Cantera Tutorial

1-D Premixed Free Flame with CANTERA

11 septembre 2017

1 Introduction - Objectives

The aim of this tutorial is to walk non-experienced CANTERA users through the computation of a 1-D steady-state free flame simulation, with the Python interface.

The script presented here deals with a Methane-Air premixed free flame, at stoichiometry and under atmospheric conditions ; but can easily be altered for other fuels or operating points. The mechanism used is the GRIMech 3.0 with mixture transport properties, but it is also straightforward to continue the computation if different transport models need to be considered. The saving format can as easily be adapted.

2 Steps

2.1 Get the script and mechanism

- The script can be downloaded here : http://www.cerfacs.fr/cantera/docs/scripts/flames/adiabatic_flame.py
- And the .cti file here : http://www.cerfacs.fr/cantera/docs/mechanisms/methane-air/DETAILED/CANTERA/gri30.cti
- They should be placed in a CANTERA sub-directory of your choice.

If later on, you wish to go further, similar scripts can be found here : http://www.cerfacs.fr/cantera/docs/scripts/flames/ .

As said previously, it will run a 1-D free flame for which methane 'CH4' is the fuel species and air the oxidizer. Other mechanism than the GRIMech 3.0 could be used, such as those presented on the website : http://www.cerfacs.fr/cantera/mechanisms/meth.php.

2.2 Load the CANTERA module

# ! Remember to source the code ! #

Open a terminal and simply type :

```
module load cantera/2.1.1
```

This will enable you to run your CANTERA scripts.

2.3 Launch your computation

Now, on the same terminal, go into your CANTERA sub-directory and type :

```
python adiabatic_flame.py
```

If all is going well, the program will automatically refine the mesh where needed in the flame domain, to predict brutal changes, until convergence is reached. The simulation is first computed with the energy equation disabled, to help the newton algorithm approach the final solution, and then switched on before refinement can take place. It takes about 50 seconds to get the results on a standard local machine.

By the end you will be able to read the mixture-averaged flame speed along with some statistics pertaining to your run, directly on your terminal :

```
..............................................................................
Attempt Newton solution of steady-state problem...    success.

Problem solved on [262] point grid(s).
..............................................................................

no new points needed in flame
mixture averaged flamespeed =  0.370574078438
Solution saved to ’ch4_adiabatic.csv’.
```

You will also see a plot pop up in front of your terminal, with a number of quantities plotted. Note that you will need to import matplotlib for this to work !

2.4 Saving files

Data are written in a **.csv** output file that can be imported into Excel. A **.xml** file is also created, very useful to restart computations later on. Another common type of output file, if you are from CERFACS, is the ”can2av” format. It can be generated by a call to the ”write_avbp” function :

```
#################################################################
# Save your results if needed
#################################################################
#Write the velocity, temperature, density, and mole fractions to a CSV file
f.write_csv(’ch4_adiabatic.csv’, quiet=False)
#f.write_avbp(’Sol-CAN2AV_P-’+str(p)+’-T-’+str(tin)+’-Phi-’+str(phi)+’.csv’, quiet=False)
```

3 To go further

3.1 Change your initial grid

If you would like to change the grid used, you will need to open the **adiabatic_flame.py** with a text editor. You will see this :

```
...
#Different initial grid, chosen to be 0.02cm long :
       # - Refined grid at inlet and outlet, 6 points in x-direction :
```

```
initial_grid = 2*array([0.0, 0.001, 0.01, 0.02, 0.029, 0.03],'d')/3 # m
       # - Uniform grid, 6 points in x-direction :
#initial_grid = 0.02*array([0.0, 0.2, 0.4, 0.6, 0.8, 1.0],'d') # m
       # - Uniform grid of 300 points using numpy :
#initial_grid = numpy.linspace(0,0.02 , 300)
...
```

Uncomment the one you'd like to use (delete the hashtag # at the begining of the line) and make the sure the others are commented. The grid is set to be 0.02 cm long. If you'd like to change this, you'll need to change the value highlighted in magenta above for the grid you've chosen.

3.2 Change transport properties

The first argument of the "Solution" function should be the name of the **.cti** mechanism used. The second argument refers to the name of the phase you are using. Usually, phases in **.cti** mechanism files differ by the type of transport model employed.

The common transport models are :

— Multi : this model uses multicomponent properties,

— Mix : a model that implements a mixture-average formulation,

— AVBP : the CERFACS AVBP transport model will be used, on mechanisms that allow it. Additional files are needed for this model.

