Tutorial 1

1.2.1 Define a gas object through a script :

```
#File gas_simple.py
import cantera as ct

gas = ct.Solution('gri30.xml')
print gas()
print gas.species_names
```

Tutorial 1

1.2.1 Define a gas object through a script (Results) :

[felden@Quincy-MacBookPro-CFD-2: Tuto1]$python gas_simple.py

WARNING: 'class GRI_30_Kinetics' is deprecated. To be removed in Cantera 2.2.

gri30:

temperature	**300 K**
pressure	**101325 Pa**
density	0.0818891 kg/m^3
mean mol. weight	2.01588 amu

	1 kg	1 kmol	
enthalpy	26470.1	5.336e+04	J
internal energy	-1.21087e+06	-2.441e+06	J
entropy	64913.9	1.309e+05	J/K
Gibbs function	-1.94477e+07	-3.92e+07	J
heat capacity c_p	14311.8	2.885e+04	J/K
heat capacity c_v	10187.3	2.054e+04	J/K

	X	Y	Chem. Pot. / RT
H2	**1**	**1**	**-15.7173**
H	0	0	
		....	
CH3CHO	0	0	

New objects created from the 'gri30.xml' input start out with a temperature of 300 K, a P of 1 atm, and have a composition that consists of only one species

Tutorial 1

1.2.2 The Python terminal :

[felden@Quincy-MacBookPro-CFD-2: Tuto1]$python
Python 2.7.10 (default, Sep 23 2015, 04:34:03)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> **import cantera as ct**
>>> **gas = ct.Solution('gri30.xml')**

Same steps as in the script …

Tutorial 1

1.2.2 The Python terminal :

[felden@Quincy-MacBookPro-CFD-2: Tuto1]$python
Python 2.7.10 (default, Sep 23 2015, 04:34:03)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> **import cantera as ct**
>>> **gas = ct.Solution('gri30.xml')**
WARNING: 'class GRI_30_Kinetics' is deprecated. To be removed in Cantera 2.2.
>>> **help(gas)**

Will list the gas object properties …

Help on Solution in module cantera._cantera object:

class **Solution**(ThermoPhase, Kinetics, Transport)
 | A class for chemically-reacting solutions.
 |
 | Instances can be created to represent any type of solution -- a
 | mixture of gases, a liquid solution, or a solid solution, for
 | example.

…

Tutorial 1

1.2.2 The Python terminal :

```
[felden@Quincy-MacBookPro-CFD-2: Tuto1]$python
Python 2.7.10 (default, Sep 23 2015, 04:34:03)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import cantera as ct
>>> gas = ct.Solution('gri30.xml')
WARNING: 'class GRI_30_Kinetics' is deprecated. To be removed in Cantera 2.2.
>>> gas.n_atoms('CH4','H')
4.0

...

>>> exit()
```

Will list the gas object properties … Try some !

Tutorial 1

1.2.3 Modify the state of your "gas" object :

```
#File gas_simple.py
import cantera as ct

#First script
gas = ct.Solution('gri30.xml')
#print gas()
#print gas.species_names

#Second script
gas.TP = 800, 202650
print gas()
```

Tutorial 1

1.2.3 &1.2.4 Modify the state of your "gas" object (Results) :

[felden@Quincy-MacBookPro-CFD-2: Tuto1]$python gas_simple.py

...

gri30:

temperature	**800**	**K**
pressure	**202650**	**Pa**
density	**0.0614168**	**kg/m^3**
mean mol. weight	2.01588	amu

	1 kg	1 kmol	
enthalpy	7.29094e+06	1.47e+07	J
internal energy	3.99135e+06	8.046e+06	J
entropy	76290.8	1.538e+05	J/K
Gibbs function	-5.37417e+07	-1.083e+08	J
heat capacity c_p	14691.3	2.962e+04	J/K
heat capacity c_v	10566.8	2.13e+04	J/K

	X	Y	Chem. Pot. / RT
H2	1	1	-16.2874
H	0	0	

...

Thermodynamics generally requires that two properties in addition to composition information be specifed to fix the intensive state of a substance …

Ex:

***gas.TP** = 800, 202650 # temperature, pressure*

***gas.TD** = 800, 0.0614168 # temperature, density*

…

***gas.TPX** = 800, 202650, 'CH4:1, O2:2, N2:7.52' # temperature, pressure, composition*

Tutorial 1

1.3.2 Generating a 'CTI' input file from CHEMKIN input files, Results :

[felden@Quincy-MacBookPro-CFD-2: Tuto1]$cd CK2CTI/
[felden@Quincy-MacBookPro-CFD-2: CK2CTI]$ck2cti --input=chem.inp --thermo=therm.dat --transport=tran.dat
INFO:root:Skipping unexpected species "C(S)" while reading thermodynamics entry.
INFO:root:Skipping unexpected species "n-C4H7" while reading thermodynamics entry.
INFO:root:Skipping unexpected species "c-C6H3" while reading thermodynamics entry.
INFO:root:Skipping unexpected species "A1C2HC2H2" while reading thermodynamics entry.
INFO:root:Skipping unexpected species "A2HR5" while reading thermodynamics entry.
INFO:root:Skipping unexpected species "A2R5" while reading thermodynamics entry.
INFO:root:Skipping unexpected species "A4H" while reading thermodynamics entry.
Wrote CTI mechanism file to 'mech.cti'.
Mechanism contains 99 species and 533 reactions.

