Flux Engine v3.0 – Changes and installation

Installing FluxEngine

1) Getting the FluxEngine source code

Download the latest version here: https://github.com/oceanflux-ghg/FluxEngine/archive/master.zip then unzip/extract the files to a location of your choice.

Alternatively, if you're a Git user you can clone the Git repository here: https://github.com/oceanflux-ghg/FluxEngine.git

You should now have a folder structure which looks something like this:

- `FluxEngine` – The root fluxengine directory
  - `fluxengine_src` – python source code
    - `tools` – various additional tools
  - `data` – bundled data
  - `configs` – example and validation configuration files

2) Installing dependencies

The only modules which are used by FluxEngine and not part of the standard library are: `netCDF4`, `numpy`, `scipy.integrate`. These need to be installed. Scripts are available to automatically install these for MacOS and Debian based systems (this uses `pip` with default settings). In most cases this will be acceptable, but some users may want to want to manually manage how these modules are installed.

MacOS – `install_dependencies_macos.sh` is located in the root (`FluxEngine`) folder and will automatically install these dependencies. Note that this has only been tested for El Capitan.

Debian/Linux – `install_dependencies_ubuntu.sh` is located in the root (`FluxEngine`) folder and will automatically install these dependencies. Note that this has only been tested on Ubuntu 16.4.

Windows users – an installation script isn't currently provided for Windows so users will need to install the dependencies manually. As python doesn't come pre-installed on Windows you may need to install *Python 2.7* (NOT *python 3.x*) as well as these additional modules. Rough instructions on how to do this are given below. They have not been tested and assume you're using *Windows 10*. Things might be a bit different for different versions of *Windows*.

Step 1) To download *Python 2.7.x* from here: https://www.python.org/downloads/
Note: at the time of writing the most recent version of *Python 2.7* was 2.7.14. Do not download version 3.x, FluxEngine will not run on it. Once downloaded run the file and you'll be guided through the installation process. Make a note of your installation directory (it will probably be something like `C:\Python27\` or `Users/<your username>/AppData/Local/Programs/Python/Python27`).

Step 2) Add Python and the python Scripts folder to your `PATH` environment variable. This means you can run python scripts without having to navigate to their specific location. To do this in *Windows 10*, follow these instructions: https://www.youtube.com/watch?v=uXqTw5eO0Mw using the installation directory from the previous step.

Step 3) Use *Pip* to install three modules: `numpy`, `scipy`, and `netCDF4`. Pip is a package manager, which comes with Python and allows you to add/remove additional libraries. First open a new command prompt window (or close and reopen an existing one) and type the following commands:

```
pip install --upgrade pip
pip install numpy
pip install netCDF4
pip install scipy
```

More detailed information on installing and configuring *Python 2.7* on *Windows* can be found here: https://docs.python.org/2.7/using/windows.html

3) Validating the FluxEngine

The easiest way to validate that FluxEngine is working correctly is to use the `validate_takahashi09.py` and/or `validate_socatv4_sst_salinity_gradients_N00.py` script. These can be found in the `tools` directory. All the data and configuration files required to run are located in the `data` and `configs` directories, respectively. The validation scripts will be able to find these if ran from the root directory, i.e.

```
python tools/ validata_socatv4_sst_salinity_gradients_N00.py
```

Running FluxEngine

FluxEngine v3.0 can be ran locally using a new python script, `ofluxghg_run.py`, which has replaced the perl script used in previous versions. To run FluxEngine from the command-line use the following command:

```
python ofluxghg_run.py configs/example_config.conf
```

Where `example_config.conf` should specify the path to your configuration file. Valid additional options are listed using the `-h` option (i.e. to view these use `python ofluxghg_run.py -h`). These include the ability to specify the years over which to run the model (`-year_start`, and `-year_end`), and whether or not to include process indicator layers (`-process_layers_off` or simply `-l`). For example the following command will run FluxEngine for the years 2000 to 2010, without calculating process indicator layers:

```
python ofluxghg_run.py configs/example_config.conf
-year_start 2000 -year_end 2010 -l
```

Creating and modifying configuration files

The format of configuration files has changed a bit from the previous version. Example configuration files can be found in the `configs` directory.

Data layers

The term 'data layer' is used to describe a 2D dataset which is used by FluxEngine either as input, as an intermediate product, or is outputted. Configuration files must specify all the input data layers which will be needed, but you only need to specify the input data layers which will be used given the specific options you've selected (i.e. if process indicator layers are off, you do not need to specify biology input files). If you try to run the FluxEngine without the required inputs you'll get an error message telling you which input datalayer was missing.

