
PyAuxetic

Release 2.0.1

The PyAuxetic Team

Apr 04, 2023

CONTENTS:

| | | |
|----------|-----------------------------------|-----------|
| 1 | Getting Started | 3 |
| 2 | Unit Cell Library | 23 |
| 3 | API Reference | 27 |
| 4 | Contribute to the Software | 45 |
| 5 | Licensing | 47 |
| 6 | The PyAuxetic Team | 49 |
| | Python Module Index | 51 |
| | Index | 53 |

PyAuxetic is a Python plugin and library for modeling, analyzing, and post-processing auxetic structures in Abaqus. Its main features are:

- **Free:** The software is provided free of charge for non-commercial use. We use the GPL license that ensures that all derivative software are also free and open source.
- **Open Source:** The entire code and documentation is open source and available on GitHub.
- **Simple GUI:** The software has a simple and elegant GUI that interfaces to Abaqus as a plugin.
- **Powerful API:** The software has powerfull API that can be used for scripting. All GUI functionality (and more) are available from the API. Scripting makes the results highly reproducible and the scripts can be archived.
- **Extensible:** The software is built on a solid object-oriented framework, making it easily extensible. New structures and output types can be added with speed and reliability.
- **Thorough Documentation:** We believe in documenting our methods. You can find in-depth documents about all aspects of the software in our online documentation.

GETTING STARTED

In this section you will learn how to install and use utilize the plugin using the graphical user interface (GUI) and the application programming interface (API).

1.1 Installation and Usage

The software can be used as both a plugin to the Abaqus software or as a library for developing Abaqus scripts. Installation for either mode is very straightforward.

1.1.1 Installing as a Plugin

To install the software as a plugin to Abaqus, follow these steps:

- Obtain a copy of the software from Github. You can either clone/fork the repository or download as a zip.
- Copy the software to you Abaqus *plugins* folder. This is generally in C:\SIMULIA\CAE\plugins\VERSION, where VERSION refers to abaqus version, e.g. 2021.
- Restart Abaqus.
- The plugin can now be seen in the *plugins* menu (Fig. 1.1).

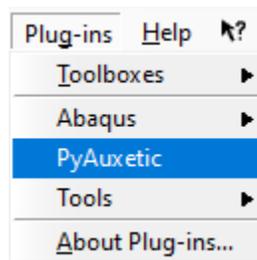


Fig. 1.1: Installed plugin in the plugins menu.

The plugin can now be opened and used. Throughout the manual, there are sections describing how to use the plugin GUI.

1.1.2 Installing as a Library

The software has an API which can be used for writing Python scripts for Abaqus. In order to import the software as a library, simply add the path to the plugin to the search path in your script:

```
import sys
sys.path.append(r'C:\SIMULIA\CAE\plugins\2017\pyauxetic')
```

Now you can import the library using `import pyauxetic`. Throughout the manual, there are sections describing how to use the API to define objects about various parts of the modeling and analysis process. These are then passed to the main API functions. Refer to examples for more information.

1.1.3 Updating the Software

In order to update the software, simply delete the old files and replace them with the new copy of the software.

1.2 Selecting a Structure

The first step for using the software is to select the structure which is to be modeled. Generally, three different questions must be answered regarding the structure:

- What is the basic unit cell used throughout the structure?
- Do the parameters of the unit cell change throughout the structure?
- Does the structure undergo additional transformations?

These questions and their possible answers are discussed in the following sections.

1.3 Defining Unit Cells

1.3.1 Introduction

In the context of this software, the unit cell is defined as the smallest repeating unit used to create a structure.

In order to define a unit cell, its geometrical dimensions must be input. A unit cell may be definable using multiple methods, each called a *variant*, but these only differ in their input values, not the resulting geometry. Henceforth, a set of the mentioned values are referred to a *unit cell parameters*.

There are two considerations when defining unit cell parameters:

1. What *variant* of the unit cell is being defined? Each variant has different parameters and care must be taken not to mistake them for each other. The desired variant can easily be selected both in the GUI and the API.
2. How many unit cells must be defined? For a single homogenous structure only a single set of unit cell parameters is required. However, there are two situations that call for a list of unit cell parameters to be defined:
 - Multiple homogenous structures are being created as a consecutively as a *batch analysis* (See [Batch Modeling](#)).
 - Although a structure can only be made of instances of a single unit cell, homogeneity is not required which means that many different unit cells can be defined for different parts of a structure (See [Assembling the Unit Cells](#)).

1.3.2 Defining Unit Cells using the GUI

The first step is to specify the basic structure information. These include:

- **Unit Cell:** Determines the unit cell used to create the structure.
- **Unit Cell Variant:** A unit cell may be definable using multiple methods which are listed here. The choice of *Unit Cell* and *Unit Cell Variant* determines the unit cell parameters which must be input in the future.
- **Structure Type:** Discussed in #TODO.
- **Modeling Mode:** Three choices are available: *Uniform (Single)*, *Uniform (Batch)*, and *Non-Uniform*. The first calls for a single set of unit cell parameters and the other two require a list of unit cell parameters.

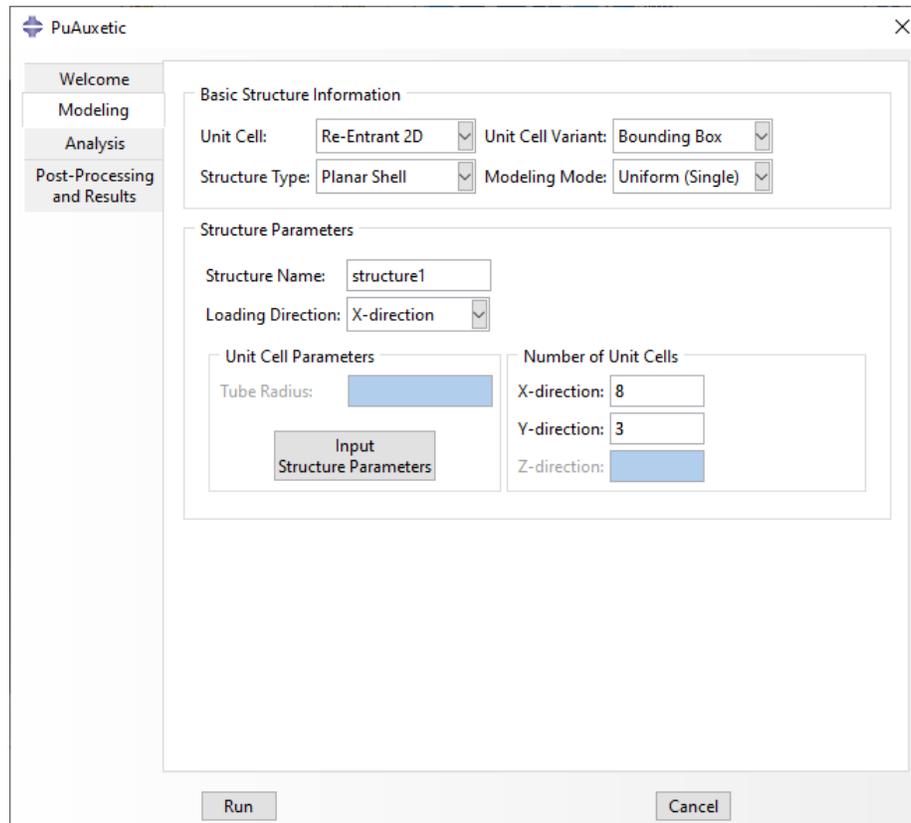


Fig. 1.2: Basic structure information in the modeling tab.

After a combination of parameters are specified in *Basic Structure Information*, the *Structure Parameters* frame is automatically activated for that combination. Contents of this frame depend on the structure, but can include the following:

- **Structure Name/Prefix:** The name given to the structure, or the prefix used for a batch analysis.
- **Loading Direction:** Discussed in *Assembling the Unit Cells*.
- **Unit Cell Parameters Frame:** A button which opens a window in which unit cell parameters can be specified. For non-uniform structures, the window also asks for the *Structure Map* (See *Assembling the Unit Cells*). This frame also has any possible parameters needed for the selected *Structure Type*. This is discussed in length in #TODO.
- **Number of Unit Cells Frame:** This frame is only shown for uniform (single and batch) structures. Depending on the unit cell, number of unit cells in the x, y, and z direction can be specified.

Title of the push button described above changes based on *Modeling Mode*. Regardless, it opens a new pop-up window which asks for the required parameters.

For *Uniform* structures, the window includes only a table asking for the unit cell parameters needed for the selected unit cell variant. Unit cell ID is automatically set to 1. A sample window is shown in Fig. 1.3.

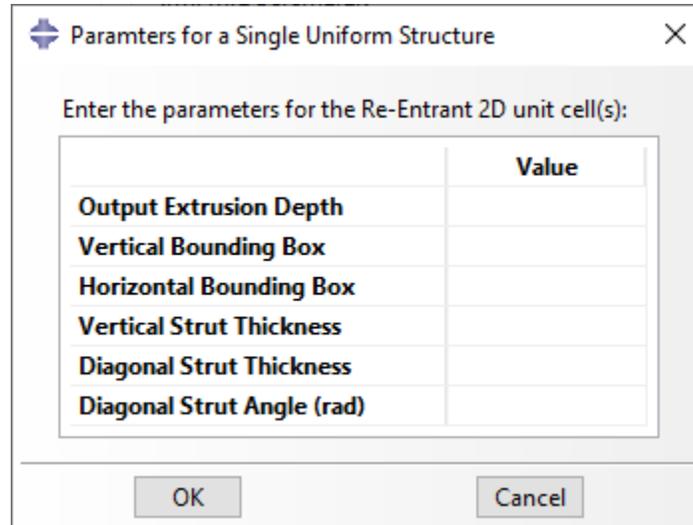


Fig. 1.3: Structure Parameters pop-up window for a uniform (single) structure.

If *Uniform (Batch)* or *Non-Uniform* modeling modes are selected, the pop-up window asks for a list of unit cell parameters in tabular format. Here, each row has a unit cell ID, which is used as analysis ID for batch modeling. Use of unit cell ID in non-uniform structures is discussed in *Assembling the Unit Cells*. A sample window is shown in Fig. 1.4.

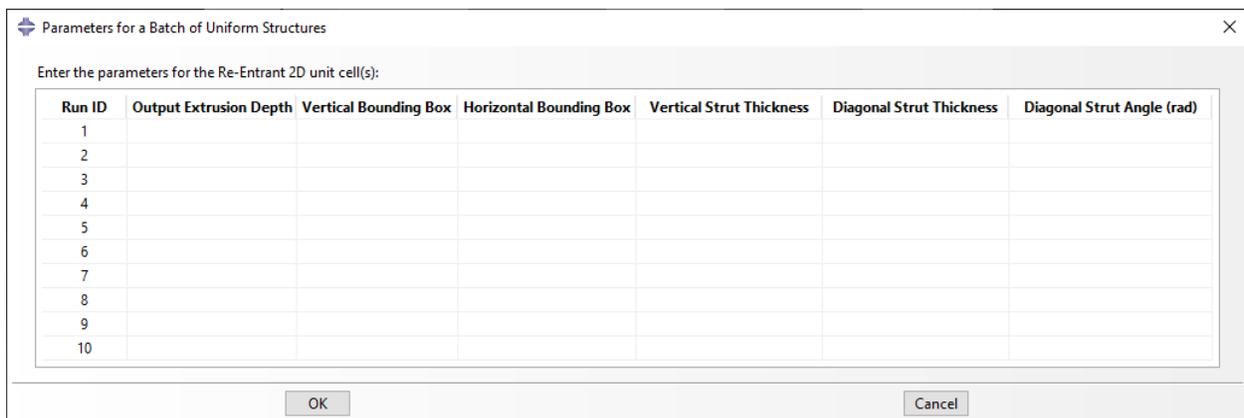


Fig. 1.4: Structure Parameters pop-up window for a uniform (batch) structure.

