AMIE configuration guide

Alain B. Giorla

January 21, 2015

This document shows how to write initiation files for AMIE. These files can be used to set up variables in simulations without re-compiling the main executable. An example of initiation file usage in AMIE can be found in the `main_2d_composite` example.

Contents

1 First example

```
# first AMIE configuration file
.sample
..height = 0.1
..width = 0.1
..behaviour
...type = PASTE_BEHAVIOUR
.stepping
..time_step = 0.1
..number_of_time_steps = 1
.discretization
..sampling_number = 8
..order = LINEAR
.boundary_condition
..condition = FIX_ALONG_XI
..position = LEFT
.boundary_condition
..condition = FIX_ALONG_ETA
..position = BOTTOM
.boundary_condition
..condition = SET_STRESS_ETA
..position = TOP
..value = -1e6
.output
..file_name = output_file
..time_step
...at = ALL
..field = REAL_STRESS_FIELD
..field = STRAIN_FIELD
.export
..file_name = export_mesh_file
..time_step
...at = ALL
..field = REAL_STRESS_FIELD
..field = STRAIN_FIELD
```

Here is a step-by-step explanation of the content of the file.

1.1 Comment

```
# first AMIE configuration file
```

Everything located on the same line after a **#** character is considered a comment and will not be accounted for in a simulation. A comment can either start a new line as in the example shown, or be inserted at the end of an existing line.

1.2 Sample definition

```
.sample
..height = 0.1
..width = 0.1
..behaviour
...type = PASTE_BEHAVIOUR
```

Here, we define the box in which the simulation will be carried out. The box is a rectangle of the specified height and width (centered on the point (0,0) if not specified otherwise). The mechanical behaviour of the box is a predefined behaviour corresponding to cement paste.

1.3 Time-stepping parameters

```
.stepping
..time_step = 0.1
..number_of_time_steps = 1
```

Here, we define the number of time steps we want to perform as well as the duration (in days) of each time step.

1.4 Mesh generation parameters

```
.discretization
..sampling_number = 8
..order = LINEAR
```

Here, we define the parameters used to generate the mesh and the subsequent finite element discretization. The mesh will be generated with 8 nodes on each edge of the sample, and we use linear shape functions for the finite element approximation.

1.5 Boundary conditions

```
.boundary_condition
..condition = FIX_ALONG_XI
..position = LEFT
.boundary_condition
..condition = FIX_ALONG_ETA
..position = BOTTOM
.boundary_condition
..condition = SET_STRESS_ETA
..position = TOP
..value = -1e6
```

Here, we define three boundary conditions. The first condition sets to zero the horizontal displacement (axis `XI`) of the left edge of the sample. The second conditions sets to zero the vertical displacement (axis `ETA`) of the bottom edge of the sample. The third condition imposes a stress on the upper edge of the sample. The negative value of the load indicates a compression.

1.6 Output

```
.output
..file_name = output_file
..time_step
...at = ALL
..field = REAL_STRESS_FIELD
..field = STRAIN_FIELD
```

Here, we define that the average real stress and average strain field values will be extracted from the simulation and written in a text file at all time steps of the simulation. This will open a file named `output_file` and write the following table:

```
0.1 -5.30733 999998 -1015.52 -2.51262e-05 8.37528e-05 -1.09992e-07
```

Each line corresponds to a time step of the simulation. The first column corresponds to the current instant, the three following columns are the three components of the average stress in the sample (ordered as σ_{xx}, σ_{yy} and σ_{xy}), and the three last columns are the three components of the average strain in the sample (ordered as ϵ_{xx}, ϵ_{yy} and ϵ_{xy}).

Note that if `output_file` already exists, its content will be overwritten.

1.7 Export

```
.export
..file_name = export_mesh_file
..time_step
...at = ALL
..field = REAL_STRESS_FIELD
..field = STRAIN_FIELD
```

Here, we choose to write mesh files containing the deformed state of the sample, as well as the real stress and the strain fields, and that such files will be written at each time step of the simulation. For the present simulation, three files will be generated:

```
export_mesh_file_header
export_mesh_file_1.svg
export_mesh_file_1
```

The header file contains simply a list of all non-SVG files generated. That is:

```
export_mesh_file_1
```

The SVG file contains eight snapshots of the mesh ordered vertically: two for the displacements components, three for the stress components, and three for the strain components.

The last file contains raw information for use of AMIE in-house `viewer`.

As for the output files, if any of these files already exist, their content will be overwritten.

1.8 Important remarks on file naming

The following remarks apply to any file name called in the configuration file (either as an input or output file):

- Do not use white spaces in the file path. Any white space is ignored during parsing.
- For Windows users, use `/` as a folder separator instead of `\`. AMIE does the conversion automatically.

2 Managing inclusions

2.1 First example

```
# example AMIE configuration file with inclusions
.sample
..height = 0.1
..width = 0.1
..behaviour
...type = PASTE_BEHAVIOUR
.stepping
..time_step = 0.1
..number_of_time_steps = 1
.discretization
..sampling_number = 8
..order = LINEAR
.inclusions
..particle_size_distribution
...type = CONSTANT
...rmax = 0.005
...number = 100
..geometry
...type = CIRCLE
..placement
...spacing = 0.001
...tries = 1000
..behaviour
...type = AGGREGATE_BEHAVIOUR
.boundary_condition
..condition = FIX_ALONG_XI
..position = LEFT
.boundary_condition
..condition = FIX_ALONG_ETA
..position = BOTTOM
.boundary_condition
..condition = SET_STRESS_ETA
..position = TOP
..value = -1e6
.output
..file_name = output_file
..time_step
```

```
...at = ALL
..field = REAL_STRESS_FIELD
..field = STRAIN_FIELD
.export
..file_name = export_mesh_file
..time_step
...at = ALL
..field = REAL_STRESS_FIELD
..field = STRAIN_FIELD
```

This file is essentially identical to the first, except that the following lines have been added:

```
.inclusions
..particle_size_distribution
...type = CONSTANT
...rmax = 0.008
...number = 100
...fraction = 0.5
..geometry
...type = CIRCLE
..placement
...spacing = 0.001
...tries = 1000
..behaviour
...type = AGGREGATE_BEHAVIOUR
```

This defines a set of circular inclusions of a radius of 8 mm each. Inclusions will be generated until either 100 of them have been created, or until they cover at least 50 % of the surface of the sample. They will be randomly placed in the sample with at least 1 mm between each particle. The placement algorithm will try 1000 iterations; any particle not placed in the sample after these 1000 iterations will be left out and ignored. The particles are attributed a predefined mechanical behaviour which corresponds to aggregates used in concrete.

AMIE comes with several analytic particle size distributions. Instead of `CONSTANT`, one of the following curves can be used:

```
...type = BOLOME_A # upper Bolome curve (contains more fine)
...type = BOLOME_B # middle Bolome curve
...type = BOLOME_C # lower Bolome curve (contains more coarse)
...type = BOLOME_D # Bolome curve adapted for mortar
```

2.2 Defining a custom particle size distribution

Custom particle size distribution can be defined in external files, and then imported in AMIE, as in the following example:

```
.inclusions
..particle_size_distribution
...type = FROM_CUMULATIVE_FILE
...file_name = ../examples/data/composite/psd_example.txt
...rmax = 0.008
```

```
...number = 100
...fraction = 0.5
..geometry
...type = CIRCLE
..placement
...spacing = 0.001
...tries = 1000
..behaviour
...type = AGGREGATE_BEHAVIOUR
```

The file `../examples/data/composite/psd_example.txt` contains a two-columns table, the second column being a set of decreasing radii, and the first column the percentage of aggregates with a radius smaller than the current radius. The particle size distribution is then linearly interpolated between these points.

```
100.0   0.008
70.0    0.004
30.0    0.0025
10.0    0.00125
1.0     0.00075
0       0.0005
```

2.3 Importing pre-placed inclusions

If you already have a file containing a list of inclusions with their center and radius, you can easily import the file using the following syntax:

```
.inclusions
..particle_size_distribution
...type = FROM_INCLUSION_FILE
...file_name = ../examples/data/composite/inclusion_example.txt
...number = 5
...column = center_x
...column = center_y
...column = radius
..geometry
...type = CIRCLE
..behaviour
...type = AGGREGATE_BEHAVIOUR
```

Where `../examples/data/composite/inclusion_example.txt` contains a table with three columns, the first being the `x` coordinate of the center of each inclusion, the second the `y` coordinate, and the third the radius. The order in which the columns are defined is relevant!

```
0        0     0.008
0.04315 -0.017 0.007
0.025    0.027 0.006
-0.015  -0.001 0.005
-0.015   0.044 0.004
```

Note that since the file already contains the position of the inclusions, we don't need to place them. Therefore the parent `inclusions` object does not contain a `placement` object in this specific case.

