## Contents

1 Parameter Files 3  
  1.1 params.py .......................................................... 3  
  1.2 var.py .................................................................. 3  
  1.3 coeffs.py ............................................................. 4  

2 Preprocessing 5  

3 Interpolation 7  

4 Generate_model.py 9  

5 Postprocessing 11  

6 Mesh Snap 13  

7 Miscellaneous 15  

Python Module Index 17
This is the documentation for our work in the optimization of inserts in sandwich structures.

The project can be found on GitHub here <https://github.com/JensPfeifle/Insert-Optimization>

Contributors:

- Jens Pfeifle
- Lukas Kalbhenn
- Jérôme Klipfel

Contents:

```python
from simulation import Manager, Job
from misc import load_json_lines, clean_up

import params

if __name__ == '__main__':
    inputlines = load_json_lines(params.WORKING_DIR + 'input.json')
    m = Manager(nprocesses=4)
    for line in inputlines:
        j = Job(parameters=line)
        m.add_job(j)
    m.run_all()

    for j in m.joblist:
        print(j.get_data)

    # clean up
    files = ['main.pyc', 'params.pyc', 'var.py', 'var.pyc', 'preprocessing.pyc', 'misc.pyc', 'postprocessing.pyc', 'interpolation.pyc', 'simulation.pyc']
    clean_up(files)
```
1.1 params.py

Main configuration script.

```python
# -*- coding: utf-8 -*-
'''
Params
**************
This is parameter file for the whole optimization. All *global* parameters are defined here.

Important: When using Matlab, MATLAB must be set to True. For the matlab script, this means:
   * The genes cannot be passed to preprocessing, which means they need to be written in var.py as a
     dictionary.

Author: Jens

Version: 0.1
'''

# script parameters
MATLAB = False # set to true if using Matlab as the main loop (this can be done inside matlab)

Working dir get auch durch
   import os
   working_dir = os.getcwd() + '/'
automatisch...
'''
WORKING_DIR = 'C:/GitHub/Insert-Optimization/
JOB_NAME = 'JobInsert'
IMAX = 10 # max number of iterations

# abaqus parameters
MODEL_NAME = "HantelInsert_3D_3Koeff"
NCPUS = 4 # number of cpus (domains) for parallelization (1-12)
MESHTEST = False # if True, only the mesh generation will be tested, no simulations will be run!
```

1.2 var.py

File to pass information about the current iteration to other scripts. Example var.py:
i = 3
jobname = 'JobInsert3'
datadir = 'C:/GitHub/Insert-Optimization/HantelInsert_3D_3Koeff/
caedir = 'C:/GitHub/Insert-Optimization/HantelInsert_3D_3Koeff/JobInsert3/
cpus = '7'
modelname = 'HantelInsert_3D_3Koeff'
meshtest = 'False'

1.3 coeffs.py

File to pass geometry parameters to generatemodel. Example coeffs.py:

```python
a = 0.05775
b = -0.5775
c = 7.5
```
Preprocessing

Takes care of the necessary processing before GenerateModel() can be called.

Author: Jens

Version: 0.1

Preprocessing.copy_iteration_files(caedir)
  When finished preprocessing, copy var and the model generation script to the caedir so that we can execute
  abaqus there.

  Parameters caedir – path to place coeffs.py

Preprocessing.process_genes(genes, caedir)
  Main function for this script to be called from python:
  :param genes: a set of genes for the simulation. :type path: dict

Preprocessing.translate(genes, caedir)
  Preprocessing from genes to INP & CAE files.

  Translates genes to geometry (currently a set of points to coefficients of an interpolated polynomial) and writes
  the translated genes to the file coeffs.py in current job directory. Copies var.py and generate_model.py to cae dir
  and then runs generate_model.py.

  Parameters
    * genes – a set of genes for the simulation.
    * caedir – path to place coeffs.py
Interpolate points quadratically, adjust volume to target volume and return coefficients.

Author: Jens
Version: 0.1

**interpolation**. **integrate** *(f, a, b, N=50)*

Surprise! Integrates a function between two bounds.


**Parameters**
- **f** (*numpy.poly1d*) – function to integrate over
- **a** (*number*) – lower bound
- **b** (*number*) – upper bound
- **N** (*int*) – number of slices to use

**Return type** float

**interpolation**. **interpolate_points** *(inputdict)*

Generates coefficients for insert function out of 3 points & target volume Input needs to be a dict with the keys p1, p2, p3, vol

Example: inputdict = {'p1': [0,15], 'p2': [10,7.5], 'p3': [20,15], 'vol':40000}

**Parameters** **inputdict** (*dict*) – p1, p2, p3, vol

**Return type** dict

**interpolation**. **plot** *(coeffs)*

Plot insert shape. Requires matplotlib & numpy. To save figure, uncomment plt.savefig()

**Parameters** **coeffs** (*dict*) – coeffs returned from interpolate()

**interpolation**. **volume** *(f, a, b, N=50)*

Integrates the volume of a function rotated around the x-axis. 
**param f**: function to integrate over **type f**: *numpy.poly1d* 
**param a**: lower bound **type a**: *number* 
**param b**: upper bound **type b**: *number* 
**param N**: number of slices to use **type N**: *int* 
**type**: float
Postprocessing

This is the postprocessing script that runs outside of Abaqus/Python. It calls separate scripts within Abaqus/Python to extract the data from the odb.

Output: main() returns a dict object. Calling this as a script will write the data to CSV files in the caedir. Call with arguments datadir and jobname

Author: Jens

Version: 0.2

postprocessing.get_firstfailure(data, threshold=0.1)
Get first failure. Assumes 10 percent drop in RF3 means failure if threshold is not overridden.