Generation of a data file for Cantera from CHEMKIN data files

Tutorial 2

Reminder : Input files syntax

```
Directives, Entries, Fields = values

units(length = "cm", time = "s", quantity = "mol", act_energy = "cal/mol")

ideal_gas(name = "gri30_multi",
    elements = " O  H  C  N  Ar ",
    species = """ H2  H  O  O2  OH  H2O  HO2  H2O2  C  CH
            CH2  CH2(S)  CH3  CH4  CO  CO2  HCO  CH2O  CH2OH  CH3O
            CH3OH  C2H  C2H2  C2H3  C2H4  C2H5  C2H6  HCCO  CH2CO  HCCOH
            N  NH  NH2  NH3  NNH  NO  NO2  N2O  HNO  CN
            HCN  H2CN  HCNN  HCNO  HOCN  HNCO  NCO  N2  AR  C3H7
            C3H8  CH2CHO  CH3CHO """,
    reactions = "all",
    kinetics = "GRI30",
    transport = "Multi",
    initial_state = state(temperature = 300.0,
                pressure = OneAtm)    )

reaction( "O + H2 <=> H + OH",   [3.87000E+04, 2.7, 6260])

three_body_reaction( "2 O + M <=> O2 + M",   [1.20000E+17, -1, 0],
      efficiencies = " AR:0.83  C2H6:3  CH4:2  CO:1.75  CO2:3.6  H2:2.4  H2O:15.4 ")
```

Tutorial 2

Reminder : Input files syntax

Units syntaxing rules

```
Pressure = 1.0e05      #Pascals
Pressure = (1.0, 'bar') #Bars

cm6/mol2/s         #OK
cm^6/mol^2/s       #Error
cm6/mol2-s         #Error
```

Tutorial 2

Reminder : Input files syntax

Fields names :

```
element(symbol='Ar', atomic_mass=39.948)    #OK recommended
element('Ar', 39.948)                       #OK is equivalent

element('Ar', atomic_mass=39.948)           #OK
element(symbol='Ar', 39.948)                #Error
```

Tutorial 2

2.5.1 Generate a CTML data

✓ Cantera starts by generating a CTML (‘.xml’) file from the CTI (‘.cti’) file

Generate a CTML file (‘.xml’) from a CTI file (‘.cti’)

```
$ cti2ctml mechanism.cti
```

This step detects syntaxing errors :

```
get_CTML_Tree: caught an exception:

*************************************************************************
CanteraError thrown by ct2ctml:
Error converting input file "./h2_sandiego.cti" to CTML.
…
```

✓ *This is done automatically when launching a script that processes a ‘.cti’ file !*

Tutorial 2

2.5.1 Generate a CTML data

Now, run the ‘h2_flame.py’ script into the subdirectory 'SANDIEGO' of the directory 'Tuto2':

```
[felden@Quincy-MacBookPro-CFD-2: Tuto2]$ cd SANDIEGO
[felden@Quincy-MacBookPro-CFD-2: SANDIEGO]$ python h2_flame.py

get_CTML_Tree: caught an exception:

***************************************************************************
CanteraError thrown by ct2ctml:
Error converting input file "./h2_sandiego.cti" to CTML.
Python command was: '/usr/bin/python'
The exit code was: 3
-------------- start of converter log --------------
SyntaxError in "./h2_sandiego.cti" on line 15:

...
```

Tutorial 2

2.5.1 Generate a CTML data

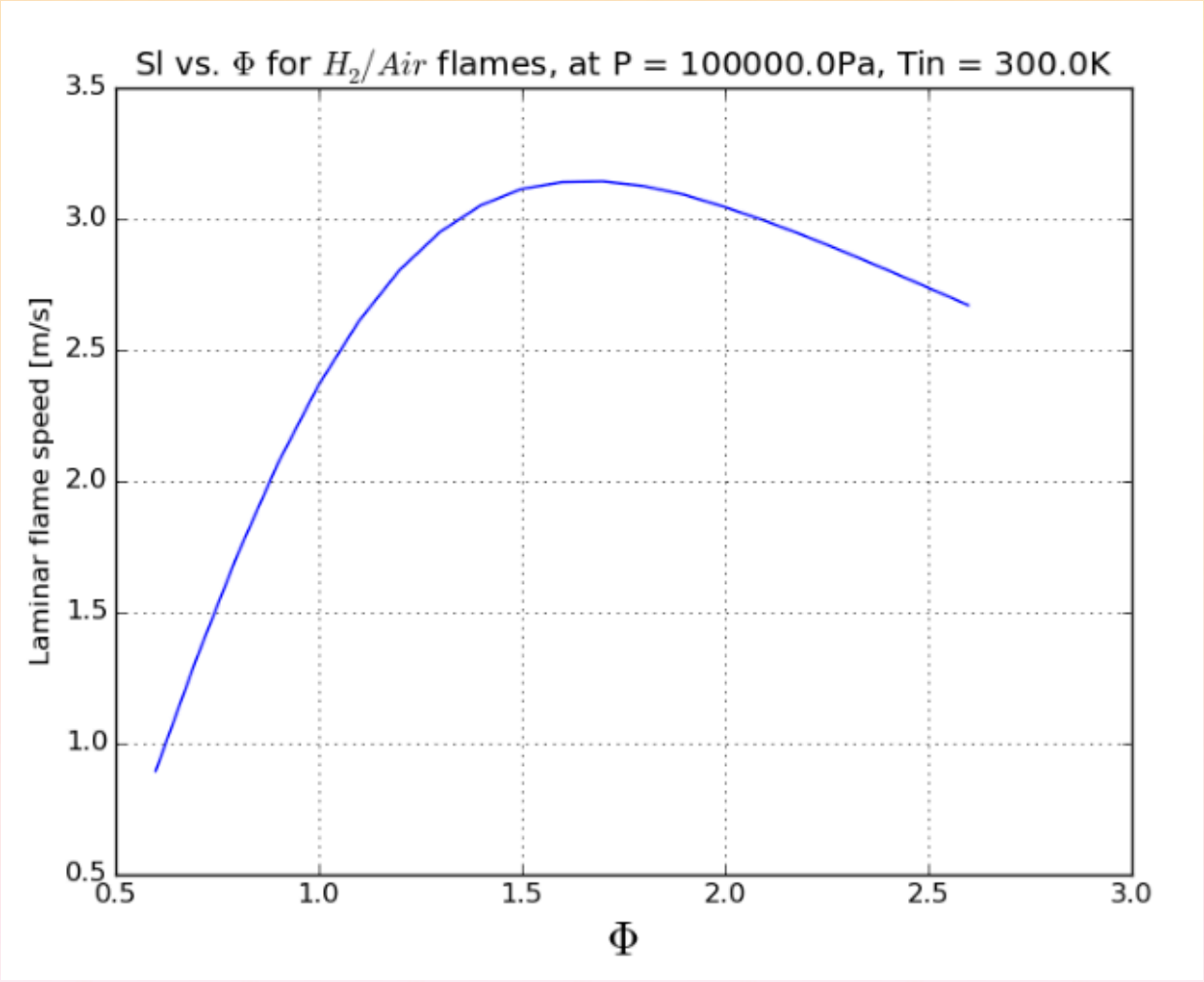
List of errors in the ‘h2_sandiego.cti’ file:

- Ideal_gas : field bar is not a string format line 9
- Reaction 21 is not commented out line 237
- The embedded 'Troe' entry in the 'falloff’ field line 223 does not have the right format for its fields. Names of the fields should be specified.