2.4 Using ellipses

Ellipses can be generated instead of circles using the `geometry` object:

```
.inclusions
..particle_size_distribution
...type = CONSTANT
...rmax = 0.008
...number = 100
...fraction = 0.5
..geometry
...type = ELLIPSE
...aspect_ratio = 0.7
...orientation = 3.141592
..placement
...spacing = 0.001
...tries = 1000
..behaviour
...type = AGGREGATE_BEHAVIOUR
```

This will generate ellipses with an aspect ratio of 0.7, and the same area as a 8mm radius circle. For each ellipse, the angle of the major axis with the horizontal axis `XI` will be randomly determined to a value between 0 and `orientation`. Therefore, if `orientation = 0`, all ellipses will be aligned with the horizontal axis.

Alternatively, ellipses can be imported from a file using the following syntax:

```
.inclusions
..particle_size_distribution
...type = FROM_INCLUSION_FILE
...file_name = ../examples/data/composite/ellipsoidal_inclusion_example.txt
...number = 5
...column = center_x
...column = center_y
...column = radius_a
...column = radius_b
...column = axis_x
...column = axis_y
..geometry
...type = ELLIPSE
..behaviour
...type = AGGREGATE_BEHAVIOUR
```

Where `radius_a` and `radius_b` are the values of the major and minor radii of the ellipses, `axis_x` and `axis_y` are coordinates of the direction of the major axis of the ellipse. Therefore, `axis_x = 1` and `axis_y = 0` define an ellipse aligned with the horizontal axis, while `axis_x = 0` and `axis_y = 1` define an ellipse aligned with the vertical axis.

2.5 Controlling the placement of inclusions

By default, the inclusions will be placed inside the sample, and so that they do not intersect the edges of the sample. However, it is possible to select a larger (or smaller) box in which to generate the inclusions. For example, this would be used to simulate microstructure from cored concrete. The following example places the inclusions in a slightly larger box:

```
.inclusions
..particle_size_distribution
...type = CONSTANT
...rmax = 0.008
...number = 100
...fraction = 0.5
..geometry
...type = CIRCLE
..placement
...spacing = 0.001
...tries = 1000
...box
....width = 0.11
....height = 0.11
..behaviour
...type = AGGREGATE_BEHAVIOUR
```

The relative position of the placement box with the sample can also be adjusted by defining its center. The following example will move the placing box to the right side of the sample, leaving the left side empty.

```
...box
....width = 1
....height = 1
....center
.....x = 0.5
.....y = 0
```

Note that by default, the sample is centered on (0,0), but this can be adjusted with a similar syntax.

When you place the particles in a different box than the sample, the target fraction of inclusions generated corresponds to a fraction of the placement box, and not on the sample itself. Variations can occur depending on the characteristics of your particle size distribution and the relative size of the sample and the placement box.

2.6 Creating several families of inclusions

You can create within the same sample different particles, with different behaviours and different particle sizes distributions. For example, the following code will create two families of inclusions: one set of large inclusions corresponding to aggregates, and a second set of inclusions, with a smaller radius, and a void behaviour (which could for example simulate pores).

```
.inclusions
..particle_size_distribution
...type = CONSTANT
...rmax = 0.008
```

```
...number = 100
...fraction = 0.4
..geometry
...type = CIRCLE
..placement
...spacing = 0.001
...tries = 1000
..behaviour
...type = AGGREGATE_BEHAVIOUR
.inclusions
..particle_size_distribution
...type = CONSTANT
...rmax = 0.001
...number = 1000
...fraction = 0.4
..geometry
...type = CIRCLE
..placement
...spacing = 0.0001
...tries = 10000
..behaviour
...type = VOID_BEHAVIOUR
```

The inclusions from the second family are placed after the inclusions in the first family, in the remaining space of the sample. The inclusions of the two families will not overlap nor intersect. Therefore, swapping the position of the second and the first family would yield very different microstructures.

You can define with this method as many families of inclusions as necessary.

2.7 Creating inclusions in inclusions

You can create families of inclusions within an existing family of inclusion with the following syntax:

```
.inclusions
..particle_size_distribution
...type = CONSTANT
...rmax = 0.008
...number = 100
...fraction = 0.4
..geometry
...type = CIRCLE
..placement
...spacing = 0.001
...tries = 1000
..behaviour
...type = AGGREGATE_BEHAVIOUR
..inclusions
...particle_size_distribution
....type = CONSTANT
....rmax = 0.005
....number = 1000
```

```
....fraction = 0.4
...geometry
....type = CIRCLE
...placement
....spacing = 0.0001
....tries = 10000
...behaviour
....type = VOID_BEHAVIOUR
```

This is very similar to the previous example, except that here the pores will be placed inside the aggregates (and not outside).

You can have several families of inclusions at the same level, and as many embedded families of inclusions as you wish. Structures of the following type are acceptable:

```
.inclusions
# family A placed in the sample
..inclusions
# family B within family A
...inclusions
# family C within family B
...inclusions
# family D within family B
..inclusions
# family E within family A
.inclusions
# family F placed in the sample
```

2.8 Creating concentric inclusions

You can create concentric inclusions inside an existing family of inclusions with the following syntax:

```
.inclusions
# family A
..particle_size_distribution
...type = CONSTANT
...rmax = 0.008
...number = 100
...fraction = 0.4
..geometry
...type = CIRCLE
..placement
...spacing = 0.001
...tries = 1000
..behaviour
...type = AGGREGATE_BEHAVIOUR
..inclusions
# family B within family A
...particle_size_distribution
....type = FROM_PARENT_DISTRIBUTION
....layer_thickness = 0.001
```

```
...behaviour
....type = VOID_BEHAVIOUR
```

This will generate one inclusion inside all the inclusions of the family A, with a difference in radius equal to `layer_thickness`. Inclusions of the family A with a radius lower than `layer_thickness` will not spawn an inner inclusion.

This feature can also make use of AMIE functions using `layer_thickness_function` (the `x` coordinate will be set as the radius of the parent distribution). The definition below is therefore equivalent to the definition above:

```
.inclusions
# family A
..particle_size_distribution
...type = CONSTANT
...rmax = 0.008
...number = 100
...fraction = 0.4
..geometry
...type = CIRCLE
..placement
...spacing = 0.001
...tries = 1000
..behaviour
...type = AGGREGATE_BEHAVIOUR
..inclusions
# family B within family A
...particle_size_distribution
....type = FROM_PARENT_DISTRIBUTION
....layer_thickness_function = "x 0.005 -"
...behaviour
....type = VOID_BEHAVIOUR
```

2.9 Placing a single inclusion

It is possible to place a single inclusions at a very specific position in the microstructure, using the `UNIQUE` keyword. Valid geometries are `CIRCLE`, `ELLIPSE` and `RECTANGLE`. For example, circular inclusions can be created with:

```
.inclusions
..particle_size_distribution
...type = UNIQUE
..geometry
...type = CIRCLE
...radius = 0.1
...center
....x = 0
....y = 0
```

Ellipses require a `major_radius` and `minor_radius`, as well as a `major_axis` direction (a vector of norm 1):

```
.inclusions
..particle_size_distribution
```

```
...type = UNIQUE
..geometry
...type = ELLIPSE
...major_radius = 0.1
...minor_radius = 0.05
...center
....x = 0
....y = 0
...major_axis
....x = 1
....y = 0
```

Rectangles are defined as a `sample` object. They cannot be oriented.

```
.inclusions
..particle_size_distribution
...type = UNIQUE
..geometry
...type = RECTANGLE
...center
....x = 0
....y = 0
...height = 0.1
...width = 0.05
```

2.10 Combining the different techniques

All of the previous features can be combined one with another to generate arbitrarily complex microstructures. However, be mindful of the following:

- Inclusions imported from a pre-generated file (with `FROM_INCLUSION_FILE`) will be placed even if they overlap or intersect with existing inclusions. However, inclusions placed after the imported family will check overlap and intersection with the pre-generated family.
- Concentric inclusions are not meant to be used with ellipses.
- When placing inclusions in a box different than the sample, the `fraction` parameter corresponds to a fraction of the placement box, not the sample.

