Cantera tutorial-V2.1

Benedetta Franzelli, Jean-Philippe Rocchi, Pierre Wolf
CERFACS, 42 Avenue Gaspard Coriolis, 31057 Toulouse Cedex, France
franzell@cerfacs.fr
modified by Pierre Wolf
wolf@cerfacs.fr

modified December 2, 2010

Contents

This documents explains quickly how to use CANTERA at CERFACS.

1 What is CANTERA?

Cantera is an open-source, object-oriented software package for problems involving chemically-reacting flows. Capabilities include:

- multiphase chemical equilibrium
- thermodynamic and transport properties
- homogeneous and heterogeneous kinetics
- reactor networks
- one-dimensional flames
- reaction path diagrams
- interfaces for MATLAB, Python, C++, and Fortran (see [1])

Some informations are given at http://www.cantera.org
Many modules are usable on all platforms (MAC OS X, Linux & Windows).
It is downloadable on the web-site.
CANTERA can be directly used at CERFACS with nuage.

2 Download and compile CANTERA

2.1 Download for basic users

- to get the last version on the network:

  ```
  svn checkout file:///home/cfd2/cantera/svn/CANTERA/
  tags/cantera-1.8.0-beta-AVBP1.0 ./cantera-1.8.0-beta-AVBP1.0
  ```

- to get the last version on a MAC:

  ```
  svn checkout svn+ssh://LOGIN@nuage/home/cfd2/cantera/svn/CANTERA/
  tags/cantera-1.8.0-beta-AVBP1.0 ./cantera-1.8.0-beta-AVBP1.0
  ```

2.2 Download for developpers

Developers will check out the last version under development in the trunk folder (not in the tags folder)

- to get the last version on the network:

  ```
  svn checkout file:///home/cfd2/cantera/svn/CANTERA/
  trunk/cantera-1.8.0-beta ./cantera-1.8.0-beta
  ```

- to get the last version on a MAC:

  ```
  svn checkout svn+ssh://LOGIN@nuage/home/cfd2/cantera/svn/CANTERA/
  trunk/cantera-1.8.0-beta ./cantera-1.8.0-beta
  ```

A commit will update only the version in the trunk folder (dowload version), and keep the tags folder clean. Once a version ready to be tagged, the version from trunk is copied in the tags folder.

2.3 compilation on a MACINTOSH

1. in

   ```
   ./Cantera/cxx/demos/Makefile.in
   ```

   replace

   ```
   @INSTALL@ Makefile -m ug+rw,o+r @ct_demodir@/cxx
   ```

by

```
@INSTALL@ Makefile @ct_demodir@/cxx
```

2. in ./configure replace

   ```
    dir1="\$NUMPY_HOME/include/python"
   ```

   by

   ```
    dir1="\$NUMPY_HOME"
   ```

3. in ./preconfig replace ”n” by ”y” at the line to choose the package NUMPY

   ```
   USE_NUMEPY=\${USE_NUMPY:="y"}
   ```

4. in ./preconfig give the address of NUMPY package

   ```
    NUMPY_HOME=${NUMPY_HOME:="THE_ADDRESS"}
   ```

   Do not forget to deleted the # symbol.
   A trick to get this address is to find the file arrayobject.h on your MAC.Then, it might be:

   ```
   THE_ADDRESS/numpy/arrayobject.h
   ```

5. in ./preconfig choose the directory where Cantera will be install

   ```
   CANTERA_CONFIG_PREFIX=\${CANTERA_CONFIG_PREFIX:="INSTALL_DIRECTORY"}
   ```

6. then, launch

   ```
   ./preconfig
   make
   make install
   ```

2.4 compilation on a UNIX machine

1. open

   ```
    ./tools/scr/finish_install.py.in
   ```

   replace lib/python by lib64/python

2. open preconfig replace ”n” by ”y” at the line to choose the package NUMERIC

   ```
   USE_NUMERIC=\${USE_NUMERIC:="y"}
   ```

3. in `./preconfig` choose the directory where Cantera will be install

   ```
   CANTERA_CONFIG_PREFIX=\${CANTERA_CONFIG_PREFIX:="INSTALL_DIRECTORY"}
   ```

4. then, launch

   ```
   ./preconfig
   make
   make install
   ```

3 Computing a 1D premixed Flame

This first example uses c++ routine.

