pynoddy Documentation

Release

Florian Wellmann

March 25, 2014

CONTENTS

Contents:

CHAPTER

ONE

PYNODDY PACKAGE

1.1 Submodules

1.2 pynoddy.history module

Noddy history file wrapper Created on 24/03/2014

@author: Florian Wellmann

class pynoddy.history.**NoddyHistory**(*history*)

Class container for Noddy history files

change_cube_size(*cube_size*)

Change the model cube size (isotropic)

Arguments:

- *cube_size* = float : new model cube size

determine_events()

Determine events and save line numbers

Note: Parsing of the history file is based on a fixed Noddy output order. If this is, for some reason (e.g. in a changed version of Noddy) not the case, then this parsing might fail!

---

load_history(*history*)

Load Noddy history

Arguments:

- *history* = string : Name of Noddy history file

write_history(*filename*)

Write history to new file

Arguments:

- *filename* = string : filename of new history file

Hint: Just love it how easy it is to 'write history' with Noddy ;-)

1.3 pynoddy.output module

Noddy output file analysis Created on 24/03/2014

@author: Florian Wellmann

class pynoddy.output.**NoddyOutput**(*output_name*)
Class definition for Noddy output analysis

export_to_vtk(***kwds*)
Export model to VTK

Export the geology blocks to VTK for visualisation of the entire 3-D model in an external VTK viewer, e.g. Paraview.

..Note:: Requires pyevtk, available for free on: https://github.com/firedrakeproject/firedrake/tree/master/python/evtk

Optional keywords:

- *vtk_filename* = string : filename of VTK file (default: output_name)

load_geology()
Load block geology ids from .g12 output file

load_model_info()
Load information about model discretisation from .g00 file

plot_section(*direction='y'*, *position='center'*, ***kwds*)
Create a section block through the model

Arguments:

- *direction* = 'x', 'y', 'z' : coordinate direction of section plot (default: 'y')
- ***position* = int or 'center'** [cell position of section as integer value] or identifier (default: 'center')

Optional Keywords:

- *ax* = matplotlib.axis : append plot to axis (default: create new plot)
- *figsize* = (x,y) : matplotlib figsize
- *colorbar* = bool : plot colorbar (default: True)
- *title* = string : plot title
- *savefig* = bool : save figure to file (default: show directly on scren)
- *fig_filename* = string : figure filename

1.4 Module contents

Package initialization file for pynoddy

pynoddy.**compute_model**(*history*, *output_name*)

CHAPTER

TWO

SIMULATION OF A NODDY HISTORY AND VISUALISATION OF OUTPUT

Examples of how the module can be used to run Noddy simulations and visualise the output.

```
# Basic settings
import sys, os
import subprocess

# Now import pynoddy
import pynoddy

# determine path of repository to set paths corretly below
os.chdir(r'/Users/Florian/git/pynoddy/docs/notebooks/')
repo_path = os.path.realpath('../..')
```

2.1 (1) Compute the model

The simplest way to perform the Noddy simulation through Python is simply to call the executable. One way that should be fairly platform independent is to use Python's own subprocess module:

```
# Change to sandbox directory to store results
os.chdir(os.path.join(repo_path, 'sandbox'))

# Path to exmaple directory in this repository
example_directory = os.path.join(repo_path,'examples')
# Compute noddy model for history file
history_file = 'simple_two_faults.his'
history = os.path.join(example_directory, history_file)
output_name = 'noddy_out'
# call Noddy
print subprocess.Popen(['noddy', history, output_name],
                       shell=False, stderr=subprocess.PIPE,
                       stdout=subprocess.PIPE).stdout.read()
#
```

For convenience, the model computation is wrapped into a Python function in pynoddy:

```
pynoddy.compute_model(history, output_name)
```

2.2 (2) Loading Noddy output files

Noddy simulations produce a variety of different output files, depending on the type of simulation. The basic output is the geological model. Additional output files can contain geophysical responses, etc.

