**Mureil Python Software**

# Introduction

## Document Summary

**Author:** Marcelle Gannon – [mgannon@student.unimelb.edu.au](mailto:mgannon@student.unimelb.edu.au), [marcelle.gannon@gmail.com](mailto:marcelle.gannon@gmail.com)

**Purpose:** This document outlines the status of the Python software written for the MUREIL energy system modelling project. It provides an overview of how the code fits together, how to configure simulations and how to extend and update the models, as well as some Python tips.

**Software:** see code.google.com, project mureil-ga, in the trunk. Written for Python v2.7.3.

**Document history:**

|  |  |  |
| --- | --- | --- |
| **Date** | **Author** | **Summary** |
| 28 February 2013 | Marcelle Gannon | Initial version |
|  |  |  |

**People named in this document:** all from the University of Melbourne – Roger Dargaville, Robert Huva, Elly Hutton, Steven Thomas, Peter Rainer

## Outline of modelling approach

### Meteorological data

### Genetic algorithm

### Multi-period operation

# Running the simulation using configuration files

## Configuration file format

### Master and sections

### Single vs multi-period values

Include a comment on how at least one of the periods in the multi-period config needs to appear in the combination of startup + run periods.

### config\_spec help

### Global variables

### Interaction between defaults, globals and configuration file(s)

## Scripts to run simulations

### Command-line options

### Setting starting points

## Helper scripts for output processing

# Software structure

## Code documentation

### Python ‘help’ command

## Python packages

## Configuration system & Model framework

### Overall framework concepts

### config\_spec

## Genetic Algorithm

## Data objects

### NetCDF reader

### Explicit data specification

## Generator objects

### Base classes

### Important considerations when implementing a generator

#### Handling of negative param values

#### Handling of ‘ts\_demand’ timeseries

### Variable generators

### Thermal generators

### Hydro generators

### Missed supply models

### Demand models

## Transmission models

## Master objects

### Output formats

## Multi-processing

## Unit and regression testing

### Unit tests

### Regression tests

### Running all tests

From the operating system prompt, run:

> python –m unittest discover –v

and expect it to take a few minutes. There will be some output as it runs. When finished, if all tests passed, it will report ‘OK’. If not, the reasons for the failures will be displayed. It’s good practice to run this before you check in code so that you know you have not broken some existing functionality. Note that there are a couple of tests that don’t pass for Roger for some reason – the test\_data/test\_ncdata.py and the test\_regression/rhuva\_test1. This is on the list of things to fix below.

# Web interface version (GE demo)

## Scripts to run GE demo

## Format of GE demo inputs and outputs

## Server implementation

# Further work

## Unfinished things

### Comments in geneticalgorithm.py on how it works

### Handling of negative params in gene

### Simple transmission model multi-period

Comment also that a simple-tx base class could be useful here.

### Using the get\_details function to identify demand and missed\_supply models in master

### Multi-period carbon handling in thermal models

### Globals for gas, coal prices, and handling in thermal models

### Complete the pumped hydro handling of multi-period, and for dam expansion

The comments in the code on what the units are of the parameters, in get\_config\_spec, also need clarification.

### Failing regression and unit tests

The test test\_data/test\_ncdata.py fails on Roger’s machine with the following message. It runs fine on Marcelle’s PC.

======================================================================

ERROR: test\_data.test\_ncdata (unittest.loader.ModuleImportFailure)

----------------------------------------------------------------------

ImportError: Failed to import test module: test\_data.test\_ncdata

Traceback (most recent call last):

File "/usr/local/python-2.7/lib/python2.7/unittest/loader.py", line 252, in \_find\_tests

module = self.\_get\_module\_from\_name(name)

File "/usr/local/python-2.7/lib/python2.7/unittest/loader.py", line 230, in \_get\_module\_from\_name

\_\_import\_\_(name)

File "/home/rogerd/MUREIL\_WC/test\_data/test\_ncdata.py", line 42, in <module>

import pupynere as nc

ImportError: No module named pupynere

The test test\_regression/rhuva\_test1 fails on Roger’s machine with the following message. It runs fine on Marcelle’s PC, and was set up from a simulation that Robert ran. The message isn’t very informative. It just says that the script single\_test.py that is in the test\_regression directory failed for some reason. A test\_out.pkl file wasn’t produced which suggests that it crashed somewhere. Further investigation is needed.