In general, the results of the mixture-averaged transport model are less accurate than those obtained with the multicomponent model, but are far less expensive to compute. It all depends upon the fuel you are using and the accuracy required! Usually however, you will have to start with the mixture-averaged transport model before switching to any other one.

The transport property chosen in the original script is mixture averaged, but will be switched to multicomponent model by adding this line in the **adiabatic_flame.py** file :

```
...
#Import gas phases with mixture transport model
gas = Solution('gri30.cti')
...
f.transport_model = 'Multi'

f.solve(loglevel, refine_grid)
```

If you change the transport properties from mixture averaged to multicomponent in the flame considered in this tutorial, you will notice a small difference in the flamespeed, and a negligibly small difference in the final adiabatic temperature (of the order of 0.1 K). However, **the computational price is quite high**, as can be seen from running the simulation :

```
..............................................................................
Attempt Newton solution of steady-state problem...    success.

Problem solved on [262] point grid(s).
..............................................................................

no new points needed in flame
mixture averaged flamespeed =  0.370574078438

..............................................................................
```

```
Attempt Newton solution of steady-state problem...    failure.
Take 10 timesteps     2.816e-07      5.475
Attempt Newton solution of steady-state problem...    failure.
Take 20 timesteps     4.572e-07      5.332
Attempt Newton solution of steady-state problem...    failure.
Take 80 timesteps     1.589e-06      4.823
Attempt Newton solution of steady-state problem...    failure.
Take 80 timesteps      2.76e-06      4.106
Attempt Newton solution of steady-state problem...    success.

Problem solved on [262] point grid(s).
..............................................................................

no new points needed in flame
multi-comp flamespeed =  0.375148989984
Solution saved to 'ch4_adiabatic.csv'.
```

3.3 Change operating conditions

There are some operating conditions that can be changed in the first lines of the script :

- Temperature and Pressure conditions

```
    #Parameter values :

           #General
p           =   1.e5            # pressure
tin         =   300.0             # unburned gas temperature
```

- Gas composition

```
phi    =  1.   # premixed gas composition
```

Try any other values ! You may have to alter the inital grid, or change the computing properties to reach convergence ...

3.4 Change computing properties

The following computing properties can (and often : should !) be changed. In a text editor again, open the **adiabatic_flame.py** script :

- Tolerance properties on steady-state or time steps :

```
tol_ss    = [1.0e-5, 1.0e-9]        # [rtol atol] for steady-state problem
tol_ts    = [1.0e-5, 1.0e-9]        # [rtol atol] for time stepping
```

- Refinement criteria :

```
#Set Refinement Criteria
f.set_refine_criteria(ratio = 10.0, slope = 1, curve = 1)
```

ratio : Additional points will be added if the ratio of the spacing on either side of a grid point exceeds this value

slope : Maximum difference in value between two adjacent points, scaled by the

```
maximum difference in the profile (0.0 < slope < 1.0).
Adds points in regions of high slope.

curve : Maximum difference in slope between two adjacent intervals, scaled by the
maximum difference in the profile (0.0 < curve < 1.0).
Adds points in regions of high curvature.
```

There are other refinement criteria when the energy equation is set on ...

- Time steps :

  ```
  #Set time Steps
  f.set_time_step(1.0e-5, [2, 5, 10, 20, 50]) #s

  In case the Newton convergence fails, the program undergoes a sequence
  of time steps to try and enter the Newton convergence domain. The first
  number is the absolute value of the time steps (here, 0.01 ms) and the
  following sequence is the number of time steps taken each time the
  Newton algorithm fails.
  ```

- Jacobian matrix :

  ```
   f.set_max_jac_age(50, 50)

  It sets the number of time the Jacobian will be used before it must be
  re-evaluated, either during the steady-state mode (first number) or
  during time-stepping mode (second one).
  ```

4 Convergence hints

Depending upon the complexity of the mechanism, the transport model used, the grid used (...) convergence issues may arise. Here are some hints to try and handle these common problems :

- Coarsen the grid : Use a initial grid with fewer points. Try a uniform grid.
- Increase slope and curve parameters : it will decrease the number of points in the solution (maximum value of 1). Of course, at the expense of accuracy... but values of around 0.1 for slope and 0.2 for curve will usually give well-resolved profiles.
- Decrease time steps : problems can arise from too large time stepping.
- Decrease both the tolerances for steady-state problem and time-stepping.

If you still have convergence problems, you might want to increase the value of the loglevel to about 5, which will generate a great deal of information, including which solution component won't converge.