PyroPython – Parameter estimation for FDS pyrolysis models in python

This is simple python script that attempts to indentify pyrolysis parameters for FDS. The program si divided into three parts:

model.py - this file containse the claa Model. Contains functions for reading datafiles, fitness evaluation and running FDS and postprocessing FDS outputs config.py - provides configuration management plot_comp.py - plotting results. pyropython.py - main executable

currently all these files need to be in the same directory. TODO: more sensible packaging

As it is now, the program uses scikit optimize [https://github.com/scikit-optimize/scikit-optimize] to optimize the model. Scikit-optimize uses sequential model-based optimization (also known as Bayesian Optimization). The optimization uses a response surface fitted to the model evaluations to guide the optimization. TODO: add GAs such as shuffled-complex-evolution. (possibly using SPOTPY [http://fb09-pasig.umwelt.uni-giessen.de/spotpy/])

Prerequisites

1. FDS
2. Python 3
3. Python packages: Numpy, Scipy, Matplotlib, pandas, yaml, scikit-learn, scikit-optimize, Jinja2

Esiest way to install the prerequiaites

1. Install Anaconda (or miniconda) [https://www.anaconda.com/download/#linux]
2. Type:

   ```
   conda install pandas scikit-learn jinja2
   conda install -c conda-forge scikit-optimize
   ```

   Matplotlib and numpy should already be included in the download of anaconda, if you downloaded miniconda or otherwise are missing packages:

   ```
   conda install numpy scipy matploltib
   ```

Usage

Test installation by typing:

```
python pyropython.py -h
```

This should output the following help text

```
usage: pyropython.py [-h] [-v VERBOSITY] [-n NUM_JOBS] [-m MAX_ITER]
                     [-i NUM_INITIAL] [-p NUM_POINTS]
                     fname

positional arguments:
  fname                 Input file name

optional arguments:
  -h, --help            show this help message and exit
  -v VERBOSITY, --verbosity VERBOSITY
                        increase output verbosity
  -n NUM_JOBS, --num_jobs NUM_JOBS
                        number of concurrent jobs
  -m MAX_ITER, --max_iter MAX_ITER
                        maximum number of iterations
  -i NUM_INITIAL, --num_initial NUM_INITIAL
                        number of points in initial design
  -p NUM_POINTS, --num_points NUM_POINTS
                        number of points per iteration
```

input files

The input files are standard FDS input files with following additions

1. Theres should be a configuration block somewhere in the .fds file. The configuration is given in YAML format [https://en.wikipedia.org/wiki/YAML]. The configuration is placed between #config_start and #config_end blocks as follows

```
#config_start

... configuration goes here

#config_end
```

2. The Variables that are to be optimized shall be replaced with {{}} (e.g. the variable name surrounded by double curly brackets). For example:

```
#config_start
...
variables:
          VAR1: [ 0.1, 0.5]
...
#config_end
```

```
...

&MATL ID="ACME STUFF"
      DENSITY=100
      CONDUCTIVITY = {{VAR1}}
      SPECIFIC_HEAT=1
      ...
```

The input file is actuallya a Jinja2 template ([http://jinja.pocoo.org/]). This allows one to create various constructs such as create ramps from parametrized curves:

```
#config_start
...
variables:
          a: [ 0.1, 0.5]
          b: [ 0.1, 0.5]
...
#config_end

...

$MATL ID="test"
    CONDUCTIVITY = {{conductivity}}
/

{% for number in range(0,600,100) %}
  &RAMP ID="CP" , T={{number}} F={{a*number**2+b}}/ {% endfor %}
```

The for loop at the bototm of the previous example will create a "RAMP from 0 to 600 with step size 100 using with the ramp values calculated from second degree polynomial with parameters a and b. For example, with parameters a=0.01and b=2 the ramp resulting file will contain:

```
...

        &RAMP ID="CP" , T=0 F=2.0/
        &RAMP ID="CP" , T=100 F=102.0/
        &RAMP ID="CP" , T=200 F=402.0/
        &RAMP ID="CP" , T=300 F=902.0/
        &RAMP ID="CP" , T=400 F=1602.0/
        &RAMP ID="CP" , T=500 F=2502.0/
...
```

The jinja2 templates can also contain logic blocks etc.