3.1 How to run the program

To run a Cantera program:

1. go to your install directory
2. go to the directory

   ```
   ./demos/cxx/flamespeed/
   ```

3. change the parameter of the flame you want you create in the file

   ```
   flamespeed.cpp
   ```

4. verify that `setup_cantera` is correctly filled. Normally, it must not be changed it. It should looks like that:

   ```
   #!/bin/sh
   LD_LIBRARY_PATH=INSTALL_DIRECTORY/cantera-1.80/lib:$LD_LIBRARY_PATH
   export LD_LIBRARY_PATH
   PATH=INSTALL_DIRECTORY/cantera-1.80/bin:$PATH
   export PATH
   PYTHON_CMD=/usr/bin/python2
   export PYTHON_CMD
   alias ctpython=/usr/bin/python2
   ```

5. source the file `setup_cantera`

   ```
   source \~/setup_cantera
   ```

6. copy the 1D premixed flame example

   ```
   cp /home/cfd2/cantera/CANTERA/INSTALL_DIR/cantera-1.80/demos/cxx/flamespeed/flamespeed
   ```

7. fill in properly the file `flamespeed.cpp` (Section 3.2)
8. compile this c++ file. To do that, a make file is needed. The `demo.mak` is an example of makefile, it must be copied, renamed and filled in properly.

```
cp /home/cfd2/cantera/CANTERA/INSTALL_DIR/cantera-1.80/demos/cxx/demo.mak .
```

In the file `flamespeed.mak`, the name of the executable and the object name must be chosen:

```
 # the name of the executable program to be created
PROG_NAME = flamespeed

# the object files to be linked together.
OBJS = flamespeed.o
```

Moreover, the libraries and include files directory:

```
 # the directory where the Cantera libraries are located
CANTERA_LIBDIR=/home/cfd2/cantera/CANTERA/INSTALL_DIR/cantera-1.80/lib

# the directory where Cantera include files may be found.
CANTERA_INCDIR=/home/cfd2/cantera/CANTERA/INSTALL_DIR/cantera-1.80/include
```

Then, everything is compiled by typing:

```
make -f  flamespeed.mak
```

It will create the executable named `flamespeed`. It can be executed by typing: `./flamespeed`

By default, solutions are written in CSV format, they can be opened with Microsoft excel or OpenOffice Spreadsheet.

3.2 What the program does

Studied gas (Section 3.2.1), domain built (Section 3.2.2), solution and its visualization (Section 3.2.3) are ruled by `flamespeed.cpp`.

3.2.1 Characterizing gas

To describe the gas:

1. choose the chemical scheme corresponding to studied fuel:

```
IdealGasMix gas("gri30.cti","gri30_mix");
```

The first parameter is the name of the file that contains the chemical scheme. The second parameter is the identification name of the chemical scheme. The `gri30_mix` use the GRI-Mech 3.0 mechanism that contains 325 reactions and 53 species (see [2]). A set of chemical scheme (files `.cti`) could be found in directory /home/cfd2/cantera/CANTERA/INSTALL_DIR/cantera-1.80/data. User could also write his own chemical scheme in Cantera format (see Section 6) or some chemical scheme in Chemkin format could be converted, using the `ck2cti` utility program.

2. choose inlet temperature, pressure and velocity:

```
doublereal temp = 300.0; // K
doublereal pressure = 1.0*OneAtm; //atm
doublereal uin=0.3; //m/sec
```

3. choose inlet gas composition. Cantera allows to set state by specifying the value of one quantity from each of the three columns in Table 1. The methods to set the state have names that begin with `setState_`, which is followed by three uppercase letters. The letters denote the properties that are being specied and they must be in the same order as the columns of the table (see [3]). For example, for set the

Temperature (T)	Density (R)	Mole Fractions (X)
Specific Enthalpy (H)	Pressure (P)	Mass Fractions (Y)
Specific Entropy (S)		

Table 1: Quantity to set the gas composition.

temperature, pressure and mole fractions it is called the method:

```
gas.setState_TPX(temp, pressure, "CH4:1.0, O2:2.0, N2:7.52").
```

Note that the mole fractions will be updated when the equivalence ratio is introduced.