1.3.3 Defining Unit Cells using the API

The mentioned options and parameters can also be specified using the API. The first variable is *structure_type*, which simultaneously selects the *Unit Cell Name* and its *Structure Type*. For example:

```
# Define a Re-Entrant 2D unit cell for a planar shell structure.
structure_type = 'reentrant2d_planar_shell'
```

Unit cell parameters are defined as subclasses of `namedtuple`, which are defined in `classes.auxetic_unit_cell_params`. For example, the *Re-Entrant 2D* unit cell can be defined using three different subclasses of `namedtuple`, namely:

- `classes.auxetic_unit_cell_params.Reentrant2DUcpFull`
- `classes.auxetic_unit_cell_params.Reentrant2DUcpBox`
- `classes.auxetic_unit_cell_params.Reentrant2DUcpSimple`

The geometrical significance of these definition methods are explained in depth in #TODO. The API expects one of these or a homogenous Iterable of one of these depending on how many are necessary. It then makes sure that the list includes only one definition method and that it is relevant to the selected unit cell. Two examples are shown below:

First, the necessary libraries must be imported:

```
# Import the necessary libraries:
from pyauxetic.classes.auxetic_unit_cell_params import *
from pyauxetic.classes.auxetic_structure_params import *

## A single set of unit cell parameters:
# Method 1:
unit_cell_params = Reentrant2DUcpBox(
    id                = 1  ,
    extrusion_depth  = 5  ,
    horz_bounding_box = 20 ,
    vert_bounding_box = 24 ,
    vert_strut_thickness = 2 ,
    diag_strut_thickness = 1.5,
    diag_strut_angle  = 70
)

# Method 2:
unit_cell_params = Reentrant2DUcpBox(1, 5, 20, 24, 2, 1.5, 70)

## A list of unit cell parameters:
# Define three unit cells for a non-uniform structure or a batch of uniform structures.
# Note that the first argument (id) is unique for each unit cell.
unit_cell_params_list = []
# (id, extrusion_depth, horz_bounding_box, vert_bounding_box,
#  vert_strut_thickness, diag_strut_thickness, diag_strut_angle)
unit_cell_params_list.append( Reentrant2DUcpBox(1, 5, 20, 24, 3.0, 1.5, 60) )
unit_cell_params_list.append( Reentrant2DUcpBox(2, 5, 20, 24, 3.0, 1.5, 60) )
unit_cell_params_list.append( Reentrant2DUcpBox(3, 5, 20, 24, 2.0, 1.5, 60) )
```

1.4 Assembling the Unit Cells

The next step is to assemble the defined unit cells. The assembly method depends on the choice of unit cell, but the following general steps apply:

1. Distribution of the unit cells is defined in a table named *Structure Map* which looks like Fig. 1.5.
 - For uniform structures, there is only one unit cell with a known ID, usually 1, which is distributed throughout the structure. The number of unit cells in different directions is used to create the *Structure Map* table and all cells (elements) are equal to that ID.
 - For non-uniform structures, a list of unit cells have already been defined (see *Defining Unit Cells*), and the *Structure Map* table is defined separately using the GUI or the API.

| | | | | | | |
|---|-------|-------|-------|-------|-------|-------|
| 3 | (0,3) | (1,3) | (2,3) | (3,3) | (4,3) | (5,3) |
| 2 | (0,2) | (1,2) | (2,2) | (3,2) | (4,2) | (5,2) |
| 1 | (0,1) | (1,1) | (2,1) | (3,1) | (4,1) | (5,1) |
| 0 | (0,0) | (1,0) | (2,0) | (3,0) | (4,0) | (5,0) |
| | 0 | 1 | 2 | 3 | 4 | 5 |

Fig. 1.5: Schematics of the Structure Map table for a structure loaded in the X-Direction. Zero-based numbering is used for rows and columns in accordance with the API. The blue vertical lines are loading ribbons.

2. Loading direction is defined and appropriate *loading ribbons* are created for the model. These can be seen in Fig. 1.5.
3. Unit cells are checked to have the same bounding box (height, width, and depth). and if they do, they are instantiated and translated to their locations.
4. *Loading Ribbons* are instantiated and translated to the appropriate locations.
5. The entire structure is merged and unnecessary parts are deleted.

It should be noted that if a solid structure is requested from a 2D structure, all the aforementioned parts are extruded by the defined *Extrusion Depth* which must be equal for all unit cells. This also applies to STL or STP export of a shell part.

1.4.1 Assembling the Unit Cells using the GUI

After defining basic structure information, select the appropriate *Loading Direction*. The next step depends on the structure:

- For uniform structures the *Number of Unit Cells* frame is activated. These numbers are automatically converted into the table by the plugin.
- For non-uniform structures, after selecting the *Input Structure Parameters* button the pop-up windows in Fig. 1.6 appears.

The top part is used for defining the unit cell parameters as explained in *Defining Unit Cells*.

The bottom part of this window has two spinners which determine how many unit cells must be present in the *X* and *Y* directions. Changing these resizes the *structure map* table. Afterwards, the table must be completed with IDs of the unit cells defined in the top section.

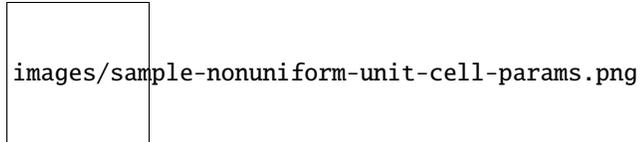


Fig. 1.6: Structure Parameters and Structure Map pop-up window for a non-uniform structure.

1.4.2 Assembling the Unit Cells using the API

The mentioned options and parameters can also be specified using the API. The first variable is *pattern_params*, which contains all patterning information.

First, the necessary libraries must be imported:

```
# Import the necessary libraries:
from pyauxetic.classes.auxetic_structure_params import *
```

Then, For a uniform structure:

```
# Define the PatternParams object.
# Note that structure_map is set to None.
pattern_params = PatternParams(
    pattern_mode     = 'uniform',
    num_cell_repeat = (8, 3)   ,
    structure_map    = None
)
```

And for a non-uniform structure:

```
# Import numpy.
import numpy as np

#Define the structure_map similar to the figure.
structure_map = np.array([
    [1, 2, 4, 9, 10, 8, 7, 4, 2, 2],
    [1, 2, 4, 9, 10, 8, 1, 4, 2, 2],
    [1, 2, 4, 9, 10, 8, 7, 4, 2, 2],
    [1, 2, 4, 9, 10, 8, 7, 4, 2, 2],
])

# Define the PatternParams object.
# structure_map must be flipped and transposed because of the way
# python iterates over it.
# Note that num_cell_repeat is set to None.
pattern_params = PatternParams(
    pattern_mode     = 'nonuniform',
    num_cell_repeat = None           ,
    structure_map    = np.fliplr( structure_map.T )
)
```

Also, loading direction must be defined using the *loading_params* object. For example:

```
# Define the LoadingParams object.
loading_params = LoadingParams(
    type      = 'disp',
    direction = 'x' ,
    data      = 20.0
)

# If only modeling is being performed, the direction attribute is enough,
# but this is not recommended.
loading_params = LoadingParams(direction = 'x')
```

1.5 Different Structure Modes

In addition to a unit cell, a structure has a *Structure Mode* which makes up the general geometry of the structure. Currently, only one *Structure Mode* is available:

1.5.1 Planar Shell Structure

Overview

A planar shell structure is a 2D structure which is meshed with 2D elements. When exported, it is extruded in the third direction by a given amount (see *Requesting Output*). A sample is shown below:

Boundary Conditions

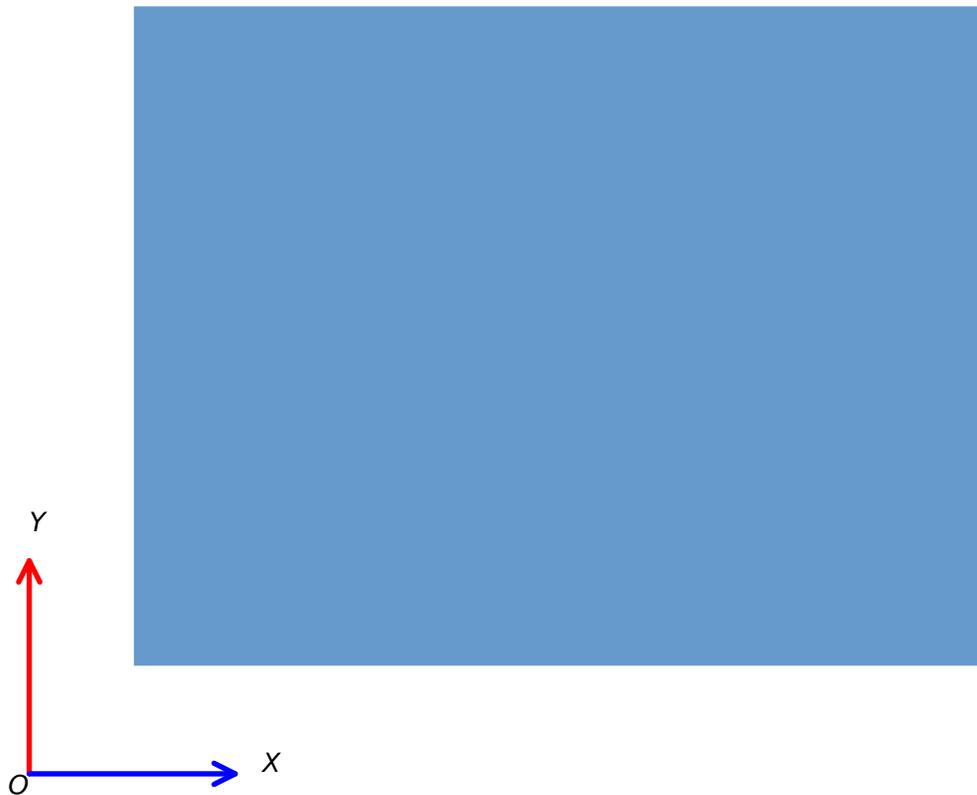
In this structure, *RP-1* is a reference point tied to the the first loading edge (*LD-Edge-1*). This point is fixed in space. The second reference point (*RP-2*) is tied to the second loading edge (*LD-Edge-2*) and receives the loading. A schematic of loading on planar shell structures is shown below:

