forces
111 reactions
112 Relative_Energy_Increment
113 Relative_Norm_Displacement_Increment
114 Relative_Norm_Unbalance
115 Relative_Norm_Unbalanced_Force
116 roundedMohrCoulomb
117 RoundedMohrCoulombMultipleYieldSurface
```

```
118 sanisand2004
119 sanisand2004_legacy
120 sanisand2008
121 self_weight
122 ShearBeam
123 solid fluid interaction transient
124 static
125 StressBasedCoupledHardContact_ElPP1Shear
126 StressBasedCoupledHardContact_NonLinHardShear
127 StressBasedCoupledHardContact_NonLinHardSoftShear
128 StressBasedCoupledSoftContact
129 StressBasedCoupledSoftContact_ElPP1Shear
130 StressBasedCoupledSoftContact_NonLinHardShear
131 StressBasedCoupledSoftContact_NonLinHardSoftShear
132 StressBasedHardContact_ElPP1Shear
133 StressBasedHardContact_NonLinHardShear
134 StressBasedHardContact_NonLinHardSoftShear
135 StressBasedSoftContact_ElPP1Shear
136 StressBasedSoftContact_NonLinHardShear
137 StressBasedSoftContact_NonLinHardSoftShear
138 SuperElementLinearElasticImport
139 support
140 surface
141 transient
142 truss
143 TsinghuaLiquefactionModelCirclePiPlane
144 TsinghuaLiquefactionModelNonCirclePiPlane
145 UMFPack
146 undrained simple shear
147 undrained triaxial
148 undrained triaxial stress control
149 uniaxial_concrete02
150 uniaxial_elastic
151 uniaxial_steel01
152 uniaxial_steel02
153 variable transient
154 variable_node_brick_8_to_27
155 vonMises
156 vonMisesArmstrongFrederick
157 vonMisesMultipleYieldSurface
158 vonMisesMultipleYieldSurfaceGoverGmax
159 With_no_convergence_check
160
161 beta
162 gamma
163 delta
164
165 ux
166 uy
167 uz
168 rx
169 ry
```

```
170 rz
171 Ux
172 Uy
173 Uz
174 p
175 M
176 m
177 kg
178 s
179 cm
180 mm
181 km
182 Hz
183 Minute
184 Hour
185 Day
186 Week
187 ms
188 ns
189 N
190 kN
191 Pa
192 kPa
193 MPa
194 GPa
195 pound
196 lbm
197 lbf
198 inch
199 in
200 feet
201 ft
202 yard
203 mile
204 psi
205 ksi
206 kip
207 g
208 pi
209
210 NUMBER_OF_NODES
211 NUMBER_OF_ELEMENTS
212 CURRENT_TIME
213 NUMBER_OF_SP_CONSTRAINTS
214 NUMBER_OF_MP_CONSTRAINTS
215 NUMBER_OF_LOADS
216 IS_PARALLEL
217 SIMULATE_EXIT_FLAG
218 then
219 while
220 do
221 let
```

```
222 vector
223
224 cos
225 sin
226 tan
227 cosh
228 sinh
229 tanh
230 acos
231 asin
232 atan
233 atan2
234 sqrt
235 exp
236 log10
237 ceil
238 fabs
239 floor
240 log
```

1.8 Integrated Development Environment (IDE) for DSL

1.9 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](#)).

1.10 Model Development and Mesh Generation using gmesh

1.11 Model Input File Editing using Sublime

<http://www.sublimetext.com/>

Bibliography

- M. A. Crisfield. A consistent co-rotational formulation for non-linear, three-dimensional, beam-elements. *Computer methods in applied mechanics and engineering*, 81(2):131–150, 1990.
- S. Dmitriev. Language oriented programming: The next programming paradigm. published online <http://www.onboard.jetbrains.com/is1/articles/04/10/lop/>, November 2004.
- A. C. Eringen. *Microcontinuum field theories: I. Foundations and solids*. Springer Science & Business Media, 2012.
- R. Faria, J. Oliver, and M. Cervera. A strain-based plastic viscous-damage model for massive concrete structures. *International Journal for Solids and Structures*, 35(14):1533–1558, 1998.
- B. Jeremić, Z. Yang, Z. Cheng, G. Jie, N. Tafazzoli, M. Preisig, P. Tasiopoulou, F. Pisanò, J. Abell, K. Watanabe, Y. Feng, S. K. Sinha, F. Behbehani, H. Yang, H. Wang, and K. D. Staszewska. *Non-linear Finite Elements: Modeling and Simulation of Earthquakes, Soils, Structures and their Interaction*. Self-Published-Online, University of California, Davis, CA, USA, 1989-2025. ISBN 978-0-692-19875-9. URL: <http://sokocalo.engr.ucdavis.edu/~jeremic/LectureNotes/>.
- S. C. Johnson. Yacc: Yet another compiler-compiler. Technical report, AT&T Bell Laboratories, Murray Hill, New Jersey 07974, 1975. URL <http://dinosaur.compilertools.net/>.
- D. E. Knuth. Literate programming. *Computer Journal (British Computer Society)*, 27(2):97–111, 1984.
- M. Lesk and E. Schmidt. Lex - a lexical analyzer generator. Technical Report UNIX TIME-SHARING SYSTEM:UNIX PROGRAMMER'S MANUAL, Seventh Edition, Volume 2B, AT&T Bell Laboratories, Murray Hill, New Jersey 07974, 1975. URL <http://dinosaur.compilertools.net/>.
- S. Mazzoni, F. McKenna, M. H. Scott, G. L. Fenves, and B. Jeremić. *Open System for Earthquake Engineers Simulation (OpenSees) : User Manual*. Pacific Earthquake Engineering Research Center, Richmond, December 2002.

- M. Mernik, J. Heering, and A. M. Sloane. When and how to develop domain-specific languages. *ACM Computing Surveys*, 37(4):316–344, December 2005. <http://doi.acm.org/10.1145/1118890.1118892>.
- E. Niebler. xpressive: Dual-mode dsel library design. In D. Musser, editor, *Library-Centric Software Design LCSD'05*. Object-Oriented Programming, Systems, Languages and Applications, October 2005.
- OpenCFD Ltd. *OpenFOAM User Guide*, v1906 edition, 2019.
- S. Sakamoto and R. Ghanem. Polynomial chaos decomposition for the simulation of non-gaussian nonstationary stochastic processes. *Journal of Engineering Mechanics*, 128(2):190–201, February 2002.
- B. Stroustrup. A rationale for semantically enhanced library languages. In D. Musser, editor, *Library-Centric Software Design LCSD'05*. Object-Oriented Programming, Systems, Languages and Applications, October 2005.
- M. P. Ward. Language oriented programming. Computer Science Department, Science Labs, South Rd, Durham, DH1 3LE, <http://www.cse.dmu.ac.uk/~mward/martin/papers/middle-out-t.pdf>, January 17 2003.
- D. Xiu. *Numerical Methods for Stochastic Computations*. Princeton University Press, 2010.
- O. C. Zienkiewicz and R. L. Taylor. *The Finite Element Method*, volume 2. McGraw - Hill Book Company, Fourth edition, 1991.