Loading the output files is simplified with a class class container that reads all relevant information and provides simple methods for plotting, model analysis, and export. To load the output information into a Python object:

```
reload(pynoddy)
N1 = pynoddy.NoddyOutput(output_name)
```

The object contains the calculated geology blocks and some additional information on grid spacing, model extent, etc. For example:

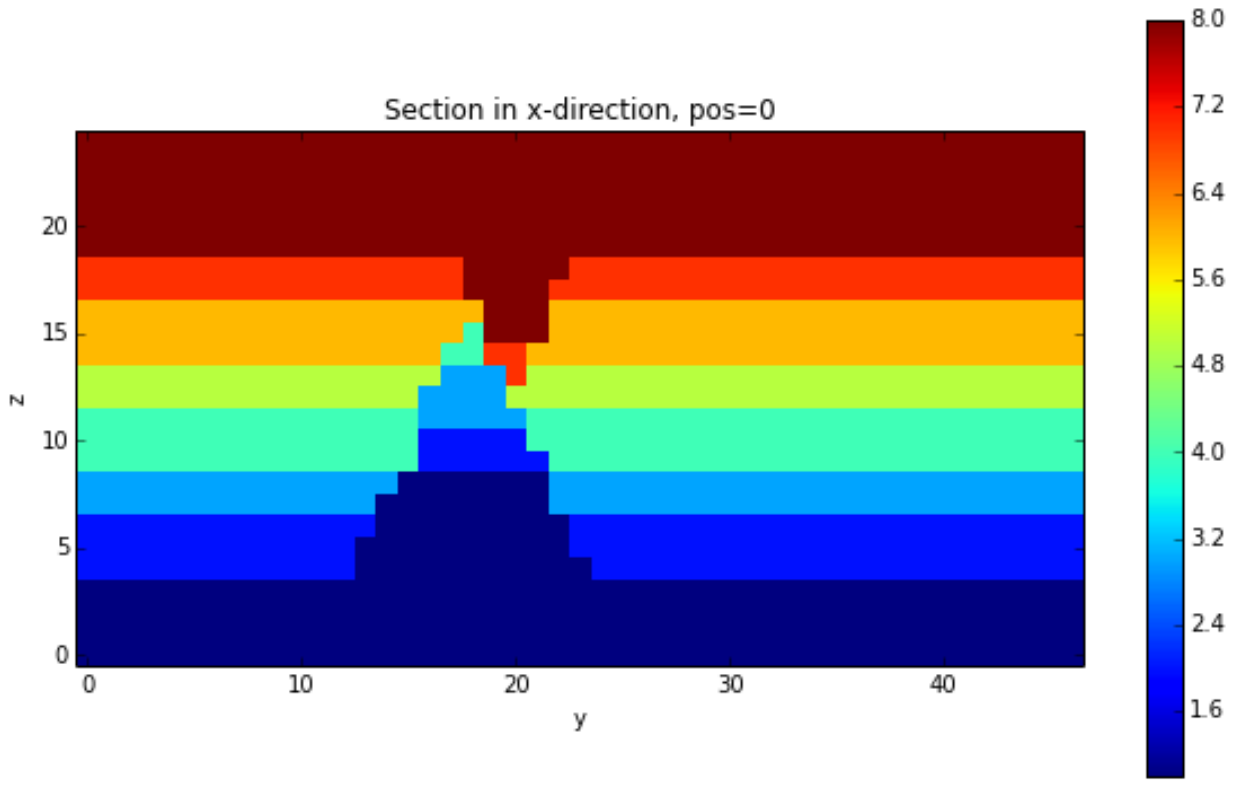
```
print("The model has an extent of %.0f m in x-direction, with %d cells of width %.0f m" %
      (N1.extent_x, N1.nx, N1.delx))
```

```
The model has an extent of 12400 m in x-direction, with 62 cells of width 200 m
```

2.3 (3) Plotting sections through the model

The NoddyOutput class has some basic methods for the visualisation of the generated models. To plot sections through the model:

```
N1.plot_section('x', position = 0)
```



2.4 (4) Export model to VTK

A simple possibility to visualise the modeled results in 3-D is to export the model to a VTK file and then to visualise it with a VTK viewer, for example Paraview. To export the model, simply use:

```
N1.export_to_vtk()
```

```
print("bla")
```

```
bla
```

CHAPTER

THREE

CHANGE NODDY INPUT FILE AND RECOMPUTE MODEL

Visualising output of a Noddy model is nice, but not terribly helpful as it can be done with the GUI itself. More interesting is it to change aspects of the Noddy input (history) file with the Python module to quickly compare the effect of changes in the geological history.