For each input data layer the configuration file must specify a `path` to the netCDF / .nc file, and a `prod` (variable name within the netCDF file). The path can be absolute or relative. Windows users might experience some problems using absolute paths. If you have problems with this please let me know (tom: t.m.holding@exeter.ac.uk) and I'll try to fix it. A path is specified like so:

```
datalayername_path = path/to/data/file.nc
```

with the data layer name prefixing "_path". One important change in this version of FluxEngine is that you should specify the path including a the filename for the netCDF file, rather simply a directory. This provides greater flexibility when working with data from many different sources. To help with this two standard Unix glob patterns can be used to specify patterns of file names: '?' and '*'. These will match any single character, or any number of characters (including zero characters), respectively. For example to specify the location of ice coverage data you could use:

```
ice_path = path/to/data/20100101_???-ice*.nc
```

This will match any file with a name that starts with "20100101_" followed by any three characters/digits, then the characters "-ice", followed by any number of characters/digits and ending in ".nc". Not that these cannot be used in directory names in paths and can only be used to specify the pattern that FluxEngine will use to match the netCDF file itself.

Several additional tokens can be used to specify file or directory paths which vary according to the current year or month being simulated, and allows different input files or directories to be used for different month/year combinations. The following tokens are supported:

- <YYYY> - four digit year, e.g. 2010
- <YY> - two digit year, e.g. 10 for 2010
- <MM> - two digit numerical month, e.g. 01 for January
- <Mmm> - three character abbreviation of the month, e.g. 'Jan' for January
- <MMM> - three character uppercase abbreviation of the month, e.g. 'JAN' for January
- <mmm> - three character lowercase abbreviation of the month, e.g. 'jan' for January

The variable name (or product/prod) is the name of the variable within the netCDF file, and must be specified for each input datalayer. The minimal specification for a single input data layer looks like this:

```
ice_path = path/to/data/<YYYY>/<YYYY><MM>_???-ice*.nc
ice_prod = sea_ice_fraction_mean
```

Optionally, a product name for standard deviation and/or count can be added by specifying `datalayername_stddev` and `datalayername_count`, respectively.

Modifying default metadata
Metadata about each input data layer will be automatically loaded. These defaults are specified in setting fluxengine_src/settings.xml, and generally shouldn't be changed directly. Config files can temporarily overwrite individual entries by prefixing with the data layer name. The following values can be overwritten for each data layer:

- `netCDFName` – the variable name which this data layer will have in any output netCDF files.
- `units` – a string description of the units
- `minBound` – minimum allowed value
- `maxBound` – maximum allowed value
- `standardName` – short standardised description of the variable/data layer
- `longName` – human-readable description of the variable/data layer

For example to overwrite the `minBound` and `maxBound` for ice coverage data layer you could add the following lines to a configuration file:

```
ice_minBound = 0.0
ice_maxBound = 100.0
```

Data preprocessing
It is sometimes convenient to apply some preprocessing to a data layer before it is used for any computations. A number of simple preprocessing functions are bundled with FluxEngine (these can be listed by running `ofluxghg_run.py` with the `-list_preprocessing` flag). For users comfortable with python, these are defined in `data_preprocessing.py`. Any functions added to this file will automatically be available as preprocessing functions. One or more preprocessing functions can be applied to a datalayer as by using `datalayername_preprocessing` and separating function names with commas. Preprocessing functions will be applied in the order they appear in the list. For example adding the following line to the config file will first transpose the 2D matrix, then convert from Kelvin to Celsius.

```
sstskin_preprocessing = kelvin_to_celsius, transpose
```

Note that no checks are made to ensure the original values are in kelvin to begin with, and it is up to the user to make sure they apply appropriate preprocessing.

k parameterisation
Rate parameterisation can now be specified by a string name, which corresponds to a class defined in `rate_parameterisation.py`. For most users the details of this do not matter, and a list of supported k parameterisations is provided in the example config files. For users comfortable programming in python and who may want to extent the FluxEngine by adding other k parameterisations, this can be done by writing a new class which extends `KcalculationBase` and placing it in `rate_parameterisation.py` whereupon it can be specified by name in the config file. A documented example is provided (see the `k_example` class).

Some k parameterisation options require additional parameters. These can be specifed by name in the configuration file. E.g. `k_generic` requires five additional parameters:

```
k_parameterisation = k_generic
k_generic_sc = ...
k_generic_a0 = ...
k_generic_a1 = ...
k_generic_a2 = ...
k_generic_a3 = ...
```

Currently the best way to check for any additionally required parameters is to look at the arguments that the `__init__` function used for the class which corresponds to your selected parameterisation in `rate_parameterisation.py` file, but FluxEngine will give error messages stating which parameters are missing if you try to run it without one.

Feedback

Any feedback, comments or bugs with this guide or the FluxEngine more generally can be sent to Tom Holding (lead developer) at t.m.holding@exeter.ac.uk or Jamie Shutler (project lead) at j.d.shutler@exeter.ac.uk