After these operations, the `equilibrate(gas,"HP")` function is called to set the gas to a state of chemical equilibrium, holding fixed specific enthalpy and pressure. The value of density, temperature and mass fractions before the `equilibrate` function will be imposed at the inlet and will be used to construct the initial guess. The values at the equilibrium state will be used to construct the initial guess.

3.2.2 Building domain

Generally they must not be changed:

1. the definition of 1D premixed flame and specification of the objects to use to compute kinetic rates and transport properties:

   ```
   FreeFlame flow(&gas);
   Transport* tr = newTransportMgr("Mix", &gas);
   flow.setTransport(*tr);
   flow.setKinetics(gas);
   flow.setPressure(pressure);
   ```

   Here the "Mix" transport class has been chosen. To set the "AVBP" transport class, the following line should be used:

   ```
   Transport* tr = newTransportMgr("AVBP", &gas);
   ```

   Please, be sure that you are using the canteraAVBP version. Verify it in your makefile:

   ```
    # the directory where the Cantera libraries are located
   CANTERA_LIBDIR=/home/cfd2/cantera/CANTERA/INSTALL_DIR/canteraAVBP/lib

   # the directory where Cantera include files may be found.
   CANTERA_INCDIR=/home/cfd2/cantera/CANTERA/INSTALL_DIR/canteraAVBP/include
   ```

2. the creation of inlet (imposing mole fractions, momentum and temperature) and outlet:

   ```
   Inlet1D inlet;
   inlet.setMoleFractions(DATA_PTR(x));
   inlet.setMdot(mdot);
   inlet.setTemperature(temp);
   Outlet1D outlet;
   ```

The length of domain `lz`, the number of points `nz` and their location `z` could be chosen. An additional node is added at the end of domain to help calculation in case of zero gradient at the outlet. Then an initial grid is created:

```
flow.setupGrid(nz, z);
```

3.2.3 Creating solutions

Domain and solution are created:

1. domain is characterized by inlet, outlet and flow and it is inserted in a container:

   ```
   vector<Domain1D*> domains;
   domains.push_back(&inlet);
   domains.push_back(&flow);
   domains.push_back(&outlet);
   Sim1D flame(domains);
   ```

2. initial guess for temperature, velocity and mass fractions is supplied. Ramp values from inlet to adiabatic flame conditions over 70% of domain are assumed and then level off at equilibrium. The percentage could be chosen. The initial guess could be visualize to check its correctness:

   ```
   flame.showSolution();
   ```

3. grid refinement criteria must be specified:

   ```
   double ratio=10.0;
   double slope=0.2;
   double curve=0.02;
   double prune=-0.00005;
   flame.setRefineCriteria(flowdomain,ratio,slope,curve,prune);
   ```

   The ratio parameter controls the maximum size ratio between adjacent cells; slope and curve should be between 0 and 1 and control adding points in regions of high gradients and high curvature, respectively. If prune is positive, points will be removed if the relative slope and curvature for all components fall below the prune level.

4. Node for fixed temperature point is added. The energy equation is disabled and the problem is solved without refining the grid:

   ```
   flow.fixTemperature();
   refine_grid=false;
   flame.setFixedTemperature(900.0);
   flame.solve(loglevel,refine_grid);
   ```

5. The energy equation is then turned on and the problem is solved again:

```
flow.solveEnergyEqn();
flame.solve(loglevel,refine_grid);
```

6. Temperature, velocity and mole fractions species are visualized as functions of the spatial variable. Finally adiabatic flame temperature and flame speed are visualized:

   ```
   Adiabatic flame temperature from equilibrium is: 2225.32
   Flame speed for phi=1 is 0.39228 m/s.
   ```

3.3 Some useful modifications

The source of these examples could be found in directory:

`/home/cfd2/cantera/CANTERA/EXAMPLES/PREMIXED.`

3.3.1 Saving solution in TECPLOT or IGOR format

To save temperature, velocity, mole fractions and reaction rates in a TECPLOT, CSV or IGOR format :