Implementing these changes in scripts is the first step to a more extensive uncertainty analysis, as we will see in the next section.

```
import sys, os
import matplotlib.pyplot as plt
# adjust some settings for matplotlib
from matplotlib import rcParams
# print rcParams
rcParams['font.size'] = 15
# determine path of repository to set paths corretly below
os.chdir(r'/Users/Florian/git/pynoddy/docs/notebooks/')
repo_path = os.path.realpath('../..')
import pynoddy
```

First step: load the history file into a Python object:

```
# Change to sandbox directory to store results
os.chdir(os.path.join(repo_path, 'sandbox'))

# Path to exmaple directory in this repository
example_directory = os.path.join(repo_path,'examples')
# Compute noddy model for history file
history_file = 'simple_two_faults.his'
history = os.path.join(example_directory, history_file)
output_name = 'noddy_out'
H1 = pynoddy.NoddyHistory(history)
```

3.1 (1) Get basic information on the model

The history file contains the entire information on the Noddy model. Some information can be accessed through the NoddyHistory object (and more will be added soon!):

```
pass
```

3.2 (2) Change model cube size and recompute model

The Noddy model itself is, once computed, a continuous model in 3-D space. However, for most visualisations and further calculations (e.g. geophysics), a discretised version is suitable. The discretisation (or block size) can be adapted in the history file. The according pynoddy function is change_cube_size.

A simple example to change the cube size and write a new history file:

```
# We will first recompute the model and store results in an output file for comparison
reload(pynoddy)
NH1 = pynoddy.NoddyHistory(history)
pynoddy.compute_model(history, output_name)
NO1 = pynoddy.NoddyOutput(output_name)

# Now: change cubsize, write to new file and recompute
NH1.change_cube_size(100)
# Save model to a new history file and recompute (Note: may take a while to compute now)
new_history = "fault_model_changed_cubesize.his"
new_output_name = "noddy_out_changed_cube"
NH1.write_history(new_history)
pynoddy.compute_model(new_history, new_output_name)
NO2 = pynoddy.NoddyOutput(new_output_name)
```

The different cell sizes are also represented in the output files:

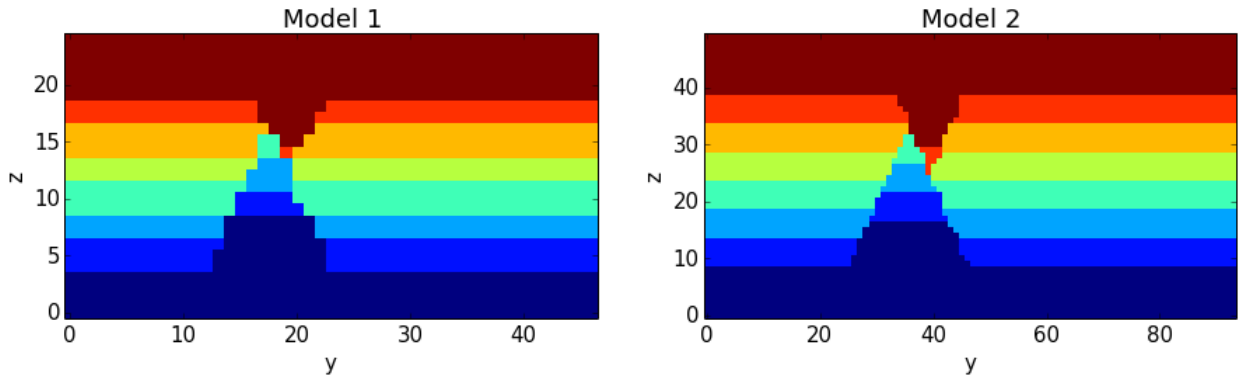
```
print("Model 1 contains a total of %7d cells with a blocksize %.0f m" %
      (NO1.n_total, NO1.delx))
print("Model 2 contains a total of %7d cells with a blocksize %.0f m" %
      (NO2.n_total, NO2.delx))
```

```
Model 1 contains a total of   72850 cells with a blocksize 200 m
Model 2 contains a total of  582800 cells with a blocksize 100 m
```

We can compare the effect of the different model discretisations in section plots, created with the plot_section method described before. Let's get a bit more fancy here and use the functionality to pass axes to the plot_section method, and to create one figure as direct comparison:

```
# create basic figure layout
fig = plt.figure(figsize = (15,5))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)
NO1.plot_section('x', ax = ax1, colorbar=False, title="Model 1")
NO2.plot_section('x', ax = ax2, colorbar=False, title="Model 2")

plt.show()
```



3.3 (3) Change aspects of geological events

Ok, now from some basic settings to the things that we actually want to change: aspects of the geological history defined in Noddy. This can happen on two hierarchical levels: on the level of each single event (i.e. changing parameters relating to one event) and on the level of the events themselves (i.e. the order of the events).

We will here have a look at the paramteres of the single events:

CHAPTER

FOUR

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

p

C

D

E

L

N

P

W