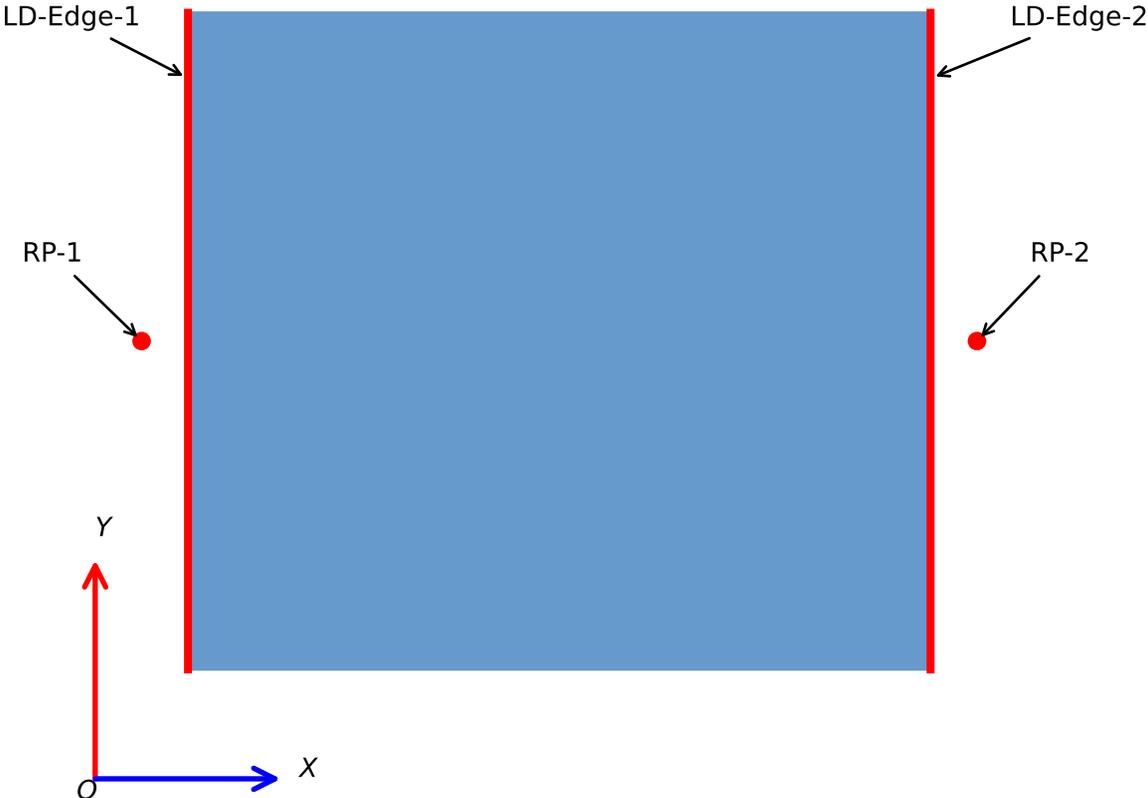
Special Outputs

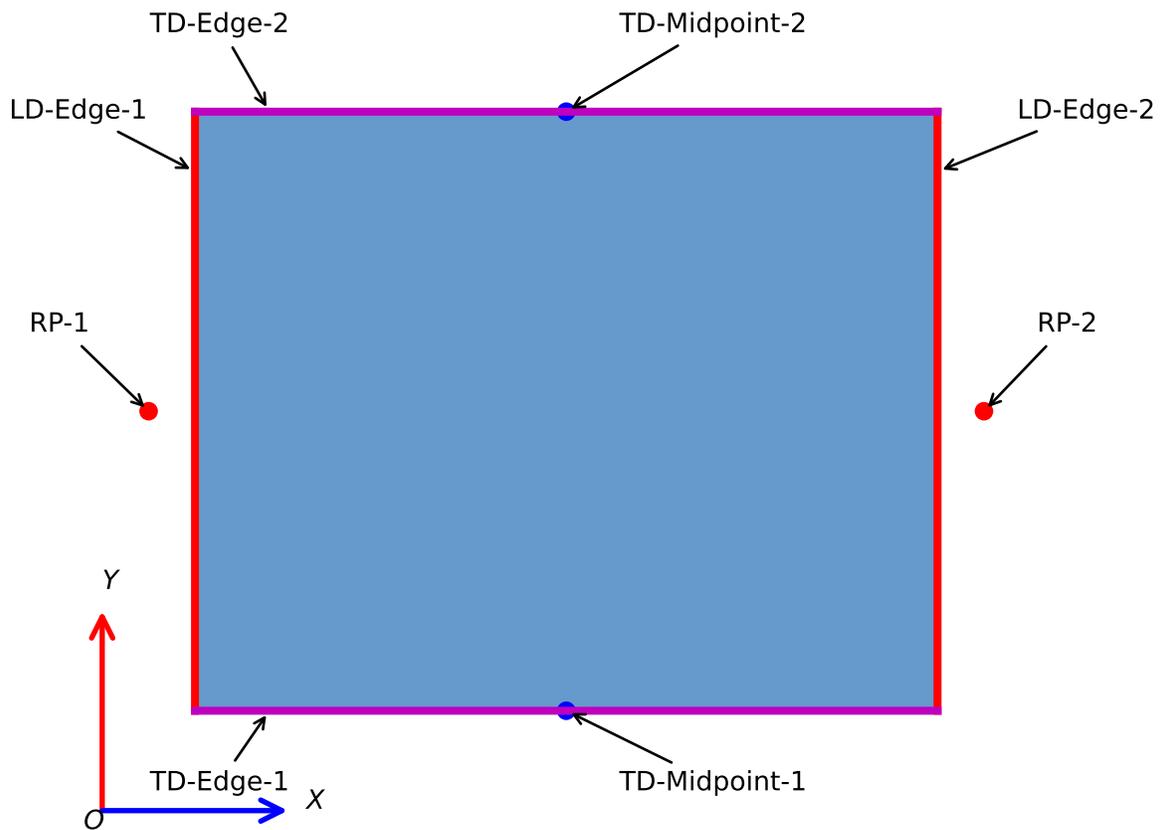
For this structure, two transverse edges *TD-Edge-1* and *TD-Edge-2* are defined and used for processing outputs. Afterwards, two special outputs are calculated:

- **Poisson's Ratio at the Midpoint:** Two nodes named 'TD-Midpoint-1' and 'TD-Midpoint-2' are always defined in the model using partitioning. The relative displacement of these nodes and the reference points *RP-1* and *RP-2* are then used to define the Poisson's Ratio at the Midpoint.
- **Mean Poisson's Ratio:** For each of *TD-Edge-1* and *TD-Edge-2*, average values of the displacement in that edge is calculated. The difference between the two displacements is the relative displacement at the transverse direction which along with the relative displacement of the reference points *RP-1* and *RP-2* are used to define the Mean Poisson's Ratio.

These geometries are shown in the following figure:







1.6 Batch Modeling

Batch modeling refers to running a series of modeling and analysis operations in consecutive order. Afterwards, output data are compiled into a separate report. Currently, this operation is only available for uniform structures.

1.6.1 Batch Modeling using the GUI

Definition of a batch modeling job is similar to a single analysis discussed in *Defining Unit Cells*. After *Modeling Mode* is set to *Uniform (Batch)*, the only differences are:

- Instead of a *structure name*, a *structure prefix* is input. The number *unit cell ID* (see below) will then be appended to the prefix to form *structure name*.
- Unit cell parameters pop-up window shows a table for entering the parameters. This is discussed in *Defining Unit Cells*. Here, each row is used for a separate analysis and its number (and name) is equal to *unit cell ID* of that row.
- After all analyses are complete, the results are compiled automatically. See #TODO for more information.

1.6.2 Batch Modeling using the API

First, *structure_prefix* and *unit_cell_params_list* must be defined. For example:

```
## Define structure_prefix.
structure_prefix = 'unnamed'

## Define unit_cell_params_list.
# Define three unit cells for a batch of uniform structures.
# Note that the first argument (id) is unique for each unit cell.
unit_cell_params_list = []
# (id, extrusion_depth, horz_bounding_box, vert_bounding_box,
#  vert_strut_thickness, diag_strut_thickness, diag_strut_angle)
unit_cell_params_list.append( Reentrant2DUcpBox(1, 5, 20, 24, 3.0, 1.5, 60) )
unit_cell_params_list.append( Reentrant2DUcpBox(2, 5, 20, 24, 3.0, 1.5, 60) )
unit_cell_params_list.append( Reentrant2DUcpBox(3, 5, 20, 24, 2.0, 1.5, 60) )

# structures will be named 'unnamed-001', 'unnamed-003', and 'unnamed-003'.
```

Afterwards, the `pyauxetic.main.main_batch()` function is called for analysis. See #TODO for more information.

1.7 Assigning Material Properties

A number of different material properties can be defined using this software. These are all available in the GUI and API but, as with any Abaqus analysis, care should be taken to define only the necessary material properties. Furthermore, some definitions may not be compatible with each other. For example, elastic and hyperelastic properties can be defined together, but will raise an error in the analysis. Finally, all material property data is passed to the Abaqus API. Any invalid inputs are either caught by the API or result in failed or erroneous analyses.

1.7.1 Assigning Material Properties using the GUI

Currently, only elastic and hyperelastic material properties can be defined using the GUI. This may be updated in the future. Fig. 1.7 shows the relevant frame in the analysis tab. After selecting a material mode, the relevant data can be input.

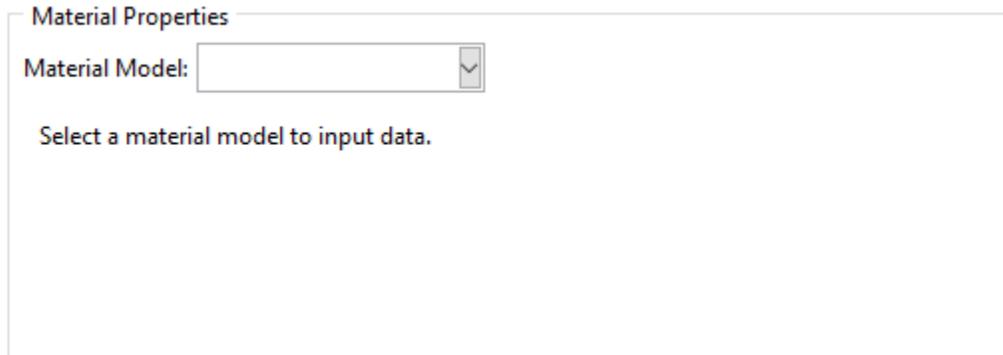


Fig. 1.7: Material Properties frame.

1.7.2 Assigning Material Properties using the API

Material properties are defined by defining a *MaterialParams* object. A list of all attributes and their significance can be found in `classes.auxetic_structure_params.MaterialParams`.

As an example, the following script defines a *MaterialParams* object with *density* and *hyperelastic* attributes.

```
# Define the material_params object.
# Undefined attributes default to None.
material_params = MaterialParams(
    density      = 1.00,
    hyperelastic = ('marlow',
                   (
                     (0.0, 0.0),
                     (1.87019, 0.021918),
                     (3.76788, 0.041096),
                     (5.63806, 0.062101),
                     (7.48075, 0.086758),
                     (9.15842, 0.122374),
                     (10.4785, 0.170776),
                     (11.4686, 0.226484),
                     (12.3212, 0.285845),
                     (13.0638, 0.346119),
                     (13.7514, 0.407306),
                     (14.5215, 0.468493),
                     (15.4015, 0.526941),
                     (16.3916, 0.583562),
                     (17.3542, 0.641096),
                     (18.2893, 0.699543),
                     (19.2244, 0.757078),
                     (20.242, 0.812785),
                     (21.3146, 0.866667),
```

(continues on next page)

(continued from previous page)

```
(22.3872, 0.921461),  
(23.4598, 0.977169),  
(24.5325, 1.03288 ),  
(25.6601, 1.08676 ),  
(26.7877, 1.14064 ),  
(27.8603, 1.19543 ),  
(28.8779, 1.25205 ),  
(29.868 , 1.30868 ),  
(30.7756, 1.36621 ),  
(31.6832, 1.42557 ),  
(32.5908, 1.48402 ),  
(33.4983, 1.54247 ),  
(34.3784, 1.60091 ),  
(35.286 , 1.65936 ),  
(36.1661, 1.71781 ),  
(37.0462, 1.77626 ),  
(37.9813, 1.8347 ),  
(38.8889, 1.89315 ),  
(39.824 , 1.9516 ),  
(40.7591, 2.00913 ),  
(41.7217, 2.06667 ),  
(42.6843, 2.1242 ),  
(43.7019, 2.18082 ),  
(44.7195, 2.23653 ),  
(45.7096, 2.29315 ),  
(46.6997, 2.34977 ),  
(47.7173, 2.40639 ),  
(48.6249, 2.46484 ),  
(49.505 , 2.5242 ),  
(50.44 , 2.58265 ),  
(51.4301, 2.63927 ),  
(52.3927, 2.6968 ),  
(53.3828, 2.75434 ),  
(54.3454, 2.81096 ),  
(55.198 , 2.87032 ),  
(55.8581, 2.93242 )  
)
```

)

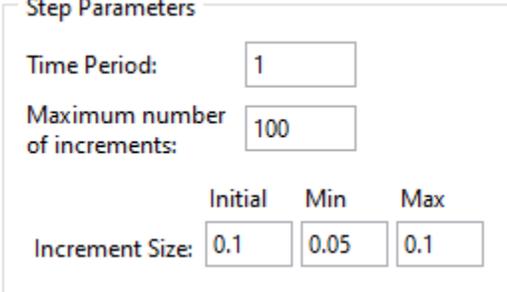
```
# Note that the above tuple can be defined in a single line  
# this is only recommended after a successful analysis.  
# Also, any errors (sorting, etc.) are only caught by Abaqus.
```

1.8 Adjusting Step Parameters

The analysis is done in one step. The main parameters of this step can (and should) be adjusted to complete the analysis in a timely manner.

1.8.1 Adjusting Step Parameters using the GUI

The *Step Parameters* frame of the analysis tab can be seen in Fig. 1.8. Note that in a batch analysis these parameters are used for all models. Therefore, care should be taken to select a set of parameters suitable for all analysis. If a model needs a different set, it should be run as a single analysis.



The screenshot shows a window titled "Step Parameters" with the following fields and values:

- Time Period: 1
- Maximum number of increments: 100
- Increment Size:

| Initial | Min | Max |
|---------|------|-----|
| 0.1 | 0.05 | 0.1 |

Fig. 1.8: Step parameters frame with the default values.

1.8.2 Adjusting Step Parameters using the API

Step parameters are defined by defining a *StepParams* object. A list of all attributes and their significance can be found in `classes.auxetic_structure_params.StepParams`. An example is shown below:

```
# Define the step_params object.
# Undefined attributes default to None.
step_params = StepParams(
    time_period = 0.1 ,
    init_inc_size = 0.01 ,
    min_inc_size = 0.005,
    max_inc_size = 0.05 ,
    max_num_inc = 10000
)
```

1.9 Adjusting Job Parameters

For each analysis, a job is defined. The main parameters of the job can (and should) be adjusted to satisfy accuracy and time constraints. Note that in a batch analysis these parameters are used for all models. Therefore, care should be taken to select a set of parameters suitable for all analysis. If a model needs a different set, it should be run as a single analysis.

1.9.1 Adjusting Job Parameters using the GUI

The *Job Parameters* frame of the analysis tab can be seen in Fig. 1.9. It should be noted that the API offers a more complete list of options.

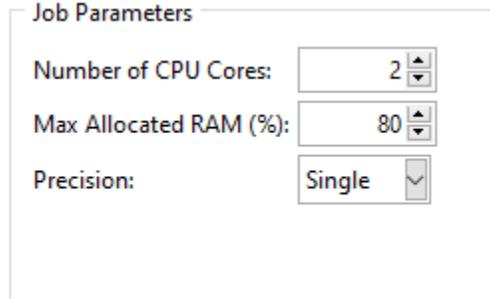


Fig. 1.9: Job parameters frame with the default values.