Parameters

• data (dict) – data dict containing force at data[1]
• threshold (decimal) – threshold parameter (0.1 = 10 percent drop in force)

Return type float

postprocessing.getfielddata(caedir)
Calls gho.py (which extracts history data from the odb) and returns the unpickled data.

Parameters caedir (string) – cae directory containing the odb

Return type unpickled object

..note: Currently not implemented. Returns none.

postprocessing.gethistdata(caedir, jobname)
Calls gho.py (which extracts history data from the odb) and returns the unpickled data.

Parameters caedir (string) – cae directory containing the odb

Return type unpickled object

postprocessing.integrate_historydata(data, t)
Calculate the time integral of a history data set up to a specific time

Parameters

• data (dict) – data dict containing time and allke, rf3, etc.
• t (float) – time to integrate up to, must be less than the total time

Return type float

postprocessing.plothistorydata(data, keys, target, filename)
Save history data plots as PNGs.
Parameters

- **data** (*dict*) – dict of plottable data, accessible by key
- **keys** (*list of strings*) – keys to plot
- **target** (*string*) – directory to save plots to.
- **filename** – filename to save plots to. Will be amended with ‘_KEY.png’
Mesh Snap

# -*- coding: utf-8 -*-

'''
Mesh Snap
**************
Description:
This takes snapshots of the mesh from cae files.
.. note:: Run using abaqus cae noGUI

Author: Lukas

Version: 0.1
'''

from abaqus import *  # from the main library
from caeModules import *  # import the modules
from abaqusConstants import *  # constants we need
import assembly
import mesh
import visualization

#load Variables
import var

#take some pictures
openMdb(pathName= var.caedir + var.jobname + '.cae')
session.Viewport(name='VP1', origin=(0, 0), width=300, height=295)
session.viewports['VP1'].makeCurrent()
session.viewports['VP1'].maximize()

al = mdb.models[var.modelname].rootAssembly
session.viewports['VP1'].setValues(displayedObject=al)
session.viewports['VP1'].assemblyDisplay.setValues(mesh=ON)

#print Top
session.viewports['VP1'].view.setValues(session.views['Top'])
session.printOptions.setValues(vpDecorations=OFF, vpBackground=ON)
session.printToFile(
    fileName= var.datadir + var.jobname + '_mesh',
    format=PNG, canvasObjects=(session.viewports['VP1'], ))
CHAPTER 7

Miscellaneous

Description: These are miscellaneous functions. As soon as a function is needed by more than a single module, it should be implemented here.

Author: Jens

Version: 0.1

misc.clean_up(names, directory='')
Removes the files passed as arguments. If specified, directory must end in ‘/’.  

Parameters
• name (string) – file names or paths to file.
• name (string) – path to directory containing the files

misc.clear_file(name)
Clears the data in a file.

Parameters name (string) – file name or path to file.

misc.dump(data, name, protocol=2)
Dumps an object to file using pickle.

Parameters
• data (any) – object to dump.
• name (string) – file name or path to file.

Note: This function allows clean array pickling whereas standard pickle.dump will raise an error if array.array are in the pickled object (which is the case of all objects in Abapy).

misc.dump_json(data, name)
Dumps data to a human-readable json file, clearing the file beforehand.

Parameters
• data (any) – object to dump.
• name (string) – file name or path to file.

misc.dump_json_appending(data, name)
Dumps data to a human-readable json file, appending the data to the file.

Parameters
• **data** *(any)* – object to dump.
• **name** *(string)* – file name or path to file.

**Note:** Careful when appending to a file that was previously written with `dump_json...` the first two lines will be missing the newline inbetween.

**misc.ensure_dir**(path)

Creates a directory, if it doesn’t exist.

**Parameters**

- **path** *(string)* – path to directory.

**Return type**

string path to new directory

**Note:** To keep the directory references consistent, this function should always be called with a trailing slash: `'/'`.

**misc.load**(name)

Loads back a pickled object.

**Parameters**

- **name** *(string)* – file name or path to file.

**Return type**

unpickled object

**Note:** This function allows clean array unpickling whereas standard `pickle.load` will raise an error if `array.array` are in the pickled object (which is the case of all objects in Abapy).

**misc.load_json**(name)

Load data from a json file containing a single object. Use this to read files written with `dump_json()`

**Parameters**

- **name** *(unpacked object)* – file name or path to file.

**misc.load_json_lines**(name)

Load data from a json file containing multiple lines of objects. Use this to read files written with `dump_json_append()`

**Parameters**

- **name** *(list of unpacked objects)* – file name or path to file.
Python Module Index

i
interpolation, 5

m
main, 1
misc, 13

p
postprocessing, 9
preprocessing, 4
## Index

**C**
- clean_up() (in module misc), 15
- clear_file() (in module misc), 15
- copy_iteration_files() (in module preprocessing), 5

**D**
- dump() (in module misc), 15
- dump_json() (in module misc), 15
- dump_json_appending() (in module misc), 15

**E**
- ensure_dir() (in module misc), 16

**G**
- get_firstfailure() (in module postprocessing), 11
- getfielddata() (in module postprocessing), 11
- gethistdata() (in module postprocessing), 11

**I**
- integrate() (in module interpolation), 7
- integrate_historydata() (in module postprocessing), 11
- interpolate_points() (in module interpolation), 7
- interpolation (module), 5

**L**
- load() (in module misc), 16
- load_json() (in module misc), 16
- load_json_lines() (in module misc), 16

**M**
- main (module), 1
- misc (module), 13

**P**
- plot() (in module interpolation), 7
- plothistorydata() (in module postprocessing), 11
- preprocessing (module), 9
- preprocessing (module), 4
- process_genes() (in module preprocessing), 5
- translate() (in module preprocessing), 5
- volume() (in module interpolation), 7