Mechanism optimization, code documentation

Vaisakh T V

Abstract

Description of a python code to perform chemical kinetic mechanism optimization using FlameMaster and NLOPT.

1 Prerequisites

This code is written in python 2.7 and require the use of external modules numpy and NLopt. FlameMaster (FlameMaster V3.3.10) must also be installed.

numpy and NLopt can be installed in ubuntu as

```
sudo apt-get install python-numpy
sudo apt-get install python-nlopt
sudo apt-get install python-matplotlib
```

2 Structure of input file

Input file is a text file containing information about the conditions needed for the optimize. When you run the script, this file should be given as the first argument. A sample of the input file is given below. Most of the keywords are self explanatory and can be tweaked within the code if necessary.

```
targets_count : 5
mechanism : /path/to/the/mechanism/file/Mechanism.mech
fuel : CH3OH
global_reaction : CH3OH + 1.5O2 == CO2 +2H2O
parallel_threads : 5
uncertainty_data : compendium.unsrt
targets:
1, target = Tig, simulation = isochor, T = 1250, P = 12e5, phi = 1.2, observed = 0.32,
2, target = Sl, simulation = premixed, T = 1000, P = 5e5, phi = 0.8, observed = 0.45
3, target = X-[species], simulation = isobar, T = 1450, P = 17e5, phi = 1.2, observed = 0.22
```

All keywords and their values must be seperated by a colon (**:**)and minimum one white space. Targets should contain all keywords followed by their values seperated by an equal sign (=). Different keywords are seperated by a comma(,).

target_count : is the number of target values present in the input file. If there are more targets, code will ignore the targets beyond the count. If there are lesser targets than mentioned in the count, it will show an error and exit.

mechanism : Location of the mechanism file. File should be a text file in FlameMaster format.

fuel and global reaction: These data should be given similar to the input format in FlameMaster.

parallel_threads : This is the number of threads that can be used while performing the simulation. Depending on the total number of cores in your system, a convenient number can be specified. Leave one or two cores free or it may cause system freeze.

uncertainty_data : This is the error associated with the estimation of reaction rate constants such that if f is the uncertainty factor pre-exponential factor should be within the limits of $\frac{A_0}{f} < A < A_0 f$. User has to provide uncertainty data for all the required reactions or in other words only those reactions whose uncertainties are given will be considered for optimization. The reactions should follow same sequence as they follow in the mechanism and should be preceeded by the same reaction index.

2.1 Targets

Each target corresponds to a particular type of simulation under a set of conditions in pressure temperature and equivalance ratio. It also include the experimental data for the same condition against which the mechanisme will be optimized. Experimental data should include the target value and associated error. Experimental data should be given in the same unit as of FlameMaster output. Errors should be given as fractions but not in percentage.

3 Structure of Program

Program is written in multiple modules so that additional components can be easily augmented. Program consist of the following actions.

- Read input file and identify the targets.
- Read mechanism file and uncertainty file to identify important reactions
- Generate Directories and required mechanism and input files to perform simulations
- Perform simulations in all the locations
- Extract the output value from simulation results.
- Generate a response surface for each target and calculate target value from it.
- Use objective function in NLopt minimization algorithm and perform optimization
- Generate optimized mechanism file using the optimized vector and print errors and other log files.

3.1 Major Modules in the code

1. Main code: Optimize
2. combustion_target_class
3. data_management
4. simulation_manager
5. make_input_file
6. FlameMaster_in_parallel
7. plotter

and several other Python standard modules..(See Python documentation for their details)

4 Explanation of program execution

This documentation is written to follow the execution pipeline of the program in its order. We will start from the main code and explain functions and classes whenever they are used in the main code. For better understanding, it is recommended to keep the code handy while reading.

Basic framework of the code is present in the file called **"optimize.py"** and all other modules are imported into it. Firstly keywords are defined for identfying informations input file. Input file is given to the code as an arguement. If no arguement is provided while calling the program, it will exit with an error.

Input file is read and each line is stored as elements into a list. Program searches for keywords in lines and store the values associated with it into variables. Target information is stored into a list by identifying the total number of targets. Each target value is stored as a string.

Once, the target strings are generated using a for loop and each string is used to generate instances of a class called **"combustion_target"** (From the module called combustion_target_class).

4.1 Class definition

A string containing the information about target is passed as an arguement while creating a class. Each target information is seperated by comma and contain a key (Keyword corresponding to a parameter)and content (value of that parameter). Program will check for the keywords in each element of list and assign the corresponding value to an instance variable. There are two methods through which input can be made. Either, parameters such as temperature, pressure, phi values can be given (FlameMaster will be executed with default configurations) or a FlameMaster input file can be directly given as an arguement(All parameters can be tweaked). If input file is specified, only the target need to be specified along with it. Even if other values are specified, they will be ignored by the program if input file is present. If target is laminar flame speed, then a start profile should also be specified with the keyword **"start_profile"**.