1.9.2 Adjusting Job Parameters using the API

Job parameters are defined by defining a *JobParams* object. A list of all attributes and their significance can be found in `classes.auxetic_structure_params.JobParams`. An example is shown below:

```
# Define the job_params object.
# Undefined attributes default to None.
job_params = JobParams(
    description      = 'This is a sample job.',
    numCpus          = 4 ,
    memoryPercent    = 80,
    explicitPrecision = 'SINGLE',
    nodalOutputPrecision = 'SINGLE',
)
```

1.10 Defining Boundary Conditions

The software defines two boundary conditions (BCs) on every model. These are applied to reference points (RPs) which are tied to suitable geometry using multi-point constraint equations. They are as follows:

1. An Encastre (fixed) boundary condition is defined on the 'RP-1-set' reference point. This means that all points tied to this RP are fixed in all translational and rotational directions. This BC is applied starting from the *Initial* step.
2. A second boundary condition is applied in the first (and only) step on the 'RP-1-set' reference point. Currently, the following loadings are available:
 - **Uniaxial Monotonic Displacement BC:** Only one value is required.
 - **Uniaxial Monotonic Concentrated Force:** Only one value is required.

It should be noted that the of the loading influences the ribbons created for the structure (see *Assembling the Unit Cells*). The boundary condition is applied based on *Structure Type* defined when creating the structure. See *Different Structure Modes* for a list of structure types and the boundary conditions applied to them.

When running a batch analysis, these parameters are used for all models. If a model needs different boundary conditions, it should be run as a separate analysis.

1.10.1 Defining Boundary Conditions using the GUI

The *Loading Parameters* frame of the analysis tab can be seen in Fig. 1.10.



The screenshot shows a window titled "Loading" with three input fields. The first is "Loading Direction:" followed by a text box. The second is "Loading Type:" followed by a dropdown menu with a downward arrow. The third is "Loading Value:" followed by a text box.

Fig. 1.10: The loading parameters frame.

1.10.2 Defining Boundary Conditions using the API

Job parameters are defined by defining a *LoadingParams* object. A list of all attributes and their significance can be found in `classes.auxetic_structure_params.LoadingParams`. An example is shown below:

```
# Define the loading_params object
# for a displacement in the x direction.
loading_params = LoadingParams(
    type      = 'disp',
    direction = 'x'  ,
    data      = 20.0
)
```

1.11 Meshing the Structure

For each analysis, meshing parameters can be defined. These are then applied to the structure. If batch modeling and analysis is performed, the parameters are used for all models regardless of geometry. Care should be taken to select suitable values and run problematic geometries separately.

1.11.1 Meshing the Structure using the GUI

The *Mesh Parameters* frame of the analysis tab can be seen in Fig. 1.11. It should be noted that the API offers a more complete list of options. Also, *Element Code* drop-down menu only contains a limited number of element codes. More element codes can be easily added upon request.

The image shows a window titled "Mesh" with three input fields. The first field is labeled "Element Shape:" and has a dropdown arrow on its right side. The second field is labeled "Element Code(s):" and also has a dropdown arrow on its right side. The third field is labeled "Seed Size" and is a standard text input box.

Fig. 1.11: The mesh parameters frame.

1.11.2 Meshing the Structure using the API

Mesh parameters are defined by defining a *MeshParams* object. A list of all attributes and their significance can be found in `classes.auxetic_structure_params.MeshParams`. An example is shown below:

```
# Define the mesh_params object.
# Undefined attributes default to None.
mesh_params = MeshParams(
    seed_size      = 1.0      ,
    elem_shape     = 'QUAD'   ,
    elem_code      = ('CPE4H') ,
    elem_library   = 'STANDARD'
)
```

1.12 Requesting Output

The software automatically saves results of the analysis to the current working directory in a folder with the same name as the structure. These outputs generally fit in one of the following categories:

- **Analysis Files:** These are the files created automatically by Abaqus. These include *.cae*, *.odb*, and various Abaqus input and log files.
- **Numerical Results:** The software automatically processes the ODB and produces a suitable report in a tabular format. For batch analyses, this is done for each structure and an additional report is created which summarizes the results of all models.
- **Graphic Results:** These are pictures of the structure before, after, and during analysis. They also include plots of the numerical results.
- **Exported Model:** The created model can be exported in *STL* and *STP* formats. These can then be used in a variety of CAD software or additive manufacturing processors. The loading ribbons can have a new and separate width to accommodate testing and planar shell models are extruded by a given amount.

1.12.1 Requesting Output using the GUI

All of the mentioned outputs can be seen the *Post-Processing and Results* tab of the GUI which is seen in :numref:``. Note that the API offers a more complete list of options and finer control over them.

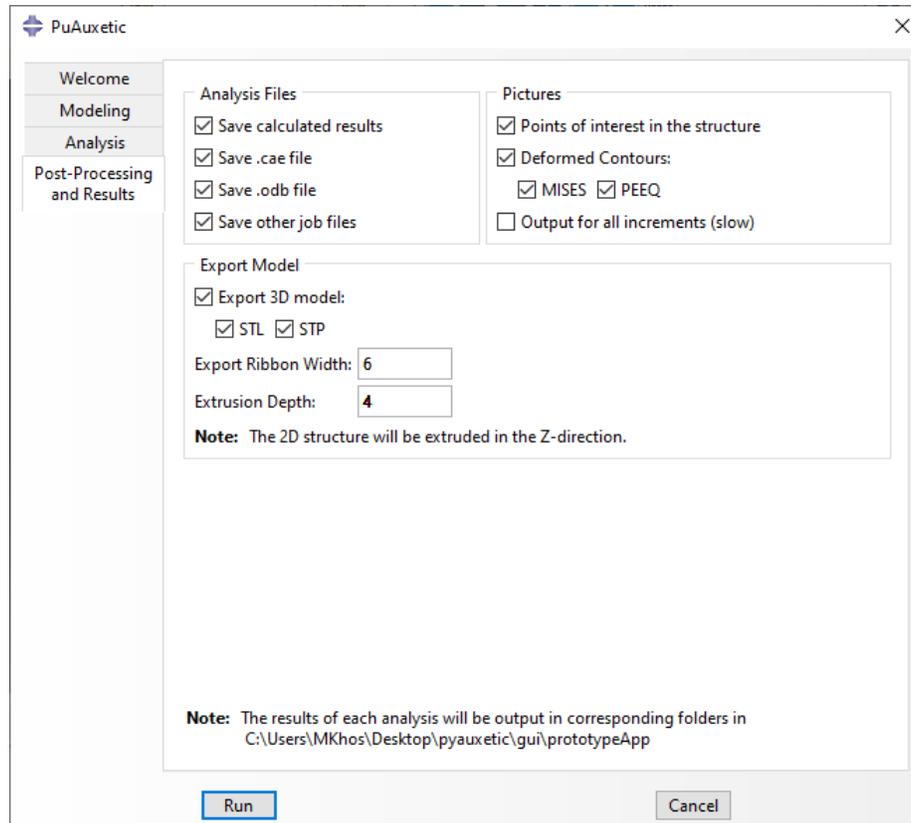


Fig. 1.12: The Post-Processing and Results tab.

1.12.2 Requesting Output using the API

Output requests are defined by defining a *OutputParams* object. A list of all attributes and their significance can be found in *classes.auxetic_structure_params.OutputParams*. An example is shown below:

```
# Define the output_params object.
# Undefined attributes default to None.
output_params = OutputParams(
    result_folder_name    = None,
    save_cae              = True,
    save_odb              = True,
    save_job_files        = True,
    export_extrusion_depth = 5.0 ,
    export_ribbon_width   = 4.0 ,
    export_stl            = True,
    export_stp            = True
)
```


UNIT CELL LIBRARY

In this section the various unit cells are explained.

2.1 Re-Entrant 2D

The Re-Entrant 2D is a two-dimensional unit cell that has been studied for a long time. It has been originally taken from the honeycomb structure [#TODO: write a paragraph and cite].

2.1.1 Variants

Three variants have been defined for this unit cell. Use of the *Bounding Box* variant is strongly recommended.

Full Parameters

This is the basic variant that gives the most amount of modeling freedom. All other variants are converted to this variant. Use of this variant for non-uniform structures is not recommended. Fig. 2.1 shows this unit cell variant and the points that need to be traced to draw it.

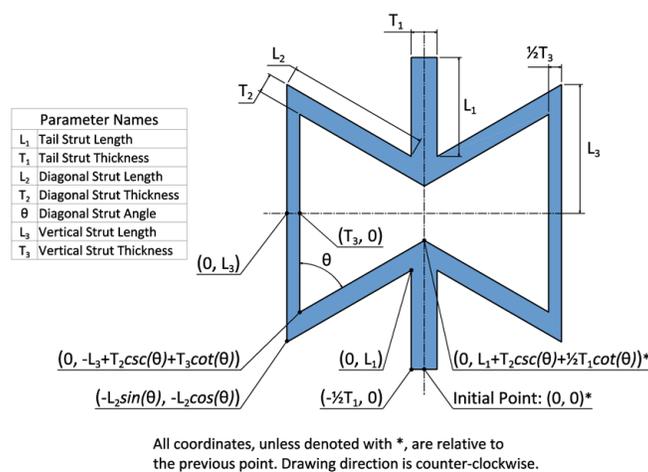


Fig. 2.1: The 'Full Parameters' variant of the Re-Entrant 2D unit cell

After drawing a sketch based on the points in Fig. 2.1, the dimensional constraints are defined. It should be noted that only one quarter of the model is drawn, which is then mirrored twice to obtain the final sketch.

Unit cell parameters for this structure are defined using the `classes.auxetic_unit_cell_params.Reentrant2DUcpFull` class.

Bounding Box

In order to assemble a non-uniform structure made from this unit cell, all unit cells must have the same bounding box and the other variants cannot be defined based on unit cell bounding box. This variant was developed to fill that need. Fig. 2.2 shows this unit cell variant and its parameters.

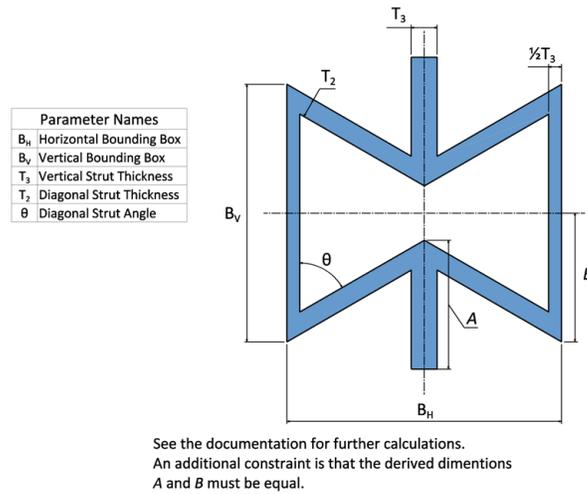


Fig. 2.2: The ‘Bounding Box’ variant of the Re-Entrant 2D unit cell

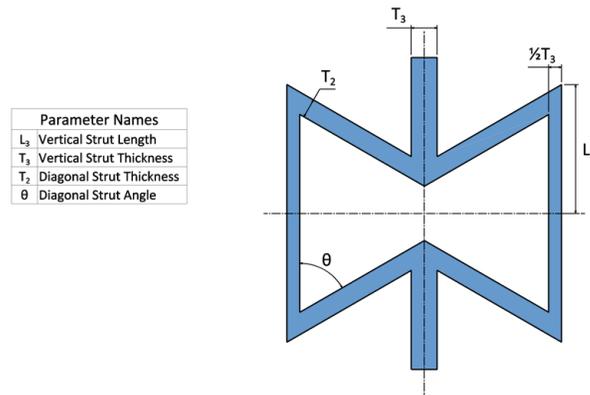
The parameters are then converted into parameters for the *Full Parameters* variant based on the following equations (note the order of calculations):

$$\begin{aligned}
 T'_1 &= T_1 \\
 T'_2 &= T_2 \\
 T'_3 &= T_3 \\
 \theta' &= \theta \\
 L'_2 &= \frac{0.5 B_H - 0.5 T_3}{\sin(\theta)} \\
 L'_3 &= 0.5 B_V + L'_2 \cos(\theta) + \frac{T_2}{\sin(\theta)} + \frac{0.5 T_3}{\tan(\theta)} \\
 L'_1 &= 0.5 L'_3 - \frac{T_2}{\sin(\theta)} - \frac{0.5 T_1}{\tan(\theta)}
 \end{aligned} \tag{2.1}$$

Unit cell parameters for this structure are defined using the `classes.auxetic_unit_cell_params.Reentrant2DUcpBox` class.