Tutorial 2

2.5.1 Generate a CTML data, Results :

Once the ‘h2_sandiego.cti’ file is corrected …



Tutorial 2

2.5.2 a second example :

In the SANDIEGO_2 folder

```
[felden@Quincy-MacBookPro-CFD-2: Tuto2]$ cd  SANDIEGO_2
[felden@Quincy-MacBookPro-CFD-2: SANDIEGO]$ python  h2_flame.py

Traceback (most recent call last):
  File "h2_flame.py", line 86, in <module>
    f = FreeFlame(gas, initial_grid)
  File "onedim.pyx", line 1013, in cantera._cantera.FreeFlame.__init__
(cantera/_cantera.cpp:53033)
  File "onedim.pyx", line 329, in cantera._cantera._FlowBase.__init__
(cantera/_cantera.cpp:43691)
RuntimeError:
**************************************************************************
CanteraError thrown by setTransport:
unknown transport model.
**************************************************************************
```

Tutorial 2

2.5.3 Generate submechanisms (OPT) :

Generate ‘submechanisms.cti’ next to the ‘gri30.cti’

[felden@Quincy-MacBookPro-CFD-2: Tuto2]$ cd submechanisms
[felden@Quincy-MacBookPro-CFD-2: submechanisms]$ vi submechanisms.cti

```
units(length = "cm", time = "s", quantity = "mol", act_energy = "cal/mol")

ideal_gas(name = 'hydrogen_submech',
        elements = 'H O',
        species = 'gri30:all',
        reactions = 'gri30:all',
        options = ('skip_undeclared_elements',
                'skip_undeclared_species',
                'skip_undeclared_third_bodies'))


ideal_gas(name = 'NOx-less',
        elements = 'gri30:all',
        species = ’ gri30:H2 H O O2 OH H2O …N2 AR C3H7 C3H8 CH2CHO CH3CHO',
        reactions = 'gri30:all',
        options = ('skip_undeclared_elements',
                'skip_undeclared_species',
                'skip_undeclared_third_bodies'))
```

Tutorial 2

2.5.3 Generate submechanisms (OPT), Results:

```
>>> import cantera as ct
>>> gas=ct.Solution(’ submechanisms.cti','hydrogen_submech')
>>> gas()

 hydrogen_submech:
...
                          X             Y          Chem. Pot. / RT
                 ------------  ------------  ------------
                H2             1             1          -917934
                 H             0             0
                 O             0             0
                O2             0             0
                OH             0             0
               H2O             0             0
               HO2             0             0
              H2O2             0             0
```

Tutorial 3

3.2.1 Generate your first equilibrium script:

```
[felden@Quincy-MacBookPro-CFD-2: Tuto3]$ cp ../../Tuto1/gri30.cti .
[felden@Quincy-MacBookPro-CFD-2: Tuto3]$ vi equil_simple.py
# homogeneous equilibrium of a gas.

from cantera import *

# create an object representing the gas phase
gas = Solution('gri30.xml')

# set the initial state
gas.TPX = 300.0, 1.0e05, 'CH4:0.5, O2:1, N2:3.76'

# equilibrate the gas holding T and P fixed
gas.equilibrate("TP")

# print a summary of the results
print gas()
```

Don't forget to copy the '.cti' mechanism in your current folder

To compute an equilibrium state, two independant intensive properties of the system have to be specified.

Tutorial 3

3.2.1 Generate your first equilibrium script (Results) :

```
[felden@Quincy-MacBookPro-CFD-2: Tuto3]$ python equil_simple.py

    ...
    gri30:

      temperature             300  K
         pressure          100000  Pa
          density         1.10784  kg/m^3
 mean mol. weight         27.6332  amu

                       1 kg             1 kmol
                  -----------      ------------
        enthalpy    -3.01529e+06      -8.332e+07     J
 internal energy    -3.10555e+06      -8.582e+07     J
         entropy         7233.97       1.999e+05     J/K
  Gibbs function    -5.18548e+06      -1.433e+08     J
heat capacity c_p          1111.3       3.071e+04     J/K
heat capacity c_v          810.419       2.239e+04     J/K
```

Tutorial 3

3.2.2 The 'element potential' solver :

[felden@Quincy-MacBookPro-CFD-2: Tuto3]$ cd ChemEquil
[felden@Quincy-MacBookPro-CFD-2: ChemEquil]$ python ChemEquil.py
Computing Equilibirum at Phi = 1, T = 400.0 K, P = 100000.0 Pa
Equilibrate holding TP constants
Using different solvers»

. . .

ChemEquil solver failed! Try the vcs solver...

Tutorial 3

3.2.2 The 'element potential' solver :

[felden@Quincy-MacBookPro-CFD-2: ChemEquil]$ vi ChemEquil.py

```
#Equilibrium:

try:
    gas.equilibrate('TP', solver = 'element_potential')
except:
    print ""
    print "ChemEquil solver failed! Try the vcs solver..."
    #exit()
    #gas.equilibrate('TP', solver = 'vcs')
```

Fails

Uncomment

»

Tutorial 3

3.2.2 The 'element potential' solver (Results) :

[felden@Quincy-MacBookPro-CFD-2: ChemEquil]$ vi ChemEquil.py

****

Final state :

Tadiabatique = 400.0 K

****

P = 1.0000e+05 Pa
T = 4.0000e+02 K
V = 1.2035e+00 m3/kg
U = -3.0233e+06 J/kg
H = -2.9029e+06 J/kg
S = 7.5571e+03 J/kg/K

