Phon Documentation

Release 0.4

Kristoffer Carlsson

October 15, 2016

Contents

Phon is a Python library which has the following features:

- Parse meshes created in Abaqus' .inp and Gmsh format and stores it into an internal class based representation. Element and node sets are also parsed and stored.
- Can out of the box insert interface elements between grains in both 2D and 3D meshes generated with Neper.
- Can export back the mesh for analysis to either Abaqus .inp or OOFEM's .in format.

The most common way Phon is used right now is to generate a mesh with Neper, then using Phon the mesh is parsed, interface elements are included and the resulting mesh is then exported for FE-analysis.

This is the central page for all of Phon's documentation.

CHAPTER 1

Contents

1.1 Installation

Phon can be installed on most computers with a standard Python installation. [Numpy](http://www.numpy.org/) is used internally so this needs to be available to the Python installation.

Phon supports Python 2.7, 2.2, 3.3 and 3.4.

1.1.1 Git

The easiest way to get the latest version is by git. To download the repository the following command is used:

```
$ git clone https://github.com/KristofferC/Phon.git
```

This will give you a folder called "Phon". Navigate to this folder and execute the following:

```
$ python setup.py install
```

This should install the package to the system. To see if the installation is successful try to import the package by executing:

```
>>> import phon
```

in the Python terminal window.

To update to the latest version, go into your repository and execute:

```
$ git pull origin master
```

and rerun the setup file.

1.1.2 Zip and Tar

You can also get the latest version compressed as a .zip or .tar from the projects homepage: http://kristofferc.github.io/Phon/

1.2 Usage

Here the functionality of Phon and how to use it is shown.

1.2.1 Reading and querying mesh

Reading a mesh is in Phon the process of parsing a file containing nodes, elements, and eventual sets (element sets and node sets). This information is then stored in a class based format in Phon which is easy to query and modify the mesh.

It is currently possible to read a mesh exported to Abaqus' .inp format or Gmsh's .msh format.

Read an Abaqus mesh

The function to read an Abaqus mesh is `phon.io.read.read_from_abaqus_inp.read_from_abaqus_inp()`.

To read the mesh, the function is first imported and then used like this:

```
>>> from phon.io_tools.read.read_from_abaqus_inp import read_from_abaqus_inp
>>> mesh = read_from_abaqus_inp("n10-id1.inp")
```

The mesh is now stored as an instance of the `phon.mesh_objects.mesh()` class.

An example Abaqus mesh file can be found here.

Gmsh

The function to read a Gmsh mesh is `phon.io_tools.read.read_from_gmsh.read_from_gmsh()`.

It is currently used a bit differently than the function to read from Abaqus meshes. The gmsh reader assumes that each grain has a separate mesh file called *fileX.msh* where *file* is the basename and *X* is the grain id. The gmsh reader can then be called as:

```
>>> from phon.io_tools.read.read_from_gmsh import read_from_gmsh
>>> basename = "gmsh_grain_file"
>>> n_grains = 3
>>> mesh = read_from_gmsh(base, n_grains)
```

An example of a microstructure with 3 grains split over 3 .msh files can be seen here.

Querying and modifying the mesh

We can now query the mesh object for information about the mesh and modify it.

A few example follows here:

- The number of elements in the mesh:

```
>>> print(len(mesh.elements))
1683
```

- The number of nodes in the mesh:

```
>>> print(len(mesh.nodes))
289
```

- A node set with a certain name:

```
>>> print(mesh.node_sets["x0y0"])
Node set with name x0y0 containing nodes with the following ids [1, 3, 34, 39, 40, 131, 132]
```

- Information about a specific node:

```
>>> print(mesh.nodes[10])
Node located at (0.722422, 0.420036, 0.196344)
```

- Changing a nodes location:

```
>>> mesh.nodes[10].c[0] = 1337.0
>>> print(mesh.nodes[10])
Node located at (1337.000000, 0.420036, 0.196344):
```

- Information about a specific element:

```
>>> print(mesh.elements[701])
Element of type C3D4 containing the vertices with id [168, 150, 154, 60].
```

For more information of the different properties to query from the mesh look at the documentation of `phon.mesh_objects`.

1.2.2 Insert interface elements

Interface elements are elements modelling an interface law between for example grains in a model of a polycrystalline material. Phon has support for inserting these elements into both a 2D mesh and 3D mesh. This is done with the function `phon.mesh_tools.create_cohesive_elements.create_cohesive_elements()`.

The function creates the interface element and also generates one element set for each grain boundary. These element sets have the name `cohesX_Y` where X and Y are the grain ids to the two grains that share the grain boundary.

The function requires some element sets with specific names to exist.