Each class is indexed from zero in ascending order to identify them in later part of the program. Once input file is read, program looks for uncertainty data file specified in the input file. This is a two column file containing reaction indices in first column and the uncertainty factor in second. reaction indices are stored in to a list and the factors are stored into a dictionary with reaction index as key. This operation is carried out using a function named **"extract_index_and_uncertainty"** in **"data_management"** module.

4.2 Preperation for FlameMaster Simulations

Next stage in the program is to prepare the directories for executing FlameMaster simulations and generate the required input files. FlameMaster is used in complete blackbox fashion in this program. A shell script is used to execute **ScanMan** and **FlameMaster**. For each execution, a perturbed mechanism named **mechanism.mech**, a FlameMaster input file named **FlameMaster.input**, a shell script named **run** are to be generated. All the commands for execution of these programs and cleaning of unwanted output files are written into the shell script. Progress of execution is stored into a file named **progress** in the home directory. This file is appended with the location of execution whenever one execution is completed. This file is compared with another file named **locations** to resume from an interuption while execution.

Operations related to preparation of directories and files are carried out by **simulation_manager** module. A function -**Make_directories_for _simulation** - contains the code for preparing the directories. Figure (1) shows a schematic representation of the execution locations. Reaction indices are used for naming the directories. A perturbed mechanism file, input file, output directory and a shell script are generated inside each directory. Function returns a list of execution locations named **dir_list**.

This function uses two other functions within itself. One to generate perturbed mechanism from the original file (**create_mechanism_file**). Original mechanism file, list of active reactions uncertainty information and a flag ("plus" or "minus"). Based on the flag, a value of +1 or -1 is assigned to **x**. A regular expression is used to identify reactions within the mechanism file and to extract pre-exponential factor from the reaction. Depending on whether forward and backward reactions are seperately mentioned

in the mechanism file, backward reactions are also perturbed by the same factor. If they are not present, it is ignored. Pre-exponential factor for the i^{th} are perturbed by the equation

$$A_{i_{new}} = A_{i_{old}} \times e^{x_i log(f_i)} \tag{1}$$

FlameMaster input file is generated using a function (**create_input_file**) from the module **make_input_file**. Templetes for different simulations are stored within that function. Based on the nature of target and the simulation required, appropriate input file will be created. There are two methods for input file creation, if user gives temperature (T), pressure (P) and equivalance ratio (ϕ), a fuel-air configuration will be created. If user provides location to the input file, then the same configuration in the file will be used.

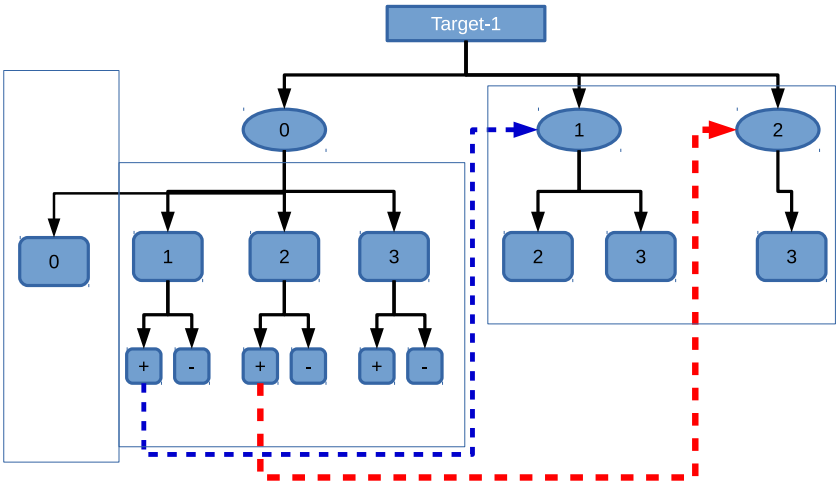

Figure 1: Schematic arrangement of directories for FlameMaster execution. One reaction is perturbed at a time. First in postive direction (reaction rate is increased) and then in negative direction (reaction rate is decreased). The positively perturbed mechansim is further used to perturb one more reaction in positive direction. 1 target with n active reactions require $1 + 2n + nC_2$ simulations.

These data are written into files named mechanism.mech and FlameMaster.input. Once the directories are created, function returns a python list containing the locations of FlameMaster execution and this is written into a text file named **locations**. If a file named **progress** is not present, all the execution locations are generated and simulations are started from the beginning. If progress file is present, the file is compared with the location file and the missing locations are identified. Simulation is performed only for the remaining locations. Simulations are performed by the function called **run_FlameMaster_parallel** from the module **FlameMaster_in_parallel**. List of execution locations, number of cpu cores to be used for the simulations, location of thermo-transport data (**.bin**) file are provided as arguements to the functions.