======================================================================

FAIL: test (test\_regression.rhuva\_test1.test.RegressionTest)

----------------------------------------------------------------------

Traceback (most recent call last):

File "/home/rogerd/MUREIL\_WC/test\_regression/rhuva\_test1/test.py", line 49, in test

test\_dir, config, pickle))

AssertionError: False is not true

### Cleanup of SVN branches

### Copyright messages

## Next steps

### Handling of discount rates

### Terminal values for models

### Calculation of O&M

### Capital cost models for multi-period

### Transmission model including flows

### Different dispatch order in different periods

### Multi-site thermal models

### Economic models

### Constraints on maximum total new build capacity

### Variable generators to use weather data instead of capacity factor data

## Ideas for performance improvement

### Genetic algorithm optimisations

#### Clone-test function performance

#### Combination with gradient-descent algorithm

#### Addition of a smaller-radius mutation

#### Definition of an ‘AlgorithmInterface’

### Orientation of timeseries data arrays

## Completion of formal testing

Some sections of the code are lightly tested, and/or not in the test set. Extra testing here would add confidence to the correctness of the models.

### timestep\_hrs

The models are written to accept the parameter ‘timestep\_hrs’ which specifies the timestep of the data timeseries. This is then used to calculate the MWh of electricity from a timeseries of MW, and the carbon emissions. Most of the use of the simulation to date has been with timestep\_hrs = 1.0. A specific review is needed of all models to check that timestep\_hrs is correctly applied, backed up with simple unit tests, and system tests where half-hourly and/or two-hourly data is used.

### Regression testing cleaning up and speeding up

The regression tests are currently a collection of whatever seemed to be an interesting test at the time, and together take a few minutes to run. Together they do cover a good proportion of the working functionality of the code. However, some of them take a long time to run. They could do with the number of iterations being reduced, with probably a minimal change to the effectiveness of the test. You can do this by editing the config file that’s in the test directory to change the iteration count, and then take the test\_out.pkl file and rename it to whatever the expected pickle file is. See the top of the test.py file for the name of the config and expected pickle files. Of course you can only do this with tests that already passed! The updated config and expected pickle files will be in SVN so make sure you commit them.

### Formal testing of the GE Demo results

# Python / System tips

## Performance Improvement

### Profiling

Profiling will help identify which parts of the program are taking the longest to run. Basic rule is - don't spend time optimising your code until you know what's taking all the time to run.

See: http://docs.python.org/2/library/profile.html#instant-user-s-manual

For example, run:

> python -m cProfile runmureil.py -f sample\_config.txt > sample\_config.prof

and browse sample\_config.prof to find where the time goes. The ‘cumtime’ column shows the total time spent inside this function and any functions it called. The ‘tottime’ column shows the time spent executing code actually in that function. Using sample\_config.txt as above you can see that 'tottime' for the calculate function is most of the run time of the sim. This is not surprising as this is the only calculate function that has a looped calculation in it - the others are all matrix maths which numpy does in a flash.

There are also ways to sort and search this information - see the help file for details.

### Numpy arrays vs Python lists

## SVN

SVN is the version control system on google code.

A list of useful commands here: http://www.thegeekstuff.com/2011/04/svn-command-examples/

The checkout instructions are on google code -> source -> checkout.

Most users will use 'add', 'commit', 'update', 'status' and 'diff'. It's good practice before doing a 'commit' to do 'status' and then do 'diff' on any files with an 'M' (for Modified) in front of them, to be sure you know what you've changed.

If you do an 'update' and it says that the merge failed, the file will be in conflict. SVN tries to combine changes that someone else has checked in with changes that you may have made locally. If you edit different parts of the same file this is likely to work. If you have edited the same parts of the file, then it will report a conflict.

See here for how to resolve it:

http://www.websanova.com/tutorials/svn/svn-conflicts

Don't whatever you do choose the (mc) mine-conflict option if 'update' offers you that. What that will do is ignore whatever you just updated and just use your new version - so you may be throwing away someone else's edits. This is often hard to find out and makes people very cross! (p) postpone is the best option.

## Finding Python help

Google is your best friend here. If you start a question with ‘numpy’ or ‘python’ you’ll get a good response. The site stackoverflow.com will often produce very useful suggestions and code snippets, with commentary.

The Python tutorial at <http://docs.python.org/2/tutorial/> is generally helpful and a good introduction to lists in particular.

The Numpy tutorial at <http://www.scipy.org/Tentative_NumPy_Tutorial> is essential reading to understand and use Numpy.

Also have a look at Scipy if you want to find library functions for a wide range of applications, in particular see scipy.optimise <http://docs.scipy.org/doc/scipy/reference/optimize.html>.

## Random Python tips

### Splitting lines

# Old stuff

This is a good place to copy stuff from the document (e.g. from the Further Work section) that’s not current, but might still be interesting to someone.