configuration

The configuration block consists of optional and required fields:

```
#start_config
num_jobs: 36 # number of parallel jobs
max_iter: 5 # maximum number of iterations
num_points: 1  # How many points explored per iteration
num_initial: 50 # Number of points in initial design
fds_command: /home/tstopi/Firemodels/fds/Build/mpi_intel_linux_64/fds_mpi_intel_linux_64
variables:
          KS300: [ 0.2, 0.9]
          KS600: [ 0.2,  0.9]
          CS600: [ 1.20,   2.5]
          RHOC : [150,400]
          KC   : [ 0.01, 0.3]
#simulation - simulation outputfiles
#entries in format:
#   Var_name: output_file, col name, conversion factor
#Var_name is used to match the simulation output and experimental data
#output_file gives the fiel where this output is located int
#col name gives the column name in output file
#conversion factor  factor for converting from e.g. g to kg (i.e, c in  y=c*x)
#experiment - experimental data in same format
#   Var_name: data_file, col name, conversion factor
simulation:
    MLR: ['cone_hrr.csv','MLR_TOTAL',100000]
    HRR: ['cone_hrr.csv','HRR',100]
experiment:
    MLR: ['09090043_red.CSV','Specific MLR',1.0]
    HRR: ['09090043_red.CSV','HRR',1.0]

#The following fields are optional:
# Options for objective function
obejctive:
    type: "RMSE" # TODO: add more objectivefunctions
    weights: {'MLR': 1.0, 'HRR': 0.0}
# Information for plot_comp.py. Note that the results of the best run need to be in the Best
plotting:
    MLR: {ylabel: "MLR (g/m$^2$s)", xlabel: "Time (s)"}
    HRR: {ylabel: "MLR (g/m$^2$s)", xlabel: "Time (s)"}
# These are options for the optimizer class in scikit-optimize you can safely ignore these o
[https://scikit-optimize.github.io/#skopt.Optimizer]
optimizer:
         base_estimator:        'ET'
         acq_func:              'EI'
```

```
        acq_optimizer:         'auto'
        acq_optimizer_kwargs:  {n_points: 10000, n_restarts_optimizer": 5,n_jobs: 1}
        acq_func_kwargs:       {xi: 0.05, kappa: 0.96}
#end_config
```

Currently, the basic algorithm works as follows

1. Pick **num_initial points** randomly
2. Evaluate the objective function at these points using **num_jobs** processes (i.e. run FDS).
3. Begin iteration: Fit a regression model F(x) to the points (x,f(x)), where x is a parameter vector anf f(x) is the objective function.
4. Bssed on f(x), ask for **num_points** new points to explore.
5. Evaluate the the objective function f(x) at the new points.
6. If current iteration is not equal to or greater than **max_iter** go to 3.

If evolutionary algorithms are implemented they will follow the same basic structure, with the regression function replaced with whatever evolutionary mechanism one wishes to use.

Conficuration options

Job options

- **num_jobs**: How many parallel jobs to run when evaluating the objective function. (i.e. how many processors available for running FDS)
- **max_iter**: How many iterations to run after the initial design.
- **num_points**: How many points should be explored during each iteration. Probably makes sense to have **num_points** = **num_jobs**.
- **num_initial**: How many points to randomly sample before fitting the regression model/beginning evolutionary optimization
- **fds_command**: Give **full path** to the FDS executable that should be used.

variables

The variables (or parameters) to identify in the optimization. These are given as a list with in the format: - KEY: [lower bound, upper bound] The KEY value is used to replace values in the FDS input template. The tuple in the brackets gives the lower and upper bound within which the variables may vary

Categorical values are not yet supported (TODO)

Files and variables simulation and experiment

These two lists describe the variables to be compared. Format of these lists is KEY: [data_file, col name, conversion factor]

On the right hand side, the first element in the list gives name of the datafile. This file is assumed to be in .csv format. **col_name** gives the column name in the csv file. Note that the program attempts to strip possible unit information from column name. This means that "MLR (kg/s)" becomes just "MLR". **conversion_factor** gives multiplier used for unit conversion. That is the actual data used will be **actual_data**=**original_data** * **conversion_factor**.

The experiment data files are expected to be typical cone calorimeter output files with one header row and the column names in format "**col_name** (units)". The simulation datafiles are assumed to be standard FDS output files, with two header rows. The first header row, contianing unit informatioin, is skipped.

Running

The directory *Birch_Example* contains an example input for PyroPython