3 Adapting the mesh parameters

The mesh is mostly controlled by the `sampling_number` in the `discretization` object, yet further refinement can be achieved.

3.1 Avoid the meshing of the smallest inclusions

Sometimes, meshing the smallest inclusions can yield meshes with elongated elements. This can be avoided by not meshing these inclusions, using the `sampling_restiction` parameter. This example will not mesh inclusions with less than 4 points along their boundary. Acceptable values are 4, 8 and 16.

```
.discretization
..sampling_number = 8
```

```
..order = LINEAR
..sampling_restriction = SAMPLE_RESTRICT_4
```

3.2 Changing the mesh density in specific inclusions

It is possible to modify the mesh density in whole families of inclusions by adding a `sampling_factor` in the definition of the `inclusions`. The following example multiplies by two the mesh density in all inclusions defined here.

```
.inclusions
# family A of inclusions ..particle_size_distribution
...type = CONSTANT
...rmax = 0.008
...number = 100
...fraction = 0.5
..geometry
...type = CIRCLE
..placement
...spacing = 0.001
...tries = 1000
..behaviour
...type = AGGREGATE_BEHAVIOUR
..sampling_factor = 2
```

`sampling_factor` is defined only for the current `inclusions` family. Inclusions from other families still use the non-modified mesh density, including the families of inclusions embedded in the current family.

It is possible to modify the mesh density only for the inclusions which intersects the edges of the sample by using `intersection_sampling_factor`. The following example will multiply by 3 the mesh density in the inclusions intersecting the edges of the samples, and by 2 the mesh density of the other inclusions.

```
.inclusions
# family A of inclusions ..particle_size_distribution
...type = CONSTANT
...rmax = 0.008
...number = 100
...fraction = 0.5
..geometry
...type = CIRCLE
..placement
...spacing = 0.001
...tries = 1000
..behaviour
...type = AGGREGATE_BEHAVIOUR
..sampling_factor = 2
..intersection_sampling_factor = 3
```

3.3 Changing the mesh density in the matrix

The mesh density can be changed inside the matrix (that is, outised of any inclusion) by adding a `sampling_factor` to the main `sample` of the simulation:

```
.sample
..height = 0.1
..width = 0.1
..behaviour
...type = PASTE_BEHAVIOUR
..sampling_factor = 2
```

4 Defining a time step

The easiest way to manage the time stepping is to use a constant time step (or a logarithmic-constant time step). However, two other methods are also available.

4.1 Importing a list of time steps from a file

It is possible to read a text file containing a list of instants at which the simulation will be carried out using the following syntax:

```
.stepping
..list_of_time_steps = ../examples/data/composite/time_steps.txt # name of the file containing the data
```

The series of instants must be strictly increasing; no time step must be equal to 0 (or lower). The first instant must be equal to 0.

The file name must not contain a comma (`,`), otherwise the second method is used.

4.2 Defining a list of time steps

Instead of calling a file, it is possible to directly define the list of instants in the input file:

```
.stepping
..list_of_time_steps = 0,1,3 # list of instants separated by commas
```

The series of instants must be strictly increasing; no time step must be equal to 0 (or lower). The first instant must be equal to 0. The instants must be separated by commas (`,`).

4.3 Using a function of time

It is possible to define each time step as a function of the current time and the previous time step:

```
.stepping
..time_step = 0.1
..number_of_time_steps = 10
..next_time_step = "t 0.1 * x +"
```

When declaring such function, use `x` as the previous time step, and `t` as previous instant. The example above constructs the following series of $(t_n, \Delta t_n)$:

$$\Delta t_0 = 0.1 \quad (1)$$

$$t_0 = 0 \quad (2)$$

$$\Delta t_{n+1} = \Delta t_n + 0.1\ t_n \quad (3)$$

$$t_{n+1} = t_n + \Delta t_{n+1} \quad (4)$$

5 Defining boundary conditions

Boundary conditions can either be applied along an edge of the sample (`LEFT`, `RIGHT`, `BOTTOM` and `TOP`) or on a corner of the sample (`BOTTOM_LEFT`, `BOTTOM_RIGHT`, `TOP_LEFT` and `TOP_RIGHT`).

5.1 Boundary conditions in displacements

A displacement can be applied using the following syntax:

```
.boundary_condition
..condition = SET_ALONG_ETA
..position = TOP
..value = 0.001
```

Note that the condition `FIX_ALONG_XI` (respectively `ETA`) is equivalent to `SET_ALONG_XI` (respectively `ETA`) associated with the `value` 0.

5.2 Boundary conditions in stress

A stress can be applied using the following syntax:

```
.boundary_condition
..condition = SET_STRESS_ETA
..position = TOP
..value = 1e6
```

Stress boundary conditions should not be used on corners, but should be defined along an edge of a sample only.

5.3 Boundary conditions in force

A force can be applied using the following syntax:

```
.boundary_condition
..condition = SET_FORCE_ETA
..position = TOP
..value = 1e6
```

Force boundary conditions should be used on single points (like corners) rather than on surfaces.

5.4 Multiple boundary conditions at the same location

In general, boundary conditions in stress/force and in displacements should not be mixed on the same location (either edge or corner). Also, boundary conditions in stress/force stack, while boundary conditions in displacement on the same axis (`XI` or `ETA`) do not: only the latest defined is applied.

5.5 Time-dependent boundary conditions

There are three ways to define a time-dependent boundary condition. The easiest way is to define a rate. The following example will apply a loading rate of 0.1 MPa per day:

```
.boundary_condition
..condition = SET_STRESS_ETA
..position = TOP
..time_evolution
...rate = 1e5
```

One can also define a function of time `t`. The previous example can be rewritten as follow:

```
.boundary_condition
..condition = SET_STRESS_ETA
..position = TOP
..time_evolution
...function = "1e5 t *"
```

Also, it is possible to import a file containing the values of the boundary condition at different instants:

```
.boundary_condition
..condition = SET_STRESS_ETA
..position = TOP
..time_evolution
...file_name = ../example/data/composite/time_dependent_load.txt
```

The file must contain two columns, the first being the time, the second the values of the boundary condition. The instants need not be equal to the time steps defined for the simulation. However, be aware that the value of the boundary condition is linearly interpolated between the instants defined in `../example/data/composite/time_dependent_load.txt`. The first of the following examples defines a loading ramp, while the second defines a loading step:

```
0       0
1       1e6
2       1e6
```

```
0       0
0.99999 0
1       1e6
2       1e6
```

Instead of a file, it is also possible to directly input the values of the boundary condition in the initiation file:

```
.boundary_condition
..condition = SET_STRESS_ETA
..position = TOP
..time_evolution
...instants = 0,1,2 # list of instants separated by commas
...values = 0,1e6,1e6 # list of values separated by commas
```

5.6 Applying a boundary condition on a section of an edge

It is possible to limit the area on which the boundary condition is applied by using the `restriction` keyword. The following example applies a stress on the `TOP` of the sample, only on the points located in

$x \in [0.4, 0.5]$, $y \in [0.4, 0.5]$:

```
.boundary_condition
..condition = SET_STRESS_ETA
..position = TOP
..restriction
...top_right
....x = 0.5
....y = 0.5
...bottom_left
....x = 0.4
....y = 0.4
```

The `position` and the box defined by `bottom_left/top_right` must intersect (or overlap), otherwise the boundary condition will not be applied.

5.7 Applying a boundary condition on a specific point

It is possible to apply the boundary condition at a specific point. The point at which the condition will be applied is the closest point on the edge `position` from the specified `point`.

```
.boundary_condition
..condition = SET_FORCE_ETA
..position = TOP
..point
...x = 0.5
...y = 0.5
```

This will not work with stress boundary conditions.