Comparison between Chem potentials and element potentials:

mu_H2 : **-2.1927e+08 , -2.1927e+08**
mu_O2 : -2.0836e+08 , -2.0836e+08
mu_OH : -2.1382e+08 , -2.1382e+08
mu_H2O : -3.2345e+08 , -3.2345e+08
Output written to all_mole_fractions.csv

When equilibrium is reached, the chemical potential of H2 should equal twice the chemical potential of H

Tutorial 3

3.2.3 Compare different solvers (OPT) :

[felden@Quincy-MacBookPro-CFD-2: SolverComparison]$ vi SolverComparison.py

«

```
#Script SolverComparison.py:

#gas.equilibrate('TP', solver = 'element_potential') # ChemEquil solver
gas.equilibrate('TP', solver = 'vcs')                # vcs solver
#gas.equilibrate('TP', solver = 'gibbs')             # gibbs solver
```

»

Ex of comparison :

'element_potential' :		'gibbs' :	
real	0m0.291s	real	0m0.324s
user	0m0.172s	user	0m0.202s
sys	0m0.033s	sys	0m0.035s

Tutorial 3

3.2.4 Perform adiabatic fame calculations :

```
[felden@Quincy-MacBookPro-CFD-2: AdiabaticFlameT]$ ../../Tuto1/CK2CTI/mech.cti .
[felden@Quincy-MacBookPro-CFD-2: AdiabaticFlameT]$ vi AdiabaticFlameT.py
```

Generated in the first tutorial

```
# Choose the equivalence ratio range range :
phi_min     =       0.3
phi_max     =       3.5
Npoints     =       50
            . . .
phi = np.zeros(npoints)
tad = np.zeros(npoints)
            . . .
# Start the loop on Phi :
for i in range(npoints):
            X[ifuel]    =       ph[i]
            X[io2]      =       stoich_O2
            X[in2]      =       X[io2]*air_N2_O2_ratio
```

Generate an array for the adiabatic temperature

Set the stoichiometry

Tutorial 3

3.2.4 Perform adiabatic fame calculations (Results) :

[felden@Quincy-MacBookPro-CFD-2: AdiabaticFlameT]$ python AdiabaticFlameT.py

```
. . .
At phi =       0.3, Tad      =      1143
At phi =    0.3653, Tad      =      1300
At phi =    0.4306, Tad      =      1451
At phi =    0.4959, Tad      =      1596
At phi =    0.5612, Tad      =      1734
. . .
```

Output written to adiabatic.csv

Uncomment the line :
#print "Output written to", "%s"%(csv_file)

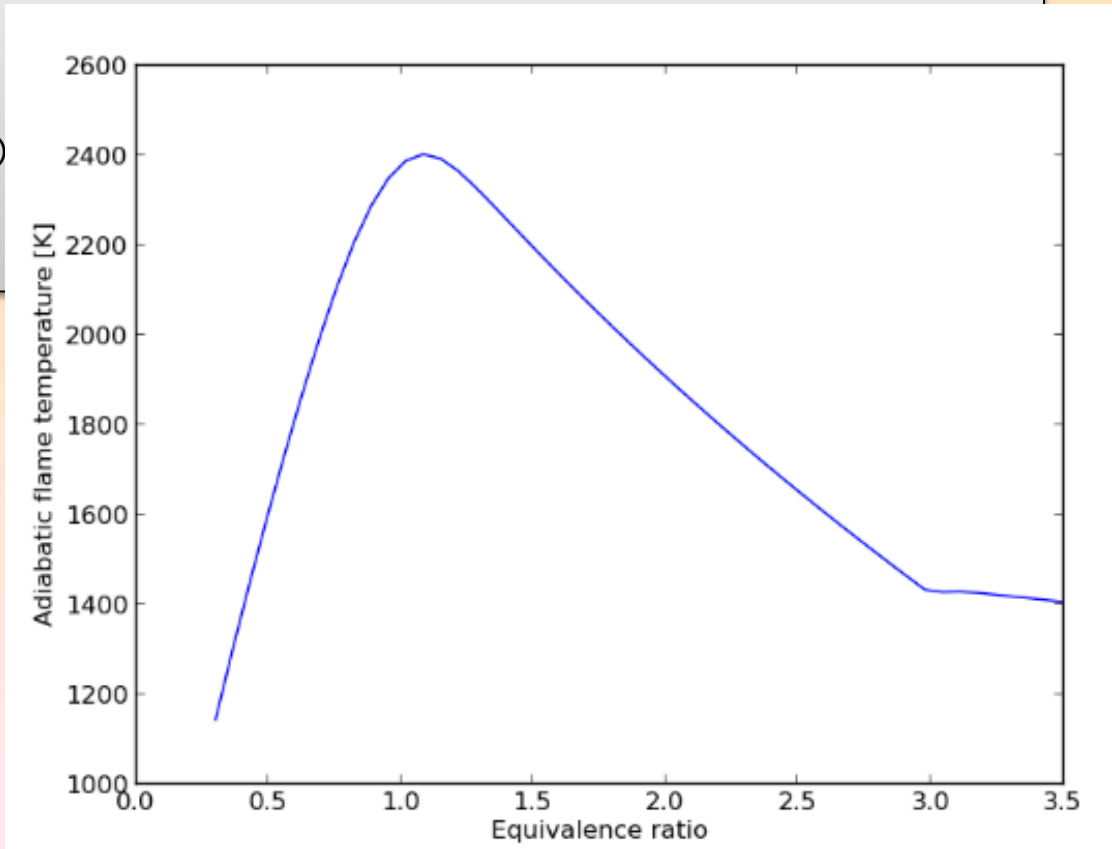
Tutorial 3

3.2.4 Perform adiabatic fame calculations (Results) :

```
# Script AdiabaticFlameT.py : Plot results

import matplotlib.pyplot as plt
plt.plot(phi, tad)
plt.xlabel('Equivalence ratio')
plt.ylabel('Adiabatic flame temperature [K]')
plt.show()
#savefig(...)
```