Simplified

This variant simplifies the parameters of the *Full Parameters* variant. Use of this variant for non-uniform structures is not recommended. Fig. 2.3 shows this unit cell variant and its parameters.



See the documentation for further calculations.

Fig. 2.3: The ‘Simplified’ variant of the Re-Entrant 2D unit cell

The parameters are then converted into parameters for the *Full Parameters* variant based on Eqs. (2.2). It should be noted that L'_2 is a dummy index and its value will be modified using dimensional constraints when sketching the *Full Parameters* unit cell variant.

$$\begin{aligned}
 T'_1 &= T_3 \\
 T'_2 &= T_2 \\
 T'_3 &= T_3 \\
 \theta' &= \theta \\
 L'_1 &= 0.5 L_3 - \frac{T_2}{\sin(\theta)} - \frac{0.5 T_1}{\tan(\theta)} \\
 L'_2 &= \frac{2}{3} L_3 \\
 L'_3 &= L_3
 \end{aligned} \tag{2.2}$$

Unit cell parameters for this structure are defined using the `classes.auxetic_unit_cell_params.Reentrant2DUcpSimple` class.

2.1.2 Assembly

Assembly of this unit cell is very straightforward. The unit cells must satisfy two requirements:

- They must have the same bounding box (height and width). As such, only the *Bounding Box* variant is guaranteed to work for non-uniform structures.
- In the distance between the initial point in the unit cell and the final drawn point which is on top of it in Fig. 2.1 must be the same for all models.

Assembly is performed as explained in *Assembling the Unit Cells*.

Currently, only shell structure model is supported which is assembled similar to Fig. 2.4.

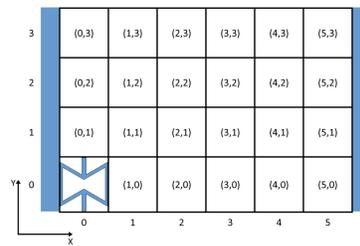


Fig. 2.4: Assembly structure map used for a shell Re-Entrant 2D structure.

API REFERENCE

This reference contains a detailed explanation of all public and private packages, classes, and functions used in this library. You should always consult the examples and the source code for usage.

3.1 Main Functions

Main functions of the PyAuxetic API.

This package contains the main functions used in the PyAuxetic API which are used for creating and analyzing one or more auxetic structures. It also defines the bindings for the GUI library used in the Abaqus plugin.

```
pyauxetic.main.main_single(unit_cell_name, structure_name, unit_cell_params, pattern_params,  
                           material_params, loading_params, mesh_params, job_params, output_params,  
                           step_params=None, run_analysis=True, is_part_of_batch=False)
```

Model and analyze a single auxetic structure.

Parameters

- **unit_cell_name** (*str*) – Type of the structure. See #TODO for a complete list of values.
- **structure_name** (*str*) – Name of the structure. Used for all output files.
- **unit_cell_params** – Parameters describing the unit cell geometry. Valid classes must be selected from `classes.auxetic_unit_cell_params` based on `unit_cell_name`. For nonuniform structures, must be a tuple where all unit cell ids used in `structure_map` are defined.
- **pattern_params** (`PatternParams`) – Special namedtuple describing the parameters for patterning the unit cell(s). See class for full description of options.
- **material_params** (`MaterialParams`) – Special namedtuple describing the material used for modeling and analysis. See class for full description of options.
- **step_params** (`StepParams`) – Special namedtuple describing the step defined for analysis. If not specified, the default values of the namedtuple are used. See class for full description of options. Defaults to `None` which uses the default step values.
- **loading_params** (`LoadingParams`) – Special namedtuple describing the loading and boundary conditions applied to the model. See class for full description of options.
- **mesh_params** (`MeshParams`) – Special namedtuple describing the mesh applied to the model. See class for full description of options.
- **job_params** (`JobParams`) – Special namedtuple describing the job created for analysis. See class for full description of options.

- **output_params** (`OutputParams`) – Special namedtuple describing the parameters for outputting the results of modeling and analysis. See class for full description of options.
- **run_analysis** (`bool`) – If `True`, The model is analyzed. Otherwise `material_params`, `step_params`, `mesh_params`, `job_params`, and `output_params` are need not be defined. Defaults to `True`.
- **is_part_of_batch** (`bool`) – If calling from `main_batch()`, must be set to `True`. Defaults to `False`.

Returns

An object of a subclass of `AuxeticStructure` class.

```
pyauxetic.main.main_batch(unit_cell_name, structure_prefix, unit_cell_params_list, pattern_params,
                          material_params, loading_params, mesh_params, job_params, output_params,
                          step_params=None, run_analysis=True)
```

Run a number of analysis in succession and merge the results to a single csv file.

All paramters of this function are the same as `main_single()`.

The exceptions are:

Parameters

- **structure_prefix** (`str`) – The prefix used for all structures. This prefix together with a number is used for defining `structure_name`.
- **unit_cell_params_list** – A list of `unit_cell_params` for the structures. The id in each parameter must be unique and is used for defining `structure_name`.

All other parameters are passed without change or validation.

```
pyauxetic.main.main_gui_proxy(**kwargs)
```

This function is not documented. You need extensive knowledge of Abaqus GUI design to modify it. Use caution and test extensively.

3.2 ABC for Auxetic Unit Cells

```
class pyauxetic.classes.auxetic_unit_cell.AuxeticUnitCell(model, params)
```

Abstract base class for defining an auxetic unit cell. It defines the core behavior and must be subclassed for different unit cells.

```
__init__(model, params)
```

Initialize the unit cell with the given parameters.

Child classes must first define `self.name` and afterwards call this function using `super(ChildClass, self).__init__(params)`. This function then creates the `self.sketch` and `self.part_main`.

Logging is done in the child classes.

Parameters

- **model** (`Model`) – Abaqus Model object in which the unit cell will be created.
- **params** – Parameters describing the unit cell geometry. See child classes for valid object types.

```
property part_3dprint
```

Abaqus Part object suitable for 3D export. If the part does not exist, it will be created.

property part_main

The main Abaqus Part of the structure which can be planar or 3D. If the part does not exist, it will be created.

abstract create_sketch()

Create the 2D sketch for the unit cell. See child classes for implementation.

abstract create_part_main()

Create the main part of the unit cell based on the sketch. See child classes for implementation.

abstract create_part_3dprint()

Create the part used for 3D printing based on the sketch. See child classes for implementation.

3.3 ABC for Auxetic Structures

class `pyauxetic.classes.auxetic_structure.AuxeticStructure(model, name, loading_params)`

Abstract base class for defining an auxetic structure. It defines the core behavior and must be subclassed for structures using different unit cells.

The following methods must be called in this exact order:

1. `add_unit_cells()`
2. `add_pattern_params()`
3. `assemble_structure()`
4. `define_step()`
5. `define_bcs()`
6. `mesh_part()`
7. `create_job()`
8. `submit_job()`
9. `output_results()`

__init__(*model, name, loading_params*)

Initialize the auxetic structure.

Child classes must define the following and class variables and then call this constructor:

- `pretty_name`
- `is_solid`
- `is_shell`
- `is_bulk`
- `is_planar`
- `is_tubular`
- `unit_cell_class`

Parameters

- **model** (*Model*) – Abaqus Model object in which the auxetic structure will be created.
- **name** (*str*) – Name of the structure. It will be used for all related files.

- **loading_params** (`LoadingParams`) – Special namedtuple describing the loading and boundary conditions applied to the model. Here, `loading_params.direction` is used for determining positioning of loading ribbons.

abstract assemble_structure(*for_3dprint=False, output_params=None, delete_all=True*)

Assemble one or more unit cells according to pattern parameters to create the auxetic structure. `add_pattern_params()` must be called before this.

This function must be defined by each child class of `AuxeticStructure`

Parameters

- **for_3dprint** (*bool*) – If `True`, the structure will be a 3D part suitable for export, Otherwise dimensionality will be governed by the structure. Defaults to `False`.
- **output_params** (`OutputParams`) – Special namedtuple describing the parameters for outputting the results of modeling and analysis. See class for full description of options. Here, `output_params.export_extrusion_depth` it is used for determining ribbon_extrusion_depth. If `for_3dprint` is `False`, this need not be passed. Defaults to `None`.
- **delete_all** (*bool*) – If `True`, all useless parts will be deleted. Defaults to `True`.

add_pattern_params(*pattern_params*)

Add the parameters used by `assemble_structure()` for assembling the structure.

Parameters

pattern_params (`PatternParams`) – Special namedtuple describing the parameters for patterning the unit cell(s). See class for full description of options.

add_unit_cells(*unit_cell_params*)

Add one or more unit cells to the auxetic structure.

Parameters

unit_cell_params – Special namedtuple describing the unit cell geometry. The namedtuple must be selected from `classes.auxetic_unit_cell_params` based on the type of structure. For nonuniform structures, must be a tuple where all unit cell ids used in `self.structure_map` are defined.

Raises

- **RuntimeError** – If `add_pattern_params()` has already been called.
- **ValueError** – If `unit_cell_params` is invalid.
- **ValueError** – If `unit_cell_params` contains repeated or non-positive ids.
- **ValueError** – If `unit_cell_params` contains more than one value for `extrusion_depth` or it's non-positive (planar structures only).

get_unit_cell_by_id(*id*)

Return a unit cell in the structure based on its id.

Parameters

id (*int*) – Unique numeric ID of the unit cell.

Returns

The unit cell whose id is specified.

Raises

ValueError – If the unit cell does not exist.

assign_material(*material_params*)

Assign material properties to the auxetic structure.

Note that *material_params.material_data* is not validated. Duck-Typing is used for calling the API and some errors are caught by the API itself, but ultimately any bad values are carried to the CAE model.

Parameters

material_params (**MaterialParams**) – Special namedtuple describing the material used for modeling and analysis. See class for full description of options.

Raises

- **ValueError** – If *material_params.hyperelastic* is invalid.
- **ValueError** – If *material_params.hyperelastic* is invalid.
- **AbaqusException** – Various exceptions raised by the Abaqus API.

define_step(*step_params=None*)

Define a single step for the analysis. Currently, only static general steps are supported. The step is named ‘Step-1’, but this name is not hard-coded elsewhere. Also, the nonlinear geometry (NLGEOM) parameter is always turned on.

Parameters

step_params (**StepParams**) – Special namedtuple describing the step defined for analysis. If not specified, the default values of the namedtuple are used. Also, validation of values is done by Abaqus API. See class for full description of options.

Raises

AbaqusException – Various exceptions raised by the Abaqus API.

define_bcs(*loading_params*)

Apply loads and boundary conditions (BCs) to the structure. This function must be called after *define_step()*. It then calls *_perpare_for_loading()* and afterwards defines the following BCs:

- An Encastre BC (fixed in all 6 directions) is applied from the initial step on the reference point *self.loading_rps[0]*, which is coupled to ‘LD-Edge-1’.
- A second load/BC is applied from the single loading step of the model based on on the reference point *self.loading_rps[1]*, which is coupled to ‘LD-Edge-2’. This load/BC is governed by *loading_params* and currently can be one of the following:
 - Uniaxial monotonic displacement BC.

Parameters

loading_params (**LoadingParams**) – Special namedtuple describing the loading and and boundary conditions applied to the model. See class for full description of options.

Raises

- **RuntimeError** – If the number of steps in the model is not exactly 2.
- **ValueError** – If *material_params.material_type* is invalid.
- **AbaqusException** – Various exceptions raised by the Abaqus API.

mesh_part(*mesh_params*)

Mesh the model.

The following functions are used in succession:

- **seedPart()**: *deviationFactor* and *minSizeFactor* are set to 0.1 and *constraint* is not supported.

- **setMeshControls()**: The following parameters are not supported: *technique*, *algorithm*, *minTransition*, *sizeGrowth*, *allowMapped*.
- **setElementType()**: The following constraints exist:
 - *region* is set to all faces/cells of the structure.
 - The *elemTypes* tuple is determined by *mesh_params.elem_code* and *mesh_params.elem_library*.

Parameters

mesh_params (**MeshParams**) – Special namedtuple describing the mesh applied to the model. See class for full description of options.

Raises

- **ValueError** – If *mesh_params.elem_shape* is invalid.
- **ValueError** – If *mesh_params.elem_shape* does not correspond to the structure’s dimensionality.
- **ValueError** – If *mesh_params.elem_library* is invalid.
- **ValueError** – If **mesh_params.elem_code* is invalid.
- **KeyError** – If *mesh_params.elem_code* does not correspond to a **SymbolicConstant**.
- **ValueError** – If *mesh_params.elem_code* does not correspond to an element.
- **AbaqusException** – Various exceptions raised by the Abaqus API.

create_job(*job_params*)

Define a single step for the analysis. Assigns *self.job*. Current limitations are:

- numDomains is set to numCpus.
- numGPUs is set to 0.
- User subroutine are not supported.