6 Extracting the average stress and strain over certain families of inclusions

For each family of inclusion it is possible to extract in the `output` the average stress, strain (or other values) for these inclusions only. Tpo do so, the following syntax is used:

```
   .output
..file_name = output_file
..time_step
...at = ALL
..field = REAL_STRESS_FIELD
..field = STRAIN_FIELD
..inclusions
...index = 0
...field = REAL_STRESS_FIELD
...field = STRAIN_FIELD
..inclusions
...index = 1
...field = REAL_STRESS_FIELD
...field = STRAIN_FIELD
```

Any number of `inclusions` objects can be defined (up to the actual number of inclusion families + 1). The index 0 denotes the matrix. Higher indexes indicate the corresponding family of inclusions, ranked in the order in which they are generated (that is, they are ranked in the order in which they appear in the script file).

7 Defining behaviours

7.1 Using a pre-defined behaviour

A set of predefined behaviours exist in AMIE to represent common materials found in concrete. These can be used by choosing the appropriate type. The existing materials are listed below:

```
..type = VOID_BEHAVIOUR
..type = PASTE_BEHAVIOUR
..type = AGGREGATE_BEHAVIOUR
..type = ASR_GEL_BEHAVIOUR
..type = CONCRETE_BEHAVIOUR
..type = STEEL_BEHAVIOUR
..type = REBAR_BEHAVIOUR
```

For each of these, it is possible to ignore the default values for some (or all) of the parameters (`young_modulus`, `poisson_ratio`, etc) by declaring it again. The list of parameters for each behaviour is listed below with their default values:

```
.behaviour
..type = VOID_BEHAVIOUR

.behaviour
..type = PASTE_BEHAVIOUR
..young_modulus = 12e9
..poisson_ratio = 0.3
..short_term_creep_modulus = 0.3 # defined as a fraction of the elastic modulus
..long_term_creep_modulus = 0.37 # defined as a fraction of the elastic modulus
..tensile_strain_limit = 0.0003
..damage = TRUE
```

The creep properties of the cement paste are ignored if the simulation is not in space-time finite elements (see Using space-time finite elements). The boolean `damage` can be set to `FALSE` to run elastic (or visco-elastic) simulations.

```
.behaviour
..type = AGGREGATE_BEHAVIOUR
..young_modulus = 59e9
..poisson_ratio = 0.3
..tensile_strain_limit = 0.0012
..damage = TRUE

.behaviour
..type = ASR_GEL_BEHAVIOUR
..young_modulus = 22e9
..poisson_ratio = 0.18
```

```
..imposed_deformation = 0.22

.behaviour
..type = CONCRETE_BEHAVIOUR
..young_modulus = 37e9
..poisson_ratio = 0.3
..compressive_strength = -37e6

.behaviour
..type = STEEL_BEHAVIOUR
..young_modulus = 210e9
..poisson_ratio = 0.3
..tensile_strength = 276e6

.behaviour
..type = REBAR_BEHAVIOUR
..young_modulus = 59e9
..poisson_ratio = 0.3
..tensile_strength = 57000
```

7.2 Defining an elastic behaviour

A purely elastic behaviour can be defined as in the following example:

```
.behaviour
..type = ELASTICITY
..young_modulus = 10e9
..poisson_ratio = 0.3
```

Alternatively, you can define the material from the bulk and shear moduli. The material defined below is the same as the one defined above:

```
.behaviour
..type = ELASTICITY
..bulk_modulus = 8.3333e9
..poisson_ratio = 3.8461e9
```

However, you cannot mix both definition, and you must define the two selected parameters.
You can add an imposed deformation with the following syntax:

```
.behaviour
..type = ELASTICITY_AND_IMPOSED_DEFORMATION
..young_modulus = 10e9
..poisson_ratio = 0.3
..imposed_deformation = 0.0001
```

7.3 Defining an elastic-and-damage behaviour

To use a damaging behaviour, two additional parameters must be set: the first one is a `fracture_criterion` which defines the failure surface of the material, possibly including its post-peak behaviour. The second one is a `damage_model` which defines how damage is resolved by AMIE. This can be achieved as in the following

example:

```
.behaviour
..type = ELASTICITY_AND_FRACTURE
..young_modulus = 10e9
..poisson_ratio = 0.3
..fracture_criterion
...type = VON_MISES
...limit_tensile_stress = 1e6
..damage_model
...type = ISOTROPIC_LINEAR_DAMAGE
```

7.4 Defining a fracture criterion

Several types of fracture criteria are available in AMIE. The two mostly used are a Mohr-Coulomb and a Von Mises criteria. The Mohr-Coulomb criterion can furthermore be completed with different descending branches. Example definitions of these criteria are listed below:

```
..fracture_criterion
...type = MOHR_COULOMB
...limit_tensile_strain = 0.001
...limit_compressive_strain = -0.01

..fracture_criterion
...type = LINEAR_SOFTENING_MOHR_COULOMB
...limit_tensile_strain = 0.001 # defines when the damage process starts
...maximum_tensile_strain = 0.002 # defines when the damage process ends
...limit_compressive_strain = -0.01
...maximum_compressive_strain = -0.02

..fracture_criterion
...type = EXPONENTIAL_SOFTENING_MOHR_COULOMB
...limit_tensile_strain = 0.001 # defines when the damage process starts
...maximum_tensile_strain = 0.002 # defines when the damage process ends
...limit_compressive_strain = -0.01
...maximum_compressive_strain = -0.02

..fracture_criterion
...type = VON_MISES
...limit_tensile_stress = 1e6
```

In all cases, the damage follows a non-local definition. The radius of the damage band can be controlled with the parameter `material_characteristic_radius` as in the following example:

```
..fracture_criterion
...type = VON_MISES
...limit_tensile_stress = 1e6
...material_characteristic_radius = 0.001
```

7.5 Using the Modified Compression Field Theory

The Modified Compressive Field Theory is a specific fracture criterion for reinforced concrete. For this criterion, you need to define the list of rebars present in the material, with the location and diameter of each:

```
..fracture_criterion
...type = MCFT
...limit_compressive_strain = -0.001
...rebar
....location = 0.001
....diameter = 0.0001
...rebar
....location = 0.005
....diameter = 0.0001
```

As many rebars as required may be defined (including zero).

7.6 Defining a damage model

Two types of damage models can be used. `ISOTROPIC_LINEAR_DAMAGE` is the default model, and searches for the exact series in which the elements are damaged, and the corresponding damage increment. That is, at each iteration of the damage algorithm a bisection is carried out to find the exact damage increment to apply.

`ISOTROPIC_INCREMENTAL_\textit{LINEAR_}DAMAGE` is a similar algorithm, except that it will apply a fixed amount of damage at each iteration. The amount can be fixed using the `damage_increment` variable:

```
..damage_model
...type = ISOTROPIC_INCREMENTAL_LINEAR_DAMAGE
...damage_increment = 0.01
```

For both damage models, it is possible to define a threshold of damage above which the material will be considered as entirely broken. In the following example, any element which reaches a damage value of 0.6 or higher will be considered as broken and removed from the simulation.

```
..damage_model
...type = ISOTROPIC_INCREMENTAL_LINEAR_DAMAGE
...damage_increment = 0.01
...maximum_damage = 0.6
```

8 Using space-time finite elements

Space-time finite elements are used for visco-elastic analysis of materials def. Several concepts must be defined first before going into the details of the input parameters.

8.1 Additional viscoelastic variables

Space-time finite elements use more unknown than standard finite elements. Indeed, for each dashpot placed in series in the model a new set of viscoelastic displacements is required. For example, the Maxwell model requires one set of additional variables, while a Burger's model requires two.

8.2 Indexed axis

For each set of viscoelastic variables two unknowns (in two dimensions, three for three-dimensional simulations) are necessary: the viscoelastic displacement along the `XI` axis and the viscoelastic displacement along the `ETA` axis.

To generalize to an arbitrary number of additional variables, the concept of indexed axis is introduced. One axis is defined for each unknown, and each axis is attributed a positive integer (its index). The two first axes (index 0 and 1) correspond respectively to the displacement along the `XI` and `ETA` axis. Then, each pair of axes corresponds to one pair of viscoelastic displacement. Even indexes correspond to displacements along the `XI` axis, while odd indexes correspond to displacements along the `ETA` index.