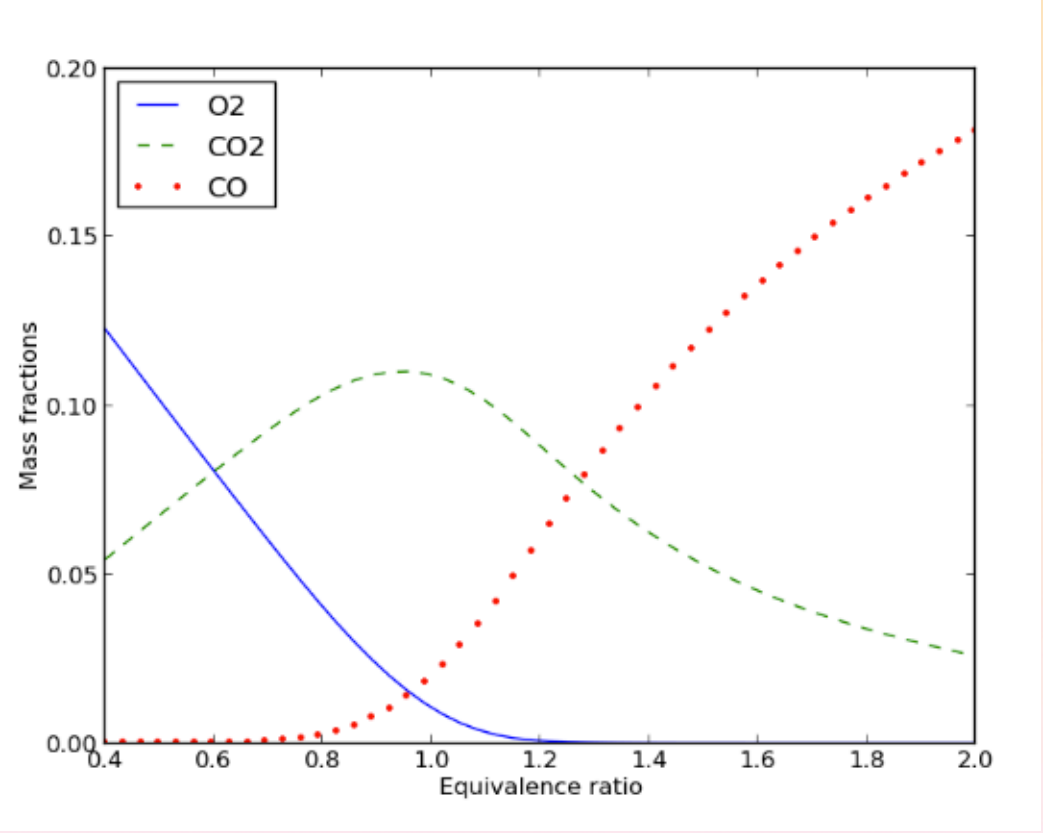
Will save the plot instead of printing it



Tutorial 3

3.2.4 Perform adiabatic fame calculations (Results) :

[felden@Quincy-MacBookPro-CFD-2: AdiabaticFlameT]$ vi AdiabaticFlameT.py
[felden@Quincy-MacBookPro-CFD-2: AdiabaticFlameT]$ python AdiabaticFlameT.py –plot



Tutorial 4

4.2.1 A simple clossed vessel:

[felden@Quincy-MacBookPro-CFD-2: Tuto4]$ cd ClossedVessel
[felden@Quincy-MacBookPro-CFD-2: ClossedVessel]$ vi reactorUV.py

...
import cantera as ct
...
#Create an ideal gas Reactor and fill with gas
r = **ct.IdealGasReactor(gas)**

4.2.1 A simple clossed vessel (Results):

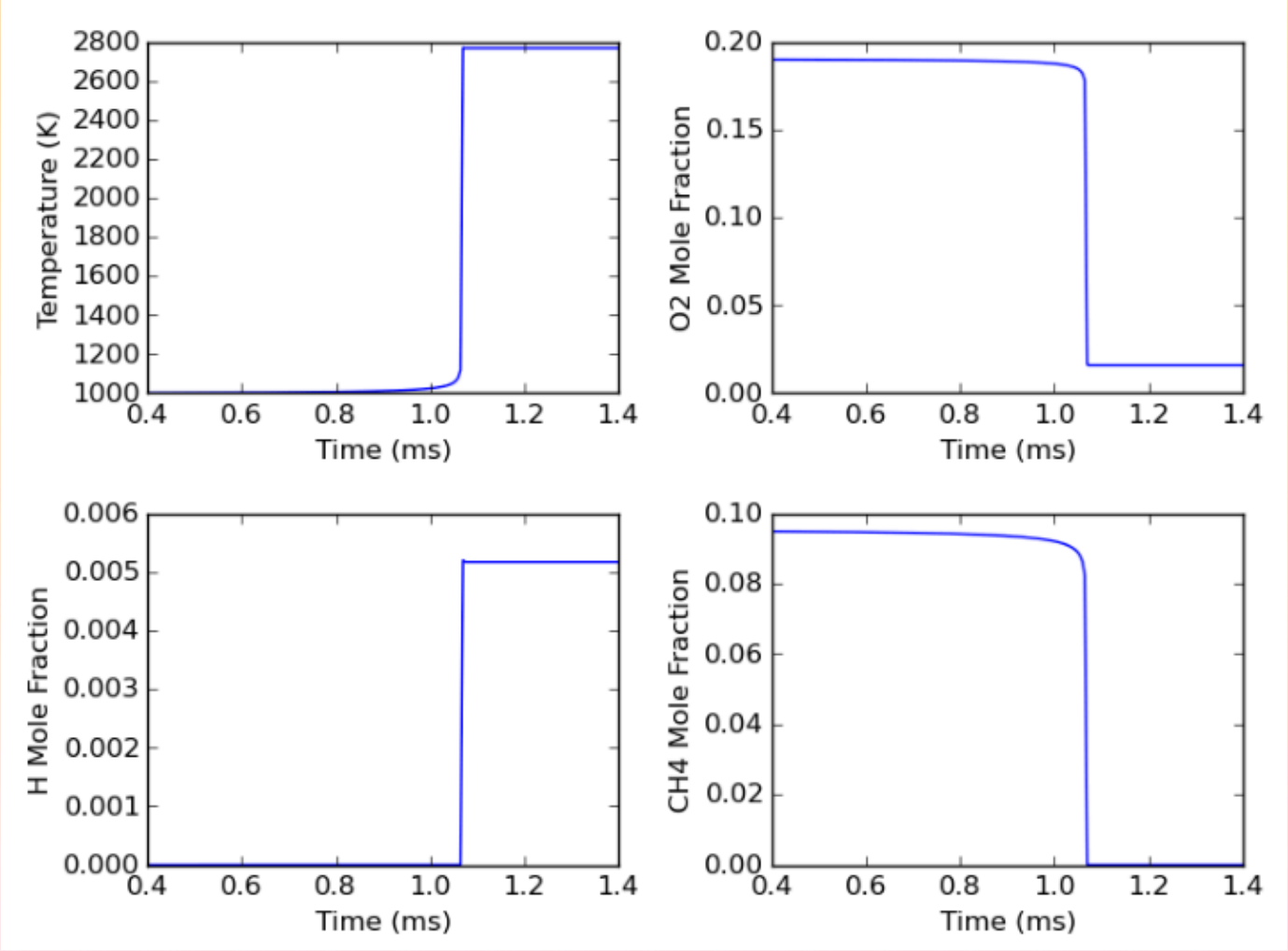
[felden@Quincy-MacBookPro-CFD-2: ClossedVessel]$ python reactorUV.py –plot