Parameters

job_params (**JobParams**) – Special namedtuple describing the job created for analysis. See class for full description of options.

Raises

AbaqusException – Various exceptions raised by the Abaqus API.

submit_job()

Submit the job and wait for it to finish. Does not clean old files, but they should not be a problem.

Raises

- **RuntimeError** – If the job has not been defined by [create_job\(\)](#).
- **RuntimeError** – If the job does not complete successfully.
- **AbaqusException** – Various exceptions raised by the Abaqus API.

output_results(*output_params*)

Output the results of the analysis.

Parameters

output_params (**OutputParams**) – Special namedtuple describing the parameters for outputting the results of modeling and analysis. See class for full description of options.

Raises

- **RuntimeError** – If the job has not been completed.
- **ValueError** – If `output_params.export_ribbon_width` has not been specified but STL or STP export is requested.
- **AbaqusException** – Various exceptions raised by the Abaqus API.

3.4 Unit Cell Parameters

This module contains instances of `namedtuple` that are used for defining the different unit cells that can be used for creating auxetic structures.

```
class pyauxetic.classes.auxetic_unit_cell_params.Reentrant2DUcpFull(id, extrusion_depth,
                                                                tail_strut_length,
                                                                tail_strut_thickness,
                                                                diag_strut_length,
                                                                diag_strut_thickness,
                                                                diag_strut_angle,
                                                                vert_strut_length,
                                                                vert_strut_thickness)
```

diag_strut_angle

Alias for field number 6

diag_strut_length

Alias for field number 4

diag_strut_thickness

Alias for field number 5

extrusion_depth

Alias for field number 1

```
formal_names = {'diag_strut_angle': 'Diagonal Strut Angle (deg)',
                'diag_strut_length': 'Diagonal Strut Length', 'diag_strut_thickness': 'Diagonal Strut Thickness',
                'extrusion_depth': 'Output Extrusion Depth', 'id': 'Unit Cell ID', 'tail_strut_length': 'Tail Strut Length',
                'tail_strut_thickness': 'Tail Strut Thickness', 'vert_strut_length': 'Vertical Strut Length', 'vert_strut_thickness': 'Vertical Strut Thickness'}
```

id

Alias for field number 0

tail_strut_length

Alias for field number 2

tail_strut_thickness

Alias for field number 3

```
unit_cell_type = 'Re-Entrant 2D'
```

vert_strut_length

Alias for field number 7

vert_strut_thickness

Alias for field number 8

```
class pyauxetic.classes.auxetic_unit_cell_params.Reentrant2DUcpBox(id, extrusion_depth,  
horz_bounding_box,  
vert_bounding_box,  
vert_strut_thickness,  
diag_strut_thickness,  
diag_strut_angle)
```

diag_strut_angle

Alias for field number 6

diag_strut_thickness

Alias for field number 5

extrusion_depth

Alias for field number 1

```
formal_names = {'diag_strut_angle': 'Diagonal Strut Angle (deg)',  
'diag_strut_thickness': 'Diagonal Strut Thickness', 'extrusion_depth': 'Output  
Extrusion Depth', 'horz_bounding_box': 'Vertical Bounding Box', 'id': 'Unit Cell  
ID', 'vert_bounding_box': 'Horizontal Bounding Box', 'vert_strut_thickness':  
'Vertical Strut Thickness'}
```

horz_bounding_box

Alias for field number 2

id

Alias for field number 0

```
unit_cell_type = 'Re-Entrant 2D'
```

vert_bounding_box

Alias for field number 3

vert_strut_thickness

Alias for field number 4

```
class pyauxetic.classes.auxetic_unit_cell_params.Reentrant2DUcpSimple(id, extrusion_depth,  
vert_strut_length,  
vert_strut_thickness,  
diag_strut_thickness,  
diag_strut_angle)
```

diag_strut_angle

Alias for field number 5

diag_strut_thickness

Alias for field number 4

extrusion_depth

Alias for field number 1

```
formal_names = {'diag_strut_angle': 'Diagonal Strut Angle (deg)',  
'diag_strut_thickness': 'Diagonal Strut Thickness', 'extrusion_depth': 'Output  
Extrusion Depth', 'id': 'Unit Cell ID', 'vert_strut_length': 'Vertical Strut  
Length', 'vert_strut_thickness': 'Vertical Strut Thickness'}
```

id

Alias for field number 0

unit_cell_type = 'Re-Entrant 2D'

vert_strut_length

Alias for field number 2

vert_strut_thickness

Alias for field number 3

3.5 Auxetic Structure Parameters

This module contains instances of `namedtuple` that are used for defining the different aspects of the structure.

```
class pyauxetic.classes.auxetic_structure_params.PatternParams(pattern_mode=None,
                                                             num_cell_repeat=None,
                                                             structure_map=None)
```

`namedtuple` instance describing the parameters for patterning the unit cell(s) that make up the structure.

num_cell_repeat

(`Tuple`) A `Tuple` of integers in the shape of (x,y) or (x,y,z) defining the number of times the unit cell is to be repeated in the x , y , and z directions.

Used only when `PatternParams.pattern_mode == 'uniform'`.

Defaults to `None`.

pattern_mode

(`str`) The type of patterning used for the structure:

- **'uniform'**: A singular unit cell is repeated based on `PatternParams.num_cell_repeat`.
- **'nonuniform'**: A number of unit cells are patterned based on `PatternParams.structure_map`.

Raises `ValueError` for other values. Defaults to `None`, which also raises the error.

structure_map

(`np.array`) A `numpy` array containing integer ids of unit cells and how they are distributed in the structure. The unit cells must be compatible for patterning.

Used only when `PatternParams.pattern_mode == 'nonuniform'`.

Defaults to `None`.

```
class pyauxetic.classes.auxetic_structure_params.MaterialParams(elastic=None, density=None,
                                                             hyperelastic=None)
```

`namedtuple` instance describing the material used for modeling and analysis. Care should be taken not to define contradicting properties.

density

(`Float`) Isotropic and temperature independent material Density.

Defaults to `None`, which does not define this property.

elastic

(Tuple) Isotropic and temperature independent elastic property. It should be a Tuple (E, ν) where E is Young's Modulus and ν is Poisson's Ratio.

Defaults to None, which does not define this property.

hyperelastic

(Tuple) Isotropic and temperature independent hyperelastic property. It should be a Tuple $(type, data)$ where $data$ is one of the following:

- **'ogden'**: The Ogden form of strain energy potential hyperelastic model. $data$ must be an iterable $((\sigma_0, \epsilon_0), (\sigma_1, \epsilon_1), \dots)$ where each pair (σ_i, ϵ_i) are a point in the isotropic uniaxial stress-strain test data.
- **'marlow'**: The Marlow form of strain energy potential hyperelastic model. $data$ is similar to the 'ogden' option.

Defaults to None, which does not define this property.

```
class pyauxetic.classes.auxetic_structure_params.StepParams(time_period=1, init_inc_size=0.1,  
                                                         min_inc_size=0.05,  
                                                         max_inc_size=0.1,  
                                                         max_num_inc=100)
```

namedtuple instance describing the step defined for analysis.

init_inc_size

(float) Initial increment size. Defaults to 0.1.

max_inc_size

(float) Maximum increment size. Defaults to 0.1.

max_num_inc

(float) Maximum number of increments. Defaults to 100.

min_inc_size

(float) Minimum increment size. Defaults to 0.05.

time_period

(float) Total time period of the step. Defaults to 1.

```
class pyauxetic.classes.auxetic_structure_params.LoadingParams(type=None, direction=None,  
                                                             data=None)
```

namedtuple instance describing the loading applied to the model.

data

The amount of loading applied to the model. See *loading_type* for format.

This variable is not validated, except for default Abaqus validations for each BC/Loading type. Define with caution.

Defaults to None.

direction

(str) Direction of loading applied to the model. Must be 'x' or 'y'. 'z' is currently not supported. Note that this also affects the positioning of the ribbons.

Raises `ValueError` for other values. Defaults to None, which also raises the error.

type

(str) The type of loading applied to the model. Valid values are:

- **'disp'**: Uniaxial monotonic displacement boundary condition. *loading_data* must be a float.
- **'force'**: Uniaxial monotonic concentrated force. *loading_data* must be a float.

Raises `ValueError` for other values. Defaults to `None`, which also raises the error.

```
class pyauxetic.classes.auxetic_structure_params.MeshParams(seed_size=None, elem_shape=None,
                                                         elem_code=None,
                                                         elem_library=None)
```

namedtuple instance describing the mesh applied to the model. See Abaqus documentation for definitions and discussions of each parameter's significance.

elem_code

(str) Element code used in the mesh. Values must be upper-case strings naming the element code, such as 'C3D10HS', 'CPE4H', or 'C3D8R'. Can also be a tuple of mentioned values for QUAD_DOMINATED or HEX_DOMINATED element shapes.

Specified element code(s) must be correct with respect to *MeshParams.elem_shape* and structure geometry.

Defaults to `None` which raises an error.

elem_library

(str) Element library used in the mesh. Must be the same as the analysis type defined in *StepParams*. Valid values are 'STANDARD' and 'EXPLICIT',

Defaults to `None` which raises an error.

elem_shape

(str) Shape of the elements used in the mesh. Valid values are 'QUAD', 'QUAD_DOMINATED', 'TRI', 'HEX', 'HEX_DOMINATED', 'TET', and 'WEDGE'.

Specified values must be correct with respect to *MeshParams.elem_code* and structure geometry. No validation is performed except for errors raised by Abaqus CAE or solver.

Defaults to `None` which raises an error.

seed_size

(float) Size of the seed used for mesh generation. Defaults to `None` which raises an error.

```
class pyauxetic.classes.auxetic_structure_params.JobParams(description="", numCpus=1,
                                                         memoryPercent=90,
                                                         explicitPrecision='single',
                                                         nodalOutputPrecision='single')
```

namedtuple instance describing the job created for analysis.

description

(str) Description of the job. Defaults to an empty string.

explicitPrecision

(str) Precision used for Abaqus/Explicit solver. Valid values are 'SINGLE' and 'DOUBLE'. Defaults to 'single'.

memoryPercent

(int) Amount of RAM in percent allocated to the analysis. Defaults to 90.

nodalOutputPrecision

(str) Nodal output precision. Valid values are 'SINGLE' and 'DOUBLE'. Defaults to 'single'.

numCpus

(int) Number of CPU cores used for the analysis. Defaults to 1.

```
class pyauxetic.classes.auxetic_structure_params.OutputParams(result_folder_name=None,
                                                             save_cae=True, save_odb=True,
                                                             save_job_files=True,
                                                             export_ribbon_width=None,
                                                             export_stl=False,
                                                             export_stp=False)
```

namedtuple instance describing the parameters for outputting the results of modeling and analysis.

export_ribbon_width

Float defining the ribbon width used for exporting the part. Must be positive, but can be None if the part will not be exported (both export_stl are export_stp are False).

Defaults to None.

export_stl

Whether or not to export the structure in the STL format. Defaults to False.

export_stp

Whether or not to export the structure in the STP format. Defaults to False.

result_folder_name

Path to the folder where the requested results are to be stored. Everything else is left at the working folder. If set to None, a suitable name is selected using #TODO.

Defaults to None.

save_cae

Whether or not to save the model database (.cae file). Defaults to True.

save_job_files

Whether or not to save the miscellaneous job files (inp, msg, log, and sta files). Defaults to True.

save_odb

Whether or not to save the output database (.odb file). Defaults to True.

3.6 Classes Based on Different Unit Cells

These entries explain the classes that are based on different unit cells. Each entry contains a unit cell class and a number of structure classes.

3.6.1 Reentrant-2D Unit Cell

```
class pyauxetic.classes.reentrant2d.Reentrant2DUnitCell(model, params)
```

Bases: *AuxeticUnitCell*

Class defining a 2D Re-Entrant unit cell.

```
params_class_list = (<class
'pyauxetic.classes.auxetic_unit_cell_params.Reentrant2DUcpFull'>, <class
'pyauxetic.classes.auxetic_unit_cell_params.Reentrant2DUcpBox'>, <class
'pyauxetic.classes.auxetic_unit_cell_params.Reentrant2DUcpSimple'>)
```

`__init__(model, params)`

Initialize the object with the given parameters.

Calls `auxetic_unit_cell.AuxeticUnitCell.__init__()` which creates `self.sketch` and `self.part_main`.

Parameters

- **model** (*Model*) – Abaqus Model object in which the unit cell will be created.
- **params** – Parameters describing the unit cell geometry. It must be from a suitable class based on the list defined in `auxetic_unit_cell_params.reentrant2d_ucp_list`.