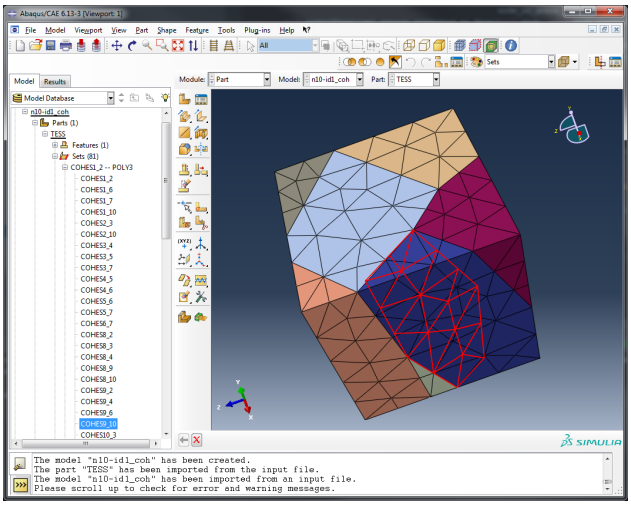
- In a mesh each grain should have its own element set with the bulk elements. These should for a 3D mesh be called *POLYx* and for a 2D mesh be called *FACEx* where *x* is the grain id.
- Each grain boundary should have its own element with the elements in the grain boundary. These elements will have a dimension one lower than the one of the bulk elements. The name for these sets should for a 3D mesh be called *FACEx* and for a 2D mesh be called *EDGEx*.

These names have been chosen so that a mesh files generated with Neper can directly be used. Note that you might need the `-dim all` flag set on the mesher in Neper for all the elements and element sets to be exported correctly. If you want to insert interface elements in a mesh generated with some other software you need to make sure the output matches the above.

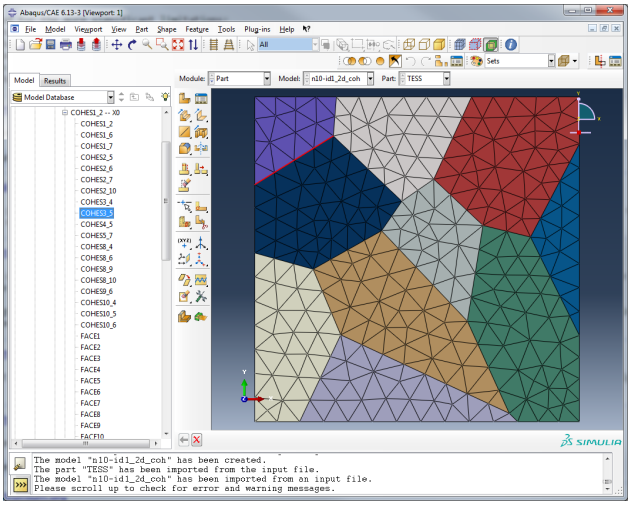
An example of a 3D mesh being read, cohesive elements inserted into it and then getting exported is shown below.:

```
>>> from phon.io_tools.read.read_from_abaqus_inp import read_from_abaqus_inp
>>> from phon.mesh_tools.create_cohesive_elements import create_cohesive_elements
>>> from phon.io_tools.write.export_to_abaqus import export_to_abaqus
>>> mesh = read_from_abaqus_inp("n10-id1.inp")
>>> create_cohesive_elements(mesh, mesh_dimension=3)
>>> export_to_abaqus("cohesive_file.inp", mesh, write_2d_elements=False)
```

Opening the generated file in Abaqus shows the generated mesh. An element sets with the cohesive elements is highlighted.



For completeness sake, an image of cohesive elements in 2D is shown below:



1.2.3 Exporting a mesh

The mesh can currently be exported to a format readable by either Abaqus or OOFEM.

One of the arguments to the export functions is if 2D elements should be exported. The reason for this is that currently the elements in the grain boundary are not deleted after insertion of interface elements. When xporting a 3D mesh we might want to ignore the left over 2D elements.

Exporting to Abaqus

Assume that we have a mesh stored in the `mesh` variable. If we want to export this to abaqus and ignore the 2D elements in the mesh it would be done like this:

```
>>> from phon.io_tools.write.export_to_abaqus import export_to_abaqus
>>> export_to_abaqus("output_file.inp", mesh, write_2d_elements=False)
```

This file can then be imported into Abaqus CAE. If you are doing 2D analysis you would want to set `write_2d_elements=True`.

Exporting to OOFEM

A mesh object can be exported to OOFEM as follows:

```
>>> from phon.io_tools.write.export_to_oofem import export_to_oofem
>>> export_to_oofem("output_file.inp", mesh, write_2d_elements=False)
```

The generated file only contains the parts of the file relevant to the mesh. The rest of the analysis records must be manually added in order to perform analyses.

CHAPTER 2

Indices and tables

- genindex
- modindex

2.1 Information

Author Kristoffer Carlsson - kristoffer.carlsson@chalmers.se

Version 0.4

License MIT License

Source https://github.com/KristofferC/phon