4.3 Execution of FlameMaster simulations

As a pre-cursor to the simulations, shell scripts are rewritten to incorporate absolute paths for the bin file location, mechanism file and input file. These files are made executable. Multiprocessing module from standard python library is used for creating a pool of cores and distribute the execution commands among them. Function takes two functions within itself. One is to run the shell script (**run_script**) and another is to identify the the termination of execution (**log_result**). The log reuslt function calculates the progress of simulation by comparing the length of total execution locations and the locations where

simulation is completed (As percentage).(Read multiprocessing module from python documentation for details of parallel execution.) A clock is used to calculate the total time for simulations.

4.4 Extraction of output data

Data_management module is used to extract information from the output directories. Type of target (Ignition delay or flame speeds) is used to identify the respective output file. A function named **generate_target_value_tables** is used to execute this operation. It takes execution locations, list of target classes, list of case directories (case-0, case-1 etc.) and the fuel name to identify the output file. Function reads execution locations and changes directory to that location. Reaction indices are extracted from the location string and stored as primary and secondary based on their location. (Refer fig(1) for identifying the primary and secondary indices.) A function named **extract_output** is used to open the output file and read the output data based on the standard output file format. One table containing the data from all directories are stored in each case directories.

The format for the table is

< **primary** > < **secondary** > < **positive/negative** > < **value** >

A sample of the data table is given below.

```
0       C1609f   plus     0.819612
0       C1609f   minus    1.71781
0       C1615f   plus     0.837801
0       C1615f   minus    1.69648
C1600f  C1711f   Non      2.14252
C1600f  C1708f   Non      2.08221
C1600f  C1760f   Non      0.873485
```

First column denotes the primary perturbation of one reaction and second column denote the secondary perturbation. Third column denote the type of perturbation (either positive or negative) . Fourth column denote the value of the target. Function returns a list and these list is written into a file (**"sim_list.lst"**)by the main code.

4.5 Generation of response surface

Output data from the list prepared earlier is used to generate the response surface coefficients. A function named **create_response_surface** from the module **combustion_target_class**. This is written as a method for the class. A function named **make_eta_lists** reads the four column data from the text file named **"sim_list.lst"** and sort them into seperate lists based on the information in first three columns. Positive and negative perturbations are strored into lists and the secondary perturbations (Two reaction rates changed at a time.) are stored into a matrix. If first column value is **'0'**, second column value is used for identifying the reaction index and the third column is used to identify whether it is positive or negative perturbation. If first column is not **'0'** then first and second column are used for identifying the perturbed reactions and the value is stored into the matrix. These arrays are used to generate the response surface co-efficients using the following expressions.

Coefficients are determined by:

$$a_i = \frac{\eta_{+i} - \eta_{-i}}{2.0}$$

$$a_{ii} = \frac{\eta_{+i} - 2\eta_0 + \eta_{-i}}{2.0}$$

$$a_{ij} = \eta_{ij} - \eta_o - a_i - a_{ii} - -a_j - a_{jj}$$

where,

$\eta_0 = 0$ 0 None $< value >$

$\eta_{+i} = 0$ i plus $< value >$

$\eta_{-i} = 0$ i minus $< value >$

$\eta_{ij} =$ i j None $< value >$

Once the coefficients are determined, they are used for generating the response surface polynomial.

The polynomial is of the form

$$\eta(x) = \eta(0) + \sum_{i=1}^{n} a_i x_i + \sum_{i=1}^{n} a_{ii} x_i^2 + \sum_{i=1}^{n} \sum_{j>i}^{n} a_{ij} x_i x_j + ... \tag{2}$$

Where x_i are the normalized pre-exponential factors given by,

$$x_i = \frac{log\left(\frac{A_i}{A_{i0}}\right)}{log(f_i)} \tag{3}$$

All these operations are carried out using two functions named **"create_response_surface"** in the module **"combustion_target_class"** rest of the code contains the specifications for the library NLopt. Dividing Rectangles algorithm is used for minimization of the objective function. Objective function is given by

$$\phi = \sum_{k=1}^{N} \left(\frac{\eta_k(\vec{x}) - \eta_{k_{obs}}}{\sigma_{obs}} \right)^2 \tag{4}$$

initial guess is a vector of length equal to the number of active reactions. These values are given to the form of numpy array. (See python-numpy documentation for details...). Lower and upper bounds are specified. Optimization algorithm is terminated either by completing 10000 function evaluations for each variables.

Optimized vector is used to generate optimized mechanism. Errors are reported and a plot containing the errors in old and new mechanism is generated.