t [s]	T [K]	vol [m3]	u [J/kg]
4.050e-01	1001.112	2.970	2.870655e+05
4.100e-01	1001.140	2.970	2.870655e+05
4.150e-01	1001.168	2.970	2.870655e+05
....			
1.395e+00	2768.160	2.970	2.870655e+05
1.400e+00	2768.160	2.970	2.870655e+05

Notice that the internal volume and energy stay constant, as required by the method used.

Tutorial 4

4.2.1 A simple clossed vessel (Results):



Tutorial 4

4.2.2 A simple constant pressure reactor (Results) :

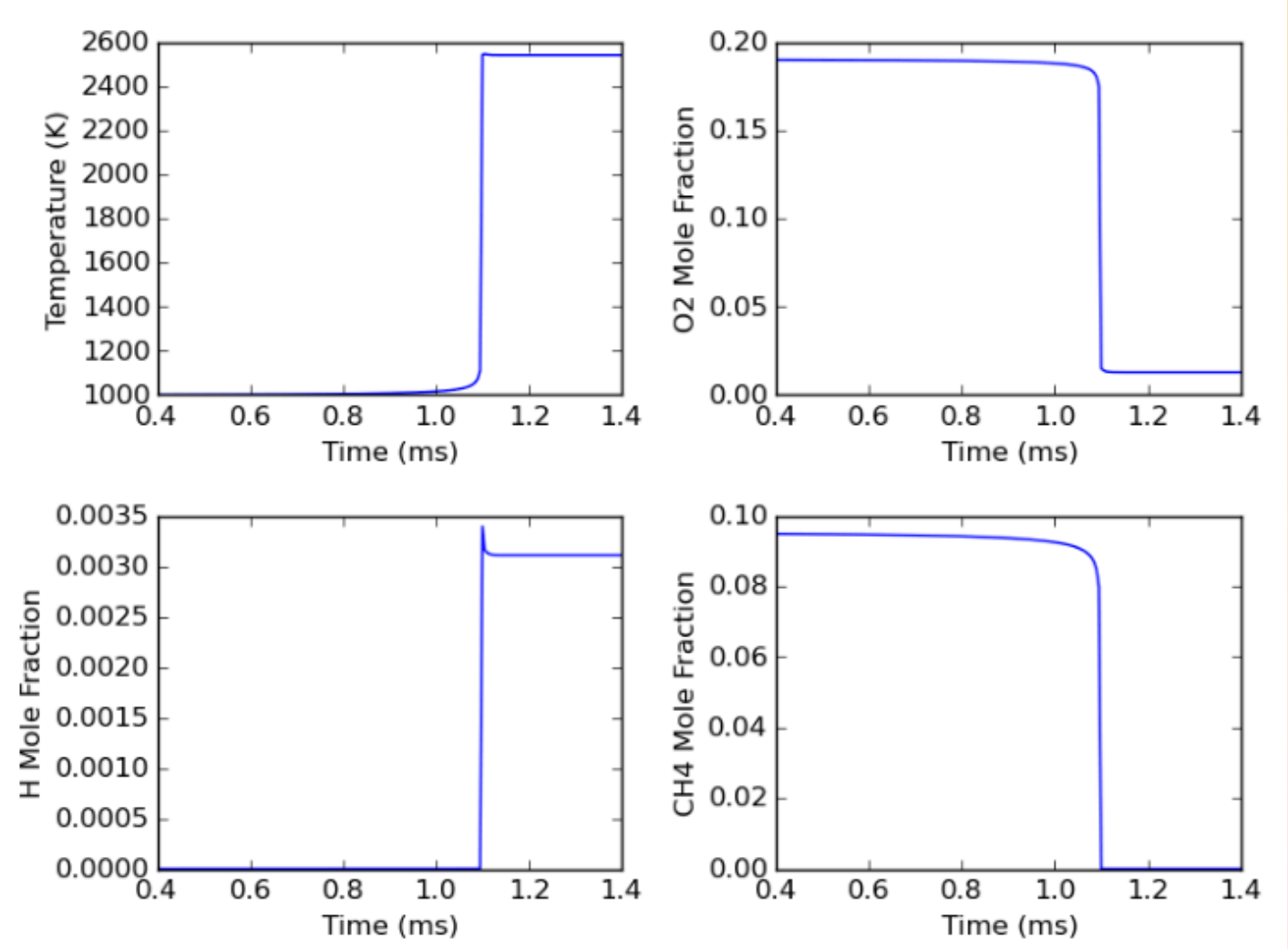
[felden@Quincy-MacBookPro-CFD-2: Tuto4]$ cd ConstantPressureReactor
[felden@Quincy-MacBookPro-CFD-2: ConstantPressureReactor]$ python reactorHP.py –plot

t [s]	T [K]	P [Pa]	h [J/kg]
4.050e-01	1000.862	101325.000	5.879518e+05
4.100e-01	1000.883	101325.000	5.879518e+05
4.150e-01	1000.905	101325.000	5.879518e+05
…			
1.395e+00	2541.147	101325.000	5.879520e+05
1.400e+00	2541.147	101325.000	5.879520e+05

Notice that the pressure and ethalpy stay constant, as required by the method used.