`create_part_main()`

Create the main part of the unit cell based on the sketch.

`create_part_3dprint()`

Create the part used for 3D printing based on the sketch.

`create_sketch()`

Create the 2D sketch for the unit cell. A suitable creation method is called based on the unit cell parameters class passed to the unit cell object.

`class pyauxetic.classes.reentrant2d.Reentrant2DPlanarShellStructure(model, name, loading_params)`

Bases: *AuxeticStructure*

Class defining an auxetic structure based on the reentrant unit cell.

`pretty_name = 'Planar Shell Re-Entrant 2D'`

`is_solid = False`

`is_shell = True`

`is_bulk = False`

`is_planar = True`

`is_tubular = False`

`unit_cell_class`

alias of *Reentrant2DUnitCell*

`assemble_core_structure(structure_map=None, for_3dprint=False, delete_all=True)`

Pattern the unit cells to form the core auxetic structure.

It is called by `assemble_structure()`, which is responsible for validating all input.

Parameters

- **structure_map** (`np.array`) – A numpy array containing integer ids of unit cells and how they are distributed in the structure. It is defined by `assemble_structure()` regardless of its `pattern_params` argument.
- **for_3dprint** (*bool*) – If True, the structure will be a 3D part suitable for export, Otherwise dimensionality will be governed by the structure. Defaults to False.
- **delete_all** (*bool*) – If True, all useless parts will be deleted. Defaults to True.

Returns

A tuple containing the created core auxetic structure part and its instance.

Raises

AbaqusException – Various exceptions raised by the Abaqus API. Sometimes exceptions will be fatal.

assemble_structure(*for_3dprint=False, output_params=None, delete_all=True*)

Assemble one or more unit cells according to pattern parameters to create the auxetic structure. *add_pattern_params()* must be called before this.

This is the implementation of *auxetic_structure.AuxeticStructure.assemble_structure()* for this class.

Parameters

- **for_3dprint** (*bool*) – If *True*, the structure will be a 3D part suitable for export, Otherwise dimensionality will be governed by the structure. Defaults to *False*.
- **output_params** (*OutputParams*) – Special namedtuple describing the parameters for outputting the results of modeling and analysis. See class for full description of options. Here, *output_params.export_ribbon_width* it is used for determining width of the ribbon. If *for_3dprint* is *False*, this need not be passed. Defaults to *None*.
- **delete_all** (*bool*) – If *True*, all useless parts will be deleted. Defaults to *True*.

`pyauxetic.classes.reentrant2d.create_sketch_reentrant2d_full(model, params, sketch_name)`

Create the 2D sketch of a reentrant2d unit cell using the full set of unit cell parameters.

Parameters

- **model** (*Model*) – Abaqus Model object in which the unit cell will be created.
- **params** (*Reentrant2DUcpFull*) – Parameters describing the unit cell geometry. See #TODO for details.
- **sketch_name** (*str*) – Name assigned to the sketch.

`pyauxetic.classes.reentrant2d.create_sketch_reentrant2d_box(model, params, sketch_name)`

Create the 2D sketch of a reentrant2d unit cell using the ‘bounding box’ set of unit cell parameters.

Parameters

- **model** (*Model*) – Abaqus Model object in which the unit cell will be created.
- **params** (*Reentrant2DUcpBox*) – Parameters describing the unit cell geometry. See #TODO for details.
- **sketch_name** (*str*) – Name assigned to the sketch.

`pyauxetic.classes.reentrant2d.create_sketch_reentrant2d_simple(model, params, sketch_name)`

Create the 2D sketch of a reentrant2d unit cell using the simplified set of unit cell parameters.

Parameters

- **model** (*Model*) – Abaqus Model object in which the unit cell will be created.
- **params** (*Reentrant2DUcpSimple*) – Parameters describing the unit cell geometry. See #TODO for details.
- **sketch_name** (*str*) – Name assigned to the sketch.

3.7 Helper Functions

Helper functions used in the PyAuxetic library for various operations.

`pyauxetic.helper.create_ribbon_part(model, length_x, length_y, is3d, extrusion_depth)`

Create a rectangular ribbon part.

Parameters

- **model** (*Model*) – Model object in which the part will be created.
- **length_x** (*float*) – Length of the part in the x (1st) direction
- **length_y** (*float*) – Length of the part in the y (2nd) direction
- **is3d** (*bool*) – If True, the part is extruded.
- **extrusion_depth** (*float*) – Extrusion depth used if *is3d* is True.

Returns

The created part object.

`pyauxetic.helper.draw_line(sketch, point1, point2)`

Draw a line using two points and return the second point.

Output of this function is intended to be used as point1 for future uses.

Parameters

- **sketch** (*ConstrainedSketch*) – The sketch in which the line is drawn.
- **point1** (*tuple*) – 2D Cartesian coordinates of the first point.
- **point2** (*tuple*) – 2D Cartesian coordinates of the second point.

Returns

A tuple in the form of $((x,y), lineObj)$ containing 2D Cartesian coordinates of the second point and the created line object.

`pyauxetic.helper.find_edges_from_coords(part, coord, value)`

Find edges of a part that exist on a certain value along a given coordinate axis.

Parameters

- **part** (*Part*) – The part in question.
- **coord** (*int*) – The coordinate axis used for the operation. Valid values are 1, 2, or 3.
- **value** (*float*) – Value of the coordinate specified in *coord*.

Returns

An EdgeArray object containing one or more edges which have a point on the given coordinates.

Raises

RuntimeError – If no edges are found.

`pyauxetic.helper.find_vertices_from_coords(part, coord, value)`

Find vertices of a part that exist on a certain value along a given coordinate axis.

Parameters

- **part** (*Part*) – The part in question.
- **coord** (*int*) – The coordinate axis used for the operation. Valid values are 1, 2, or 3.
- **value** (*float*) – Value of the coordinate specified in *coord*.

Returns

A VertexArray object containing one or more vertices which have a point on the given coordinates.

Raises

RuntimeError – If no vertices are found.

`pyauxetic.helper.find_vertices_from_coords_minmax(part, coord, value)`

Find the first and last vertices of a part that exist on a certain value along a given coordinate axis.

Parameters

- **part** (*Part*) – The part in question.
- **coord** (*int*) – The coordinate axis used for the operation. Valid values are 1, 2, or 3.
- **value** (*float*) – Value of the coordinate specified in *coord*.

Returns

A Tuple of two VertexArray objects for the first and last vertices.

Raises

RuntimeError – If no vertices are found.

`pyauxetic.helper.find_regular_geometries(sketch)`

Searches the sketch and returns a list of *REGULAR* geometries.

Geometries in an Abaqus sketch are either *REGULAR* or *CONSTRUCTION*, the latter of which is used for defining relationships. This function is intended to filter out construction lines defined for mirroring parts.

Parameters

sketch (*ConstrainedSketch*) – The sketch in which the line is drawn.

Returns

A list of ConstrainedSketchGeometry objects which are *REGULAR*.

`pyauxetic.helper.get_part_box_size(part)`

Calculates size of a part's rectangular boundary.

Parameters

part (*Part*) – The part which is queried.

Returns

A tuple in the form of (x,y,z) containing size of the part's rectangular boundary in the Cartesian coordinate system.

`pyauxetic.helper.get_box_coords(object_list)`

Find the minimum and maximum Cartesian coordinates for a Part or PartInstance.

Parameters

object_list (*Part/PartInstance/Repository/tuple*) – The part(s) or instance(s) which are queried.

Returns

A tuple of tuples in the form of ((min_x, min_y, min_z), (max_x, max_y, max_z)) the minimum and maximum Cartesian coordinates of the parts.

`pyauxetic.helper.return_results_folder_path(structure_name, root_folder_name=None)`

Return a unified path for storing results of analysis of a structure.

Parameters

- **structure_name** (*str*) – Name of the structure for which the folder is created.

- **root_folder_name** (*str*) – Name for the root folder. Defaults to None.

Returns

Absolute path for the results folder.

`pyauxetic.helper.return_sketch_name(base_name)`

Return a unified name for a sketch based on a base name.

Parameters

base_name (*str*) – Base name to which a suffix is added.

Returns

A suitable name for an Abaqus sketch.

`pyauxetic.helper.return_unit_cell_name_main(base_name)`

Return the name of a main part based on a unit cell.

Parameters

base_name (*str*) – Base name to which a suffix is added.

Returns

A suitable name for an Abaqus part.

`pyauxetic.helper.return_unit_cell_name_3dprint(base_name)`

Return the name of a 3D printing part based on a unit cell. Use this function only for naming parts that are used for 3D printing.

Parameters

base_name (*str*) – Base name to which a suffix is added.

Returns

A suitable name for an Abaqus part.

`pyauxetic.helper.return_instance_name(base_name, suffix='')`

Return the name of an instance based on a unit cell.

Parameters

base_name (*str*) – Base name to which a suffix is added. Defaults to an empty string.

Returns

A suitable name for an Abaqus instance.

`pyauxetic.helper.transfer_instance_to_zero(model, instance)`

Transfer a PartInstance so it's vertex with minimum coordinates is at global (0,0,0).

Parameters

- **model** (*Model*) – Model object in which the instance is defined.
- **instance** (*PartInstance*) – The instance to be moved.

CONTRIBUTE TO THE SOFTWARE

We welcome all contributions. You can help in the following ways:

- **Testing and Bug Reports:** We always appreciate testing various features and reporting any problems. You can use our GitHub issue tracker for bug reports.
- **Example Problems:** While we provide entries for example problems, not all of them have been tested experimentally. We always appreciate users testing the concepts and structures in the real world. We will, of course, give appropriate citations when applicable.
- **Documentation:** The software always needs more documentaion. We use Sphinx which is very straightforward.
- **New Features:** We appreciate implementation of new features. There are a few ways to go about this:
 - If you can add features using the object-oriented approach, submit a pull request and we will review your code.
 - If you can write the code using Abqus' Python API but would rather not bother with the object-oriented framework, you can send us scripts and we may be able to add them to the software. Make sure to say this in your feature request.
 - If you have a new concept that you think can add value to the scientific community, send the maintainer of the repository (M. Khoshbin) a private message on GitHub. We may be able to collaborate in a scientific framework.

LICENSING

Use of this software is licensed under *GNU Affero General Public License v3.0 or later* standard software license. You can find the latest version of this license in the [GNU website](#).

THE PYAUXETIC TEAM

This software is a result of a collaboration between academics and engineers with vastly different skill sets. We believe in giving credit where it's due, and as such maintain the following list of contributors. If you believe that we are missing something, let us know and we will update the list.

6.1 Main Team

Javad Kadkhodapour, PhD: Project Lead. Developed concepts and coordinates research and programming efforts.

Mohammadreza Khoshbin, PhDc: Software Developer and Maintainer. Developed concepts, created the software and maintains the software and documentation.

6.2 Contributors

Ali Pourkamali Anaraki, PhD: Provided the experimental and computational facilities.

Hossein Dibajian, PhD: Developed concepts and wrote the initial version of the software named *Auextic2D*.

Alireza Sangsefidi, PhDc: Performed experiments for example problems and tested the software.

Shane O' Sullivan (University of Limerick, Ireland): Tested the GUI and reported multiple bugs.