For example, the correlation between the index of the axes and the displacement of a Burger material is as follow:

Index	Definition
0	Total displacement along the `XI` axis
1	Total displacement along the `ETA` axis
2	Internal displacement of the Maxwell dashpot along the `XI` axis
3	Internal displacement of the Maxwell dashpot along the `ETA` axis
4	Internal displacement of the Kelvin-Voigt unit along the `XI` axis
5	Internal displacement of the Kelvin-Voigt unit along the `ETA` axis

8.3 Boundary conditions

For all boundary conditions, the `position` of the boundary condition must be complemented with `_AFTER`. `TOP` must thus be replaced with `TOP_AFTER`, `LEFT` with `LEFT_AFTER`, etc.

Boundary conditions in stress and in force can be used as previously defined.

Boundary conditions in displacement must use the `condition` `SET_ALONG_INDEXED_AXIS` complemented with the index of the `axis`. Therefore, to set the vertical displacement (which is the axis indexed with 1) equal to 0 on the bottom of the sample one must use:

```
.boundary_condition
..condition = SET_ALONG_INDEXED_AXIS
..position = BOTTOM_AFTER
..value = 0
..axis = 1
```

These two rules must be followed when defining displacement boundary conditions in space-time finite elements:

- If you set the horizontal (respectively vertical) displacement to 0, you must also set all displacements along the even (respectively odd) indexed axes to 0.
- If you set the horizontal (respectively vertical) displacement to a value different than 0, you must leave free the displacements along the other even (respectively odd) indexed axes (that is, no boundary condition related to these axes).

8.4 Behaviours and additional sets of variables

As stated above, each behaviour comes with its own set of additional viscoelastic variables. However, when mixing different behaviours, then you mus make sure that all behaviours have the same number of additional

viscoelastic variables. This can be achieved by setting the `additional_viscoelastic_variable` to the proper number when necessary.

For example, if the matrix is a Burger material (2 sets of viscoelastic variables) and the inclusions an elastic material (no additional set of viscoelastic variables), then the behaviour of the inclusions must be completed with the two missing sets of viscoelastic variables:

```
.sample
# sample description here...
..behaviour
...type = BURGER
# sample behaviour description here...
.inclusions
# inclusions description here...
..behaviour
...type = ELASTICITY
...additional_viscoelastic_variables = 2
# inclusions behaviour description here...
```

If the inclusions are a Maxwell material (one set of viscoelastic variables), then one set is missing and must be declared:

```
.sample
# sample description here...
..behaviour
...type = BURGER
# sample behaviour description here...
.inclusions
# inclusions description here...
..behaviour
...type = MAXWELL
...additional_viscoelastic_variables = 1
# inclusions behaviour description here...
```

8.5 Using the predefined material behaviours

The predefined `PASTE_BEHAVIOUR`, `AGGREGATE_BEHAVIOUR` and `ASR_GEL_BEHAVIOUR` have each two sets of viscoelastic variables (this cannot be changed, as they are meant to be used together).

The other predefined behaviours (except the `VOID_BEHAVIOUR`) are not meant to be used with space-time finite elements. The various `ELASTICITY` behaviour can be used in space-time finite elements, provided the number of `additional_viscoelastic_variables` is correctly set.

8.6 Defining a viscoelastic material

Different types of viscoelastic materials are available in AMIE. Their definition is summarized here.

`VISCOSITY`, `KELVIN_VOIGT` and `MAXWELL` are the three simplest viscoelastic behaviours. They require three parameters: a `young_modulus`, a `poisson_ratio`, and a `characteristic_time` (in day). The `young_modulus`/`poisson_ratio` pair can be replaced with the `bulk_modulus`/`shear_modulus` pair. They have no set of viscoelastic variables, except for the `MAXWELL` model which requires one. Example of a `KELVIN_VOIGT` material is given here:

```
..behaviour
...type = KELVIN_VOIGT
```

```
...young_modulus = 10e9
...poisson_ratio = 0.3
...characteristic_time = 10
```

The Burger material corresponds to a Maxwell unit placed in series with a Kelvin-Voigt unit. It requires two sets of viscoelastic variables, and is defined as follow:

```
..behaviour
...type = BURGER
...maxwell
....young_modulus = 50e9
....poisson_ratio = 0.3
....characteristic_time = 100
...kelvin_voigt
....young_modulus = 10e9
....poisson_ratio = 0.3
....characteristic_time = 10
```

Kelvin-Voigt and Maxwell chains with arbitrary numbers of units can be defined with the `GENERALIZED_KELVIN_VOIGT` and `GENERALIZED_MAXWELL` behaviours. These requires as many sets of viscoelastic variables as the number of `branch` in the chain. The `first_branch` variable defines the first elastic spring in the chain. An example of generalized Kelvin-Voigt chain with three branches can be found here:

```
..behaviour
...type = GENERALIZED_KELVIN_VOIGT
...first_branch
....young_modulus = 10e9
....poisson_ratio = 0.3
...branch
....young_modulus = 0.1e9
....poisson_ratio = 0.3
....characteristic_time = 10
...branch
....young_modulus = 5e9
....poisson_ratio = 0.3
....characteristic_time = 100
...branch
....young_modulus = 10e9
....poisson_ratio = 0.3
....characteristic_time = 1000
```

8.7 Fracture criterion in space-time finite elements

Space-time finite elements use a different set of fracture criterion. Only the four following criteria are available.

`MAXIMUM_TENSILE_STRAIN` defines a purely brittle material in which failure occurs when the strain reaches the specified `limit_tensile_strain`.

```
.fracture_criterion
..type = MAXIMUM_TENSILE_STRAIN
..limit_tensile_strain = 0.001
```

`LINEAR_SOFTENING_MAXIMUM_TENSILE_STRAIN` defines a material with a linear descending branch in the strain-stress diagram. `limit_tensile_strain` and `limit_tensile_stress` indicate the position of the peak, while `maximum_tensile_strain` indicates the ultimate strain at failure.

```
.fracture_criterion
..type = LINEAR_SOFTENING_MAXIMUM_TENSILE_STRAIN
..limit_tensile_strain = 0.001
..limit_tensile_stress = 1e6
..maximum_tensile_strain = 0.002
```

`MAXIMUM_TENSILE_STRESS` defines a pseudo-plastic material in which failure occurs when the stress reaches the specified `limit_tensile_stress`.

```
.fracture_criterion
..type = MAXIMUM_TENSILE_STRESS
..limit_tensile_stress = 1e6
```

`ELLIPSOIDAL_SOFTENING_MAXIMUM_TENSILE_STRESS` defines a material which shifts from a `MAXIMUM_TENSILE_STRAIN` material with a fast loading rate, towards a `MAXIMUM_TENSILE_STRESS` material with a slow loading rate. The `instantaneous_modulus` (which corresponds to the apparent modulus of the material with an infinitely fast loading rate) and the `relaxed_modulus` (apparent modulus with an infinitely slow loading rate) are required as well. The `instantaneous_modulus` must be strictly higher than the `relaxed_modulus`.

```
.fracture_criterion
..type = ELLIPSOIDAL_SOFTENING_MAXIMUM_TENSILE_STRESS
..limit_tensile_strain = 0.001
..limit_tensile_stress = 1e6
..instantaneous_modulus = 10e9
..relaxed_modulus = 1e9
```

As for any fracture criteria, these criteria can also have a `material_characteristic_radius` defined.

8.8 Damage model in space-time finite elements

Only the `ISOTROPIC_INCREMENTAL_LINEAR_DAMAGE` damage model is available in space-time finite elements. It has an additional parameter, `time_tolerance`, which indicates with which accuracy the instant at which damage occurs is captured. A small time tolerance indicates a high accuracy. The default value is set to `1e-9`.

```
..damage_model
...type = ISOTROPIC_INCREMENTAL_LINEAR_DAMAGE
...damage_increment = 0.01
...time_tolerance = 1e-9
```

9 The generalized logarithmic creep behaviour

This specific behaviour defines a visco-elastic material with imposed deformation. It uses space-time finite elements, so the guidelines of the previous section applies.