Tutorial 4

4.2.2 A simple constant pressure reactor (Results) :



Tutorial 4

4.2.2 A simple constant pressure reactor, to go further:

```
#Create Reactors used for the process and initialize them
r = ct.IdealGasReactor(gri3)
env = ct.Reservoir(air)
r=ct.IdealGasConstPressureReactor(gri3)

# Define a wall between the reactor and the environment, and
# make it flexible, so that the pressure in the reactor is held
# at the environment pressure.
w = ct.Wall(r, env)
w.expansion_rate_coeff = 1.0e6
w.area = 1.0
```

Tutorial 4

4.2.4 Auto-ignition timing (Results):

```
[felden@Quincy-MacBookPro-CFD-2: Tuto4]$ cd Autoignition
[felden@Quincy-MacBookPro-CFD-2: Autoignition]$ vi Autoignition.py
                ...
[felden@Quincy-MacBookPro-CFD-2: Autoignition]$ python Autoignition.py
WARNING: 'class GRI_30_Kinetics' is deprecated. To be removed in Cantera 2.2.
Autoignition time = 0.0215485
output written to Phi-1_P-1_Trange_UV.csv
```

Tutorial 4

4.2.4 Auto-ignition timing, various initial T :

[felden@Quincy-MacBookPro-CFD-2: Autoignition]$ cp Autoignition.py Autoignition_Tvar.py
[felden@Quincy-MacBookPro-CFD-2: Autoignition]$ vi Autoignition_Tvar.py

- **Replace the line : «** gas.TPX = 1250, one_atm, 'CH4:0.5,O2:1,N2:3.76 **» by**

```
#Initial temperatures
Tmin        = 0.65 # Kelvin
Tmax        = 0.85 # Kelvin
npoints     = 11
```

- **Modify the storage option :**

```
#Temperature storage variables
Ti = np.zeros(npoints,'d')
Ti2 = np.zeros(npoints,'d')
...
#Additional storage variables are needed to differentiate each case
Autoignition_cas = np.zeros(npoints,'d')
FinalTemp_cas = np.zeros(npoints,'d')
mfrac_cas = np.zeros([npoints,gas.n_species],'d')
```

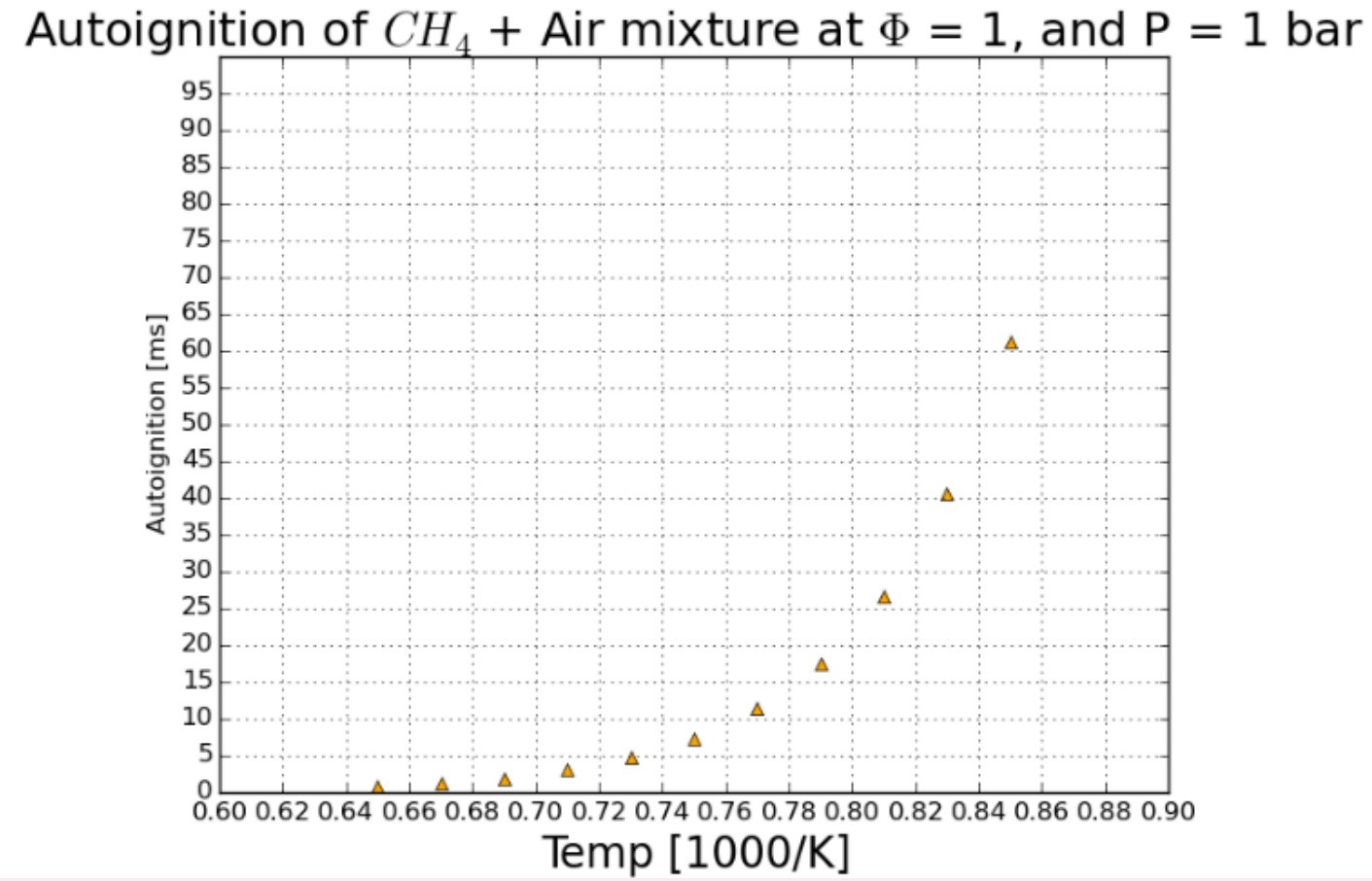
Storage variables now become storage vectors, storage vectors are now storage matrices …