PYTHON MODULE INDEX

p

`pyauxetic.classes.auxetic_structure`, 29
`pyauxetic.classes.auxetic_structure_params`,
35
`pyauxetic.classes.auxetic_unit_cell`, 28
`pyauxetic.classes.auxetic_unit_cell_params`,
33
`pyauxetic.classes.reentrant2d`, 38
`pyauxetic.helper`, 41
`pyauxetic.main`, 27

Symbols

`__init__()` (*pyauxetic.classes.auxetic_structure.AuxeticStructure* method), 29

`__init__()` (*pyauxetic.classes.auxetic_unit_cell.AuxeticUnitCell* method), 28

`__init__()` (*pyauxetic.classes.reentrant2d.Reentrant2DUnitCell* method), 38

A

`add_pattern_params()` (*pyauxetic.classes.auxetic_structure.AuxeticStructure* method), 30

`add_unit_cells()` (*pyauxetic.classes.auxetic_structure.AuxeticStructure* method), 30

`assemble_core_structure()` (*pyauxetic.classes.reentrant2d.Reentrant2DPlanarShellStructure* method), 39

`assemble_structure()` (*pyauxetic.classes.auxetic_structure.AuxeticStructure* method), 30

`assemble_structure()` (*pyauxetic.classes.reentrant2d.Reentrant2DPlanarShellStructure* method), 40

`assign_material()` (*pyauxetic.classes.auxetic_structure.AuxeticStructure* method), 30

`AuxeticStructure` (class in *pyauxetic.classes.auxetic_structure*), 29

`AuxeticUnitCell` (class in *pyauxetic.classes.auxetic_unit_cell*), 28

C

`create_job()` (*pyauxetic.classes.auxetic_structure.AuxeticStructure* method), 32

`create_part_3dprint()` (*pyauxetic.classes.auxetic_unit_cell.AuxeticUnitCell* method), 29

`create_part_3dprint()` (*pyauxetic.classes.reentrant2d.Reentrant2DUnitCell* method), 39

`create_part_main()` (*pyauxetic.classes.auxetic_unit_cell.AuxeticUnitCell* method), 29

`create_part_main()` (*pyauxetic.classes.reentrant2d.Reentrant2DUnitCell* method), 39

`create_ribbon_part()` (in module *pyauxetic.helper*), 41

`create_sketch()` (*pyauxetic.classes.auxetic_unit_cell.AuxeticUnitCell* method), 29

`create_sketch()` (*pyauxetic.classes.reentrant2d.Reentrant2DUnitCell* method), 39

`create_sketch_reentrant2d_box()` (in module *pyauxetic.classes.reentrant2d*), 40

`create_sketch_reentrant2d_full()` (in module *pyauxetic.classes.reentrant2d*), 40

`create_sketch_reentrant2d_simple()` (in module *pyauxetic.classes.reentrant2d*), 40

D

`data` (*pyauxetic.classes.auxetic_structure_params.LoadingParams* attribute), 36

`define_bcs()` (*pyauxetic.classes.auxetic_structure.AuxeticStructure* method), 31

`define_step()` (*pyauxetic.classes.auxetic_structure.AuxeticStructure* method), 31

`density` (*pyauxetic.classes.auxetic_structure_params.MaterialParams* attribute), 35

`description` (*pyauxetic.classes.auxetic_structure_params.JobParams* attribute), 37

`diag_strut_angle` (*pyauxetic.classes.auxetic_unit_cell_params.Reentrant2DUcpBox* attribute), 34

`diag_strut_angle` (*pyauxetic.classes.auxetic_unit_cell_params.Reentrant2DUcpFull* attribute), 33

`diag_strut_angle` (*pyauxetic.classes.auxetic_unit_cell_params.Reentrant2DUcpSimple* attribute), 33

attribute), 34
diag_strut_length (*pyauxetic.classes.auxetic_unit_cell_params.Reentrant2DUcpFull* *attribute*), 33
diag_strut_thickness (*pyauxetic.classes.auxetic_unit_cell_params.Reentrant2DUcpBox* *attribute*), 34
diag_strut_thickness (*pyauxetic.classes.auxetic_unit_cell_params.Reentrant2DUcpFull* *attribute*), 34
diag_strut_thickness (*pyauxetic.classes.auxetic_unit_cell_params.Reentrant2DUcpSimple* *attribute*), 33
diag_strut_thickness (*pyauxetic.classes.auxetic_unit_cell_params.Reentrant2DUcpSimple* *attribute*), 34
direction (*pyauxetic.classes.auxetic_structure_params.LoggingParams* *attribute*), 36
draw_line() (*in module pyauxetic.helper*), 41
E
elastic (*pyauxetic.classes.auxetic_structure_params.MaterialParams* *attribute*), 35
elem_code (*pyauxetic.classes.auxetic_structure_params.MeshParams* *attribute*), 37
elem_library (*pyauxetic.classes.auxetic_structure_params.MeshParams* *attribute*), 37
elem_shape (*pyauxetic.classes.auxetic_structure_params.MeshParams* *attribute*), 37
explicitPrecision (*pyauxetic.classes.auxetic_structure_params.JobParams* *attribute*), 37
export_ribbon_width (*pyauxetic.classes.auxetic_structure_params.OutputParams* *attribute*), 38
export_stl (*pyauxetic.classes.auxetic_structure_params.OutputParams* *attribute*), 38
export_stp (*pyauxetic.classes.auxetic_structure_params.OutputParams* *attribute*), 38
extrusion_depth (*pyauxetic.classes.auxetic_unit_cell_params.Reentrant2DUcpBox* *attribute*), 34
extrusion_depth (*pyauxetic.classes.auxetic_unit_cell_params.Reentrant2DUcpFull* *attribute*), 33
extrusion_depth (*pyauxetic.classes.auxetic_unit_cell_params.Reentrant2DUcpSimple* *attribute*), 34
F
find_edges_from_coords() (*in module pyauxetic.helper*), 41
find_regular_geometries() (*in module pyauxetic.helper*), 42
find_vertices_from_coords() (*in module pyauxetic.helper*), 41
find_vertices_from_coords_minmax() (*in module pyauxetic.helper*), 42
format_names (*pyauxetic.classes.auxetic_unit_cell_params.Reentrant2DUcpBox* *attribute*), 34
format_names (*pyauxetic.classes.auxetic_unit_cell_params.Reentrant2DUcpFull* *attribute*), 34
format_names (*pyauxetic.classes.auxetic_unit_cell_params.Reentrant2DUcpSimple* *attribute*), 33
G
get_box_size() (*in module pyauxetic.helper*), 42
get_part_box_size() (*in module pyauxetic.helper*), 42
get_unit_cell_by_id() (*pyauxetic.classes.auxetic_structure.AuxeticStructure* *method*), 30
H
is_bounding_box (*pyauxetic.classes.auxetic_unit_cell_params.Reentrant2DUcpBox* *attribute*), 34
hyperelastic (*pyauxetic.classes.auxetic_structure_params.MaterialParams* *attribute*), 36
I
id (*pyauxetic.classes.auxetic_unit_cell_params.Reentrant2DUcpBox* *attribute*), 34
id (*pyauxetic.classes.auxetic_unit_cell_params.Reentrant2DUcpFull* *attribute*), 33
id (*pyauxetic.classes.auxetic_unit_cell_params.Reentrant2DUcpSimple* *attribute*), 35
init_inc_size (*pyauxetic.classes.auxetic_structure_params.StepParams* *attribute*), 36
is_bulk (*pyauxetic.classes.reentrant2d.Reentrant2DPlanarShellStructure* *attribute*), 39
is_planar (*pyauxetic.classes.reentrant2d.Reentrant2DPlanarShellStructure* *attribute*), 39
is_shell (*pyauxetic.classes.reentrant2d.Reentrant2DPlanarShellStructure* *attribute*), 39
is_solid (*pyauxetic.classes.reentrant2d.Reentrant2DPlanarShellStructure* *attribute*), 39
is_tubular (*pyauxetic.classes.reentrant2d.Reentrant2DPlanarShellStructure* *attribute*), 39
J
JobParams (*class in pyauxetic.classes.auxetic_structure_params*), 37

L

LoadingParams (class in *pyauxetic.classes.auxetic_structure_params*), 36

M

main_batch() (in module *pyauxetic.main*), 28

main_gui_proxy() (in module *pyauxetic.main*), 28

main_single() (in module *pyauxetic.main*), 27

MaterialParams (class in *pyauxetic.classes.auxetic_structure_params*), 35

max_inc_size (pyauxetic.classes.auxetic_structure_params.StepParams attribute), 36

max_num_inc (pyauxetic.classes.auxetic_structure_params.StepParams attribute), 36

memoryPercent (pyauxetic.classes.auxetic_structure_params.JobParams attribute), 37

mesh_part() (pyauxetic.classes.auxetic_structure.AuxeticStructure method), 31

MeshParams (class in *pyauxetic.classes.auxetic_structure_params*), 37

min_inc_size (pyauxetic.classes.auxetic_structure_params.StepParams attribute), 36

module

pyauxetic.classes.auxetic_structure, 29

pyauxetic.classes.auxetic_structure_params, 35

pyauxetic.classes.auxetic_unit_cell, 28

pyauxetic.classes.auxetic_unit_cell_params, 33

pyauxetic.classes.reentrant2d, 38

pyauxetic.helper, 41

pyauxetic.main, 27

N

nodalOutputPrecision (pyauxetic.classes.auxetic_structure_params.JobParams attribute), 37

num_cell_repeat (pyauxetic.classes.auxetic_structure_params.PatternParams attribute), 35

numCpus (pyauxetic.classes.auxetic_structure_params.JobParams attribute), 37

O

output_results() (pyauxetic.classes.auxetic_structure.AuxeticStructure method), 32

OutputParams (class in *pyauxetic.classes.auxetic_structure_params*), 38

P

params_class_list (pyauxetic.classes.reentrant2d.Reentrant2DUnitCell attribute), 38

part_3dprint (pyauxetic.classes.auxetic_unit_cell.AuxeticUnitCell property), 28

part_main (pyauxetic.classes.auxetic_unit_cell.AuxeticUnitCell property), 28

pattern_mode (pyauxetic.classes.auxetic_structure_params.PatternParams attribute), 35

PatternParams (class in *pyauxetic.classes.auxetic_structure_params*), 35

pretty_name (pyauxetic.classes.reentrant2d.Reentrant2DPlanarShellStructure attribute), 39

pyauxetic.classes.auxetic_structure module, 29

pyauxetic.classes.auxetic_structure_params module, 35

pyauxetic.classes.auxetic_unit_cell module, 28

pyauxetic.classes.auxetic_unit_cell_params module, 33

pyauxetic.classes.reentrant2d module, 38

pyauxetic.helper module, 41

pyauxetic.main module, 27

R

Reentrant2DPlanarShellStructure (class in *pyauxetic.classes.reentrant2d*), 39

Reentrant2DUcpBox (class in *pyauxetic.classes.auxetic_unit_cell_params*), 34

Reentrant2DUcpFull (class in *pyauxetic.classes.auxetic_unit_cell_params*), 33

Reentrant2DUcpSimple (class in *pyauxetic.classes.auxetic_unit_cell_params*), 34

Reentrant2DUnitCell (class in *pyauxetic.classes.reentrant2d*), 38

result_folder_name (pyauxetic.classes.auxetic_structure_params.OutputParams attribute), 38

return_instance_name() (in module *pyauxetic.helper*), 43

return_results_folder_path() (in module *pyauxetic.helper*), 42

return_sketch_name() (in module *pyauxetic.helper*), 43

return_unit_cell_name_3dprint() (in module *pyauxetic.helper*), 43

return_unit_cell_name_main() (in module *pyaux-etic.helper*), 43

S

save_cae (*pyauxetic.classes.auxetic_structure_params.OutputParams* attribute), 38

save_job_files (*pyaux-etic.classes.auxetic_structure_params.OutputParams* attribute), 38

save_odb (*pyauxetic.classes.auxetic_structure_params.OutputParams* attribute), 38

seed_size (*pyauxetic.classes.auxetic_structure_params.MeshParams* attribute), 37

StepParams (class in *pyaux-etic.classes.auxetic_structure_params*), 36

structure_map (*pyaux-etic.classes.auxetic_structure_params.PatternParams* attribute), 35

submit_job() (*pyaux-etic.classes.auxetic_structure.AuxeticStructure* method), 32

vert_strut_length (*pyaux-etic.classes.auxetic_unit_cell_params.Reentrant2DUcpFull* attribute), 33

vert_strut_length (*pyaux-etic.classes.auxetic_unit_cell_params.Reentrant2DUcpSimple* attribute), 35

vert_strut_thickness (*pyaux-etic.classes.auxetic_unit_cell_params.Reentrant2DUcpBox* attribute), 34

vert_strut_thickness (*pyaux-etic.classes.auxetic_unit_cell_params.Reentrant2DUcpFull* attribute), 33

vert_strut_thickness (*pyaux-etic.classes.auxetic_unit_cell_params.Reentrant2DUcpSimple* attribute), 35

T

tail_strut_length (*pyaux-etic.classes.auxetic_unit_cell_params.Reentrant2DUcpFull* attribute), 33

tail_strut_thickness (*pyaux-etic.classes.auxetic_unit_cell_params.Reentrant2DUcpFull* attribute), 33

time_period (*pyauxetic.classes.auxetic_structure_params.StepParams* attribute), 36

transfer_instance_to_zero() (in module *pyaux-etic.helper*), 43

type (*pyauxetic.classes.auxetic_structure_params.LoadingParams* attribute), 36

U

unit_cell_class (*pyaux-etic.classes.reentrant2d.Reentrant2DPlanarShellStructure* attribute), 39

unit_cell_type (*pyaux-etic.classes.auxetic_unit_cell_params.Reentrant2DUcpBox* attribute), 34

unit_cell_type (*pyaux-etic.classes.auxetic_unit_cell_params.Reentrant2DUcpFull* attribute), 33

unit_cell_type (*pyaux-etic.classes.auxetic_unit_cell_params.Reentrant2DUcpSimple* attribute), 35

V

vert_bounding_box (*pyaux-etic.classes.auxetic_unit_cell_params.Reentrant2DUcpBox* attribute), 34