9.1 Basic behaviour

The behaviour consists in possibly four parts:

- An elastic behaviour.
- A non-linear visco-elastic behaviour which corresponds to a logarithmic creep curve.
- An imposed deformation (for example, thermal expansion, drying shrinkage, etc).
- A damage behaviour (defined as a pair of a fracture criterion and damage model).

All parts (except the elastic behaviour) are optional, which allows this behaviour to cover a wide range of materials.

9.2 Internal parameters

The behaviour relies on a certain number of internal parameters, which might represent for example physical fields (like temperature, relative humidity), material constants (modulus, thermal expansion coefficients, etc), or temporary mathematical variables for ease of calculation. These parameters might be changed dynamically during the simulation (see below).

The parameters are declared with their default values in a `parameters` object as follow:

```
.behaviour
..type = LOGARITHMIC_CREEP
..parameters
...young_modulus = 10e9
...poisson_ratio = 0.3
...temperature = 293
...thermal_expansion_coefficient = 10e-6
```

All `LOGARITHMIC_CREEP` materials must have the following parameters:

```
young_modulus
poisson_ratio
```

Or alternatively:

```
bulk_modulus
shear_modulus
```

In order to use the logarithmic creep law itself, the material must have the following parameters:

```
creep_modulus
creep_poisson
creep_characteristic_time
```

Or alternatively:

```
creep_bulk
creep_shear
creep_characteristic_time
```

In order to use an imposed deformation, the material ,must have the following parameter:

```
imposed_deformation
```

In order to use a damage and creep behaviour, one can either define explicitly a `fracture_criterion` and `damage_model` as follow (in which case AMIE will be unable to change the fracture properties as a function of the internal parameters), or use the method described below.

```
.behaviour
..type = LOGARITHMIC_CREEP
..parameters
# definition of the material parameters
..fracture_criterion
# definition of the fracture criterion
..damage_model
# definition of the damage model
```

9.3 Embedded fracture criterion for the logarithmic creep behaviour

There are two options available:

- Import strain-stress curves from a text file
- Declare the strain-stress curve
- Explicitly declare the maximum stress and strain in the input file

The two methods cannot be combined. If both are defined, then the import will be used, and the other parameters will not be accounted for.

9.3.1 Import strain-stress curves

```
.behaviour
..type = LOGARITHMIC_CREEP
..parameters
...tension_file_name = /path/to/the/stress/strain/curve/in/tension
...compression_file_name = /path/to/the/stress/strain/curve/in/compression
...material_characteristic_radius = # value
...strain_renormalization_factor = # optional, default value = 1e4
...stress_renormalization_factor = # optional, default value = 1e-6
...damage_increment = # value
...maximum_damage = # value
...time_tolerance = # optional, default value = 1e-9
```

The behaviour can be defined in tension, compression, or both. If `tension_file_name` (alternatively `compression_file_name`) is not defined, then the material will not fail in tension (or in compression). If none of them are defined, then AMIE will try to use the other method described below.

The text files must contain a table with the strain (first column) and stress (second column), in SI units. The file for tension must contain only positive (or zero) values, and the file for compression must contain only negative (or zero) values. Furthermore, it is preferable for the strain to be strictly increasing in the table.

For each element, AMIE will create the following variables, which can then be acted upon using material laws (see below):

```
tensile_strength
tensile_strain
tensile_ultimate_strength
tensile_ultimate_strain
compressive_strength
compressive_strain
compressive_ultimate_strength
compressive_ultimate_strain
```

9.3.2 Declare a stress-strain curve

```
.behaviour
..type = LOGARITHMIC_CREEP
..parameters
...strain_stress_curve
....tension
.....strain = 0.0001,0.0002 # values of the strain separated by commas (must be positive)
.....stress = 1e6,0 # values of the stress separated by commas (must be positive)
....compression
.....strain = -0.0001,-0.0002 # values of the strain separated by commas (must be negative)
.....stress = -1e6,0 # values of the stress separated by commas (must be negative)
...material_characteristic_radius = # value
...strain_renormalization_factor = # optional, default value = 1e4
...stress_renormalization_factor = # optional, default value = 1e-6
...damage_increment = # value
...maximum_damage = # value
...time_tolerance = # optional, default value = 1e-9
```

The behaviour can be defined in tension, compression, or both. If `tension` (alternatively `compression`) is not defined, then the material will not fail in tension (or in compression). If none of them are defined, then AMIE will try to use the other method described below.

For each element, AMIE will create the following variables, which can then be acted upon using material laws (see below):

```
tensile_strength
tensile_strain
tensile_ultimate_strength
tensile_ultimate_strain
compressive_strength
compressive_strain
compressive_ultimate_strength
compressive_ultimate_strain
```

9.3.3 Declare the maximum stress and strain

```
.behaviour
..type = LOGARITHMIC_CREEP
..parameters
```

```
...tensile_strength = # stress at the peak
...tensile_ultimate_strength = # stress at the end of the damage process
...tensile_ultimate_strain = # strain at the end of the damage process
...compressive_strength = # stress at the peak
...compressive_ultimate_strength = # stress at the end of the damage process
...compressive_ultimate_strain = # strain at the end of the damage process
...material_characteristic_radius = # value
...strain_renormalization_factor = # optional, default value = 1e4
...stress_renormalization_factor = # optional, default value = 1e-6
...damage_increment = # value
...maximum_damage = # value
...time_tolerance = # optional, default value = 1e-9
```

The behaviour can be defined in tension, in compression, or both. If no `tensile strength` (alternatively `compressive_strength`) is found, then the material will not fail in tension (or in compression). If none of them are defined (and no stress-strain curve files are defined), then the material will not fail.

The behaviour will be assumed linear between the peak and the ultimate value.

Tensile strength and strain must be positive, compressive strength and strain must be negative.

`tensile_strength` (alternatively `compressive_strength`) can be replaced with `tensile_strain` (or `compressive_strain`), the strain at the peak. The `tensile_strength` will then be derived from the `young_modulus` of the material.

The ultimate values are not required. If they are not defined, then the material will only fail when its stress reaches the prescribed strength.

`tensile_ultimate_strength` can be replaced with `tensile_strength_decrease_factor`, a scalar between 0 and 1. It is then calculated from the `tensile_strength` using:

$$\texttt{tensile\_ultimate\_strength} = \texttt{tensile\_strength} \times (1 - \texttt{tensile\_strength\_decrease\_factor}) \quad (5)$$

`tensile_ultimate_strain` can be replaced with `tensile_strain_increase_factor`, a scalar higher than 1. It is then calculated from the `tensile_strain` using:

$$\texttt{tensile\_ultimate\_strain} = \texttt{tensile\_strain} \times \texttt{tensile\_strain\_increase\_factor} \quad (6)$$

The same manipulation can be made for the `compressive_ultimate_strength` and `compressive_ultimate_strain`.

9.3.4 Common parameters

The `strain_renormalization_factor` and `stress_renormalization_factor` are introduced to increase the numerical precision of the damage algorithm. They should be chosen so that the quantities `tensile_strain`×`strain_renormalization_factor` and `tensile_strength`×`stress_renormalization_factor` are approximately on the same order of magnitude. The default values should be valid in most cases.

The `material_characteristic_radius` is the characteristic length for the non-local averaging of the strain and depends on the materialś underlying microstructure.

The `damage_increment` controls how much damage is applied at each step of the algorithm (the value should be between 0 and 1, and at least lower than 0.1), while the `time_tolerance` controls how many elements can be damaged at each step of the algorithm (the smaller `time_tolerance`, the less elements are damaged at each step). The default `time_tolerance` should be valid in most cases.

`maximum_damage` controls how much damage an element can sustain before failing completely.

Note that none of these parameters can be changed during the simulation.

9.4 Material laws

The internal parameters defined above are constants, unless appropriate material laws are defined to act upon them. These laws can be used to set variables to a certain value, or to perform specific operations on these variables. Any number of material law can be defined, using the following syntax:

```
.behaviour
..type = LOGARITHMIC_CREEP
..parameters
# definition of the material parameters
..material_law
# definition of first material law
..material_law
# definition of second material law
..material_law
# definition of third material law, etc
```

When doing a time step, AMIE will first apply all the material laws defined, change the mechanical properties of the element accordingly, and then compute the results at that time step.