- **Remove the rest of the script, and loop over the several initial conditions**

Tutorial 4

4.2.4 Auto-ignition timing, various initial T (Results) :

[felden@Quincy-MacBookPro-CFD-2: Autoignition]$ python Autoignition_Tvar.py
For 1538.46153846, Autoignition time = 0.0008075
…
For 1176.47058824, Autoignition time = 0.0639745
output written to Phi-1_P-1_Trange_UV.csv



Tutorial 5

5.2 General structure of a 1D simulation :

```
[felden@Quincy-MacBookPro-CFD-2: Tuto5]$  cd 1DpremixedFlame
[felden@Quincy-MacBookPro-CFD-2: 1DpremixedFlame]$  vi premixed_flame.py

#import :
from cantera import *
from numpy import *

#######################################################################
# Prepare your run
#######################################################################

#Import gas phases with mixture transport model
gas = Solution(...)

#Parameter values :
      #General
p             =  1e5                  # pressure
tin           =  300.0                # unburned gas temperature
```

1) Import your packages

2) Import your gaseous object

3) Set your parameters

Tutorial 5

5.2 General structure of a 1D simulation :

[felden@Quincy-MacBookPro-CFD-2: 1DpremixedFlame]$ vi premixed_flame.py

```
#Stoechiometry :
phi                 =   ...
stoich_O2           = ...
air_N2_O2_ratio     = 3.76

fuel_species        = 'CH4'
ifuel               = gas.species_index(fuel_species)
io2                 = gas.species_index('O2')
in2                 = ...

m                   = gas.n_species
x                   = zeros(m,'d')
x[ifuel]    = ...
x[io2]      = ...
x[in2]      = ...

gas.TPX     = tin, p, x
```

4) Set your gaseous composition (other methods exists ! See previous tutorials …)

Tutorial 5

5.5.1 A premixed flat flame simulation :

[felden@Quincy-MacBookPro-CFD-2: 1DpremixedFlame]$ vi premixed_flame.py

5) Initialize a grid

```
#Initial grid
initial_grid = 2*array([0.0, 0.001, 0.01, 0.02, 0.029, 0.03],'d')/3
```

[felden@Quincy-MacBookPro-CFD-2: 1DpremixedFlame]$ vi premixed_flame.py

```
#Create the free laminar premixed flame
f = FreeFlame(gas, initial_grid)
        #set inlet conditions
f.inlet.X = x
f.inlet.T = tin
        #No energy for starters
f.energy_enabled = False
        #Mixture averaged transport model
f.transport_model = 'Mix'
```

6) Initialize your flame object :

- *With a gaseous composition & a grid*
- *Boundary conditions*
- *A set of equations to solve*

Tutorial 5

5.5.1 A premixed flat flame simulation :

[felden@Quincy-MacBookPro-CFD-2: 1DpremixedFlame]$ vi premixed_flame.py

```
#First flame:
      #No energy for starters, no grid refinement
f.energy_enabled = False
...
refine_grid = False
f.solve(loglevel, refine_grid)
...
```

→ ***Solve a first flame !***

[felden@Quincy-MacBookPro-CFD-2: 1DpremixedFlame]$ python premixed_flame.py

```
.....
Attempt Newton solution of steady-state problem...    failure.
Take 2 timesteps     2.813e-06      2.362
...
Take 20 timesteps      0.02996     -2.651
Attempt Newton solution of steady-state problem...    success.

Problem solved on [7] point grid(s).
.....
grid refinement disabled.
Solution saved to file ch4_adiabatic.xml as solution no energy.
Solution saved to 'ch4_adiabatic.csv'.
```

Tutorial 5

5.5.1 A premixed flat flame simulation :

[felden@Quincy-MacBookPro-CFD-2: 1DpremixedFlame]$ vi premixed_flame.py

```
##################
#Second flame:
      #Energy equation enabled
f.energy_enabled = ...

      #Calculation and save of the results
refine_grid = ...

      #Refinement criteria when energy equation is enabled
f.set_refine_criteria(ratio = …, slope = …, curve =…)

f.solve(loglevel, refine_grid)

f.save('ch4_adiabatic.xml','energy',
     'solution with the energy equation enabled')

      #See the sl to get an idea of whether or not you should continue
print('mixture-averaged flamespeed = {0:7f} m/s'.format(f.u[0])) #m/s
```

- ***Enable the energy equ. & refine the grid***
- ***Set the refinment cirteria***

→ ***Solve a 2nd flame ! …***

→ ***And print sl on screen***

Tutorial 5

5.5.1 A premixed flat flame simulation :

```
[felden@Quincy-MacBookPro-CFD-2: 1DpremixedFlame]$ python premixed_flame.py
...
###############################################################################
Refining grid in flame.
   New points inserted after grid points 16 17 20 21 22 23 24
   to resolve C C2H C2H2 C2H3 C2H4 C2H5 C2H6 C3H7 C3H8 CH CH2 CH2(S) CH2CO CH2OH CH3
CH3CHO CH3O HCCO HCCOH HCN HCO N
###############################################################################


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

Problem solved on [49] point grid(s).
.....................................................................

no new points needed in flame
Solution saved to file ch4_adiabatic.xml as solution energy.
mixture-averaged flamespeed = 0.389835 m/s
Solution saved to 'ch4_adiabatic.csv'.
```

Tutorial 5

5.5.1 A premixed flat flame simulation :

Refine your grid and continue your simulation until you feel confident your flame has converged …

How ?

- Check that the printed flame speed has converged
- Uncomment the « f.write_csv('ch4_adiabatic.csv', quiet=False) » part of your script and plot the temperature for example … You should have enough points in region og high gradients !
- Usually, ~200 points is enough for a 1D premixed simulation