The material laws will be applied in the order in which they are declared. Therefore, if the material uses a `THERMAL_EXPANSION` material law without any temperature defined first, no thermal deformation will be occur.

9.5 Predefined material laws

In this section, a few common material laws are defined for ease of use. Their definition include the list of internal `parameters` required for their proper use.

9.5.1 Arrhenius law

```
..material_law
...type = ARRHENIUS
...parameter_affected = $parameter
...reference_temperature = 293
```

Requires the following `parameters`:

```
temperature
$parameter
$parameter_activation_energy
```

Applies:

$$x = x\ e^{E_a[1/T - 1/T_0]} \tag{7}$$

With:

$$\begin{aligned} x &: \texttt{\$parameter} \\ E_a &: \texttt{\$parameter\_activation\_energy} \\ T &: \texttt{temperature} \\ T_0 &: \texttt{reference\_temperature} \end{aligned}$$

9.5.2 Arrhenius law for creep

```
..material_law
...type = CREEP_ARRHENIUS
...reference_temperature = 293
```

Requires the following parameters:

```
temperature
creep_characteristic_time
creep_activation_energy
```

Applies:

$$\eta = \eta\ e^{E_c[1/T - 1/T_0]} \tag{8}$$
$$\tag{9}$$

With:

$$\begin{aligned} \eta &: \texttt{creep\_characteristic\_time} \\ E_c &: \texttt{creep\_activation\_energy} \\ T &: \texttt{temperature} \\ T_0 &: \texttt{reference\_temperature} \end{aligned}$$

9.5.3 Drying shrinkage

```
..material_law
...type = DRYING_SHRINKAGE
...reference_relative_humidity = 1.
```

Requires the following parameters:

```
imposed_deformation
relative_humidity
drying_shrinkage_coefficient
```

Applies:

$$\alpha = \begin{cases} 0 & \text{if } h > h_0 \\ \alpha - k_s(h - h_0) & \text{otherwise} \end{cases} \tag{10}$$

With:

$$\begin{aligned} \alpha &: \texttt{imposed\_deformation} \\ k_s &: \texttt{drying\_shrinkage\_coefficient} \\ h &: \texttt{relative\_humidity} \\ h_0 &: \texttt{reference\_relative\_humidity} \end{aligned}$$

9.5.4 Non-linear creep

```
..material_law
...type = LOAD_NONLINEAR_CREEP
```

Requires the following `parameters`:

```
relative_humidity
creep_modulus
revoverable_modulus
tensile_strength
compressive_strength
```

Applies:

$$C_v = C_v \frac{1 - f^{10}}{1 + f^2} \tag{11}$$

$$C_r = C_r \frac{1 - f^{10}}{1 + f^2} \tag{12}$$

With:

$$\begin{aligned} C_v &: \texttt{creep\_modulus} \\ C_r &: \texttt{recoverable\_modulus} \\ f &: \text{stress/strength ratio} \end{aligned} \tag{13}$$

9.5.5 Radiation-induced volumetric expansion

```
..material_law
...type = RADIATION_INDUCED_VOLUMETRIC_EXPANSION
```

Requires the following `parameters`:

```
imposed_deformation
neutron_fluence
neutron_fluence_correction
```

```
radiation_expansion_delay
maximum_radiation_expansion
```

Applies:

$$\alpha = \alpha + \kappa\ \epsilon_{max} \frac{e^{\delta N} - 1}{\epsilon_{max} + \kappa\ e^{\delta N}} \tag{14}$$

With:

$$\begin{aligned} \alpha &: \texttt{imposed\_deformation} \\ \kappa &: \texttt{radiation\_expansion\_delay} \\ \epsilon_{max} &: \texttt{maximum\_radiation\_expansion} \\ \delta &: \texttt{neutron\_fluence\_correction} \\ N &: \texttt{neutron\_fluence} \end{aligned}$$

9.5.6 Relative humidity effect on creep

```
..material_law
...type = CREEP_HUMIDITY
```

Requires the following `parameters`:

```
relative_humidity
creep_modulus
revoverable_modulus
creep_humidity_coefficient
```

Applies:

$$C_v = C_v \left(k_h\ (1-h) + e^{k_h\ (1-h)} \right) \tag{15}$$

$$C_r = C_r \left(k_h\ (1-h) + e^{k_h\ (1-h)} \right) \tag{16}$$

With:

$$\begin{aligned} C_v &: \texttt{creep\_modulus} \\ C_r &: \texttt{recoverable\_modulus} \\ k_h &: \texttt{creep\_humidity\_coefficient} \\ h &: \texttt{relative\_humidity} \end{aligned}$$

9.5.7 Thermal expansion

```
..material_law
...type = THERMAL_EXPANSION
...reference_temperature = 293
```

Requires the following `parameters`:

```
imposed_deformation
temperature
thermal_expansion_coefficient
```

Applies:

$$\alpha = \alpha + k_T(T - T_0) \tag{17}$$

With:

$$\begin{aligned} \alpha &: \texttt{imposed\_deformation} \\ k_T &: \texttt{thermal\_expansion\_coefficient} \\ T &: \texttt{temperature} \\ T_0 &: \texttt{reference\_temperature} \end{aligned}$$

9.6 Generic material laws

In this section, generic material laws are described. These laws can be used to formulate various material behaviours.

9.6.1 Extract strain or stress field values

```
..material_law
...type = GET_FIELD
...field = STRAIN_FIELD
```

This law is a utility function to access the values of the common field (displacement, strain, stress, etc). The example above will generate three variables (in two dimensions) `strain_field_0`, `strain_field_1` and `strain_field_2` containing the three components of the strain field in the element.

9.6.2 As a function of a single variable

```
..material_law
...type = SIMPLE_DEPENDENT
...input_parameter = $input
...output_parameter = $output
...function = # declare function of x here
...additive = FALSE
```

Requires the following `parameters`:

```
$input
$output
```

Applies:

$$\mathcal{Y} = \begin{cases} f(\mathcal{X}) & \text{if } \texttt{additive} = \texttt{FALSE} \\ \mathcal{Y} + f(\mathcal{X}) & \text{otherwise} \end{cases} \tag{18}$$

With:

$$\begin{aligned} \mathcal{X} &: \texttt{\$input} \\ \mathcal{Y} &: \texttt{\$ouput} \end{aligned}$$

9.6.3 As a function of space and time

```
..material_law
...type = SPACE_TIME_DEPENDENT
...output_parameter = $output
...function = # declare function of x,y,z,t here
...additive = FALSE
```

Requires the following `parameters`:

```
$output
```

Applies:

$$\mathcal{Y} = \begin{cases} f(x,y,z,t) & \text{if } \texttt{additive} = \texttt{FALSE} \\ \mathcal{Y} + f(x,y,z,t) & \text{otherwise} \end{cases} \tag{19}$$

With:

$$\mathcal{Y} : \texttt{\$output}$$

9.6.4 As a function of multiple variables

```
..material_law
...type = VARIABLE_DEPENDENT
...output_parameter = $output
...x = $input_x
...y = $input_y
...z = $input_z
...t = $input_t
...u = $input_u
...v = $input_v
...w = $input_w
...function = # declare function of x,y,z,t,u,v,w here
...additive = FALSE
```

Requires the following `parameters`:

```
$output
$input_x
$input_y
```

```
$input_z
$input_t
$input_u
$input_v
$input_w
```

Applies:

$$\mathcal{Y} = \begin{cases} f(x,y,z,t,u,v,w) & \text{if } \texttt{additive} = \texttt{FALSE} \\ \mathcal{Y} + f(x,y,z,t,u,v,w) & \text{otherwise} \end{cases} \tag{20}$$

With:

$$\begin{aligned} \mathcal{Y} &: \texttt{\$output} \\ x &: \texttt{input\_x} \\ y &: \texttt{input\_y} \\ z &: \texttt{input\_z} \\ t &: \texttt{input\_t} \\ u &: \texttt{input\_u} \\ v &: \texttt{input\_v} \\ w &: \texttt{input\_w} \end{aligned}$$

It is not necessary to define all variables (x, z, y, t, u, v, w). Only those used in `f` are required. Note that if `f` is a function of x, y, z or t, and the respective `input_x`, `input_y`, `input_z` or `input_t` is not declared, then the function uses the actual space-time coordinates of the element.

9.6.5 As a linear interpolation of another variable

```
..material_law
...type = LINEAR_INTERPOLATED
...input_parameter = $input
...output_parameter = $output
...file_name = # path to the file containing the interpolation data
```

Or:

```
..material_law
...type = LINEAR_INTERPOLATED
...input_parameter = $input
...output_parameter = $output
...input_values = 0,1,2 # values of the input parameter separated by commas
...output_values = 0,1,2 # values of the output parameter separated by commas
```

Requires the following `parameters`:

```
$input
$output
```

Applies:

$$\mathcal{Y} = linear\_interpolation(\mathcal{X}) \tag{21}$$

With:

$$\begin{aligned} \mathcal{X} &: \texttt{\$input} \\ \mathcal{Y} &: \texttt{\$ouput} \end{aligned}$$

The interpolation file must contain two columns, the first being the values of `$input` and the second the corresponding values of `$param`.

9.6.6 Maximum

```
..material_law
...type = MAXIMUM
...output_parameter = $output
...input_parameter = $input_x
...input_parameter = $input_y
...input_parameter = $input_z
```

Requires the following `parameters`:

```
$output
$input_x
$input_y
$input_z
```

Applies:

$$\mathcal{Y} = \max(x, y, z, ...) \tag{22}$$

With:

$$\begin{aligned} \mathcal{Y} &: \texttt{\$output} \\ x &: \texttt{\$input\_x} \\ y &: \texttt{\$input\_y} \\ z &: \texttt{\$input\_z} \end{aligned}$$

This law can accept any number of `input_parameter` (at least one).

9.6.7 Minimum

```
..material_law
...type = MINIMUM
...output_parameter = $output
...input_parameter = $input_x
...input_parameter = $input_y
...input_parameter = $input_z
```

Requires the following `parameters`:

```
$output
$input_x
$input_y
$input_z
```

Applies:

$$\mathcal{Y} = \min(x, y, z, ...) \tag{23}$$

With:

$$\begin{aligned} \mathcal{Y} &: \texttt{\$output} \\ x &: \texttt{\$input\_x} \\ y &: \texttt{\$input\_y} \\ z &: \texttt{\$input\_z} \end{aligned}$$

This law can accept any number of `input_parameter` (at least one).

9.6.8 Time derivative

```
..material_law
...type = TIME_DERIVATIVE
...input_parameter = $input
```

Requires the following `parameters`:

```
$input
$input_rate
```

Applies:

$$\mathcal{Y} = \frac{\partial \mathcal{X}}{\partial t} \tag{24}$$

With:

$$\begin{aligned} \mathcal{X} &: \texttt{\$input} \\ \mathcal{Y} &: \texttt{\$input\_rate} \end{aligned}$$

9.6.9 Time integral

```
..material_law
...type = TIME_INTEGRAL
...input_parameter = $input
```

Requires the following `parameters`:

```
$input
$input_integral
```

Applies:

$$\mathcal{Y} = \int \mathcal{X} \mathrm{d}t \tag{25}$$

With:

$$\begin{aligned} \mathcal{X} &: \texttt{\$input} \\ \mathcal{Y} &: \texttt{\$input\_integral} \end{aligned}$$

9.6.10 Weibull-distributed variable

```
..material_law
...type = WEIBULL
...parameter = $input_x
...parameter = $input_y
...parameter = $input_z
...scale_parameter = # optional, default value is 1, must be positive
...shape_parameter = # optional, default value is 5, must be positive
...variability = # optional, default value is 0.2, must be less than 1
...weibull_variable_name = # optional, default name is "weibull_variable"
```

Requires the following `parameters`:

```
$input_x
$input_y
$input_z
```

Applies:

$$\begin{aligned} w &= \mathrm{Weibull}(\lambda, k) && (26)\\ x &= x \times (1 - v + v \times w) && (27)\\ y &= y \times (1 - v + v \times w) && (28)\\ z &= z \times (1 - v + v \times w) && (29)\\ & && (30) \end{aligned}$$

With:

$$\begin{aligned} w &: \texttt{weibull\_variable} \\ \lambda &: \texttt{scale\_parameter} \\ k &: \texttt{shape\_parameter} \\ v &: \texttt{variability} \\ x &: \texttt{\$input\_x} \\ y &: \texttt{\$input\_y} \\ z &: \texttt{\$input\_z} \end{aligned} \tag{31}$$

This law can accept any number of `parameter` (at least one).

The weibull variable will be computed during the first simulation for each element. Once calculated, it does not change during the simulation.

9.7 Example

The following example can be used to simulate the serpentine aggregates used in Elleuch's irradiation experiments.

```
.behaviour
..type = LOGARITHMIC_CREEP
# define material parameters
..parameters
...young_modulus = 130e9
...poisson_ratio = 0.2
...imposed_deformation = 0
...temperature = 293
...neutron_fluence = 0
...thermal_expansion_coefficient = 7.5e-6
...radiation_expansion_delay = 84.268
...maximum_radiation_expansion = 0.0972
...neutron_fluence_correction = 0.0099
# damage parameters, purely brittle material in tension
...tensile_strain = 0.00039
...tensile_ultimate_strength = 0
...tensile_ultimate_strain = 0.00039
...material_characteristic_radius = 0.001
...damage_increment = 0.01
...maximum_damage = 0.999
...time_tolerance = 1e-9 # define temperature as function of time
..material_law
...type = LINEAR_INTERPOLATED
...input_parameter = t
...output_parameter = temperature
...file_name = ../examples/data/elleuch/temperature_history_g1
# define neutron fluence as function of time
..material_law
...type = LINEAR_INTERPOLATED
...input_parameter = t
...output_parameter = neutron_fluence
...file_name = ../examples/data/elleuch/fluence_history_g1
# apply thermal expansion
..material_law
...type = THERMAL_EXPANSION
...reference_temperature = 293
# apply radiation-induced expansion
..material_law
...type = RADIATION_INDUCED_VOLUMETRIC_EXPANSION
# apply radiation-induced loss of stiffness
..material_law
...type = LINEAR_INTERPOLATED
...input_parameter = neutron_fluence
```

```
...output_parameter = young_modulus
...file_name = ../examples/data/elleuch/aggregate_damage
```

10 Using templates

Templates are a way to use the same general setup with variations. It is an effective way to do parametric studies, or to simulate set of experiments in which only a few parameters change.

Templates use two files: a `source` which contains the main parameters of the simulation, as well an `specification` file which changes a few parameters of the simulation.

10.1 Source file

The `source` file is a `*.ini` file as described above. In addition, you can define named variables using the `.define` structure. Named variables are composed by a string starting with the character `@`. Later during the file, you can set any element to the value of the named variable. For example, the following file defines a variable named `@paste_young_modulus` (which has a default value of 12 GPa), and attributes the value of the variable `@paste_young_modulus` to the elastic modulus of the sample:

```
.define
..@paste_young_modulus = 12e9
.sample
..height = 0.1
..width = 0.1
..behaviour
...type = ELASTICITY
...young_modulus = @paste_young_modulus
```

You can define as many named variables as required. Named variables are unique: if two share the same name, then the second will be ignored. If you do not define a named variable, then the program stops. Named variables can either be numbers or strings.

The `source` file should be able to run with AMIE without any change.

10.2 Specification file

The `specification` file is a `*.ini` file which contains only the path to the `source` file and a set of named variables definition. By calling the `specification` file, the associated `source` file will be run, except that all named variables in the `source` file will be given the values attributed in the `specification` file. An example of `specification` file would be:

```
.template
..source = path/to/the/source/file.ini
..define
...@paste_young_modulus = 20e9
```

10.3 Command line arguments

Instead of a `specification` file it is possible to set the named variables directly from the command line arguments. The following syntax is equivalent to the previous `specification` file:

```
./2d_composite path/to/the/source/file.ini @paste_young_modulus 20e9
```

There should be a space between the name of the variable and its value (and no other characters). Any named variable without a corresponding value will be ignored.

11 Future features

The following features might be implemented/corrected at some point in the future.

- The first SVG output file seems to have more fields than required. This is a bug.
- Possibility to refine locally the mesh.
- Possibility to not use the SVG output.