1. copy file `example_utils.h` in the work directory

   ```
   cp /home/cfd2/cantera/CANTERA/CERFACS_ADDED/include/example_utils.h .
   ```

2. open the `flamespeed.cpp` file and insert the command `#include "example_utils.h"`:

   ```
   #include <cantera/Cantera.h>
   #include <cantera/onedim.h>
   ..
   #include <cantera/transport.h>
   #include "example_utils.h"
   ```

3. modify function `flamespeed` creating a 2D array to hold the output variables, and store the values for the initial state just before the last for-cicle:

   ```
   printf("\n%9s\t%8s\t%5s\t%7s\n","z (m)", "T (K)", "U (m/s)", "Y(CO)");
   Array2D soln(2*nsp+5, 1);
   for(int n=0;n<np;n++){
       ...
   }
   ```

4. in that for-cicle add a command to save solution to make a TECPLOT data file

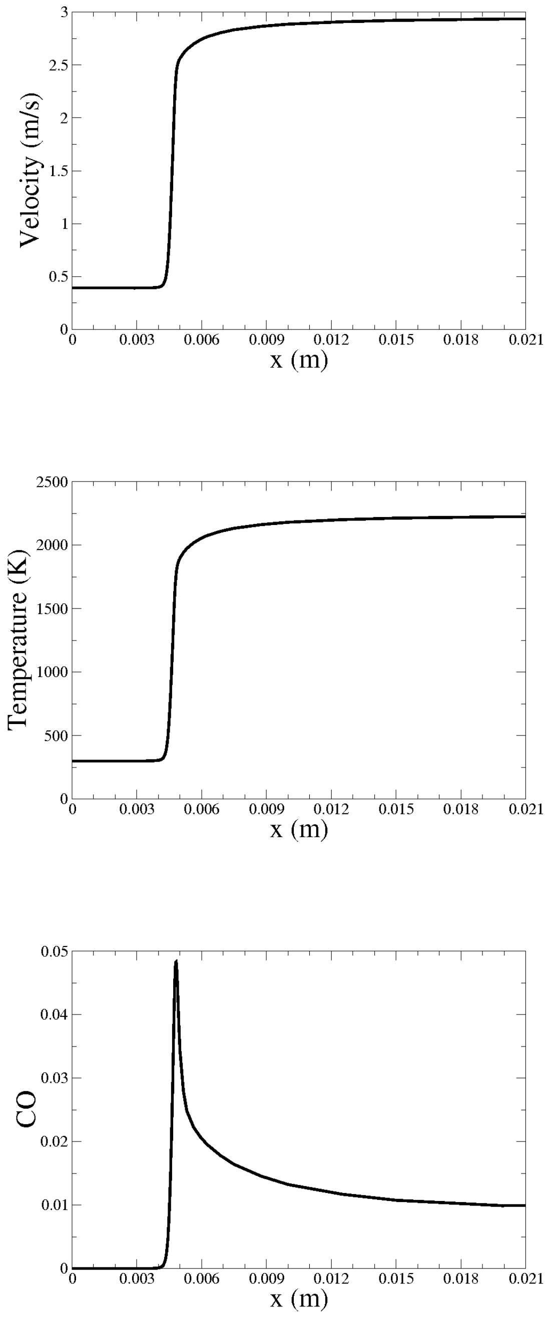


Figure 1: Velocity, temperature and mole fraction of CO for equivalence ratio $\phi = 1.0$

```
for(int n=0;n<np;n++){
    ...
  flow.setGas(flame.solution() + flame.start(flowdomain), n);
  (n == 0) ? saveSoln(0, flow.grid(n), Uvec[n], gas, soln)
  : saveSoln(flow.grid(n), Uvec[n], gas, soln);
}
```

5. after for-cicle define some useful variables:

   ```
   char str_phi[4];
   sprintf(str_phi, "%1.2f", phi);
   string plotTitle = "1D laminar flame phi = ";
   plotTitle += str_phi;
   string filename, extension;
   ```

6. construct the output file (TECPLOT/Excel/IGOR):

   ```
   filename = "soln_phi";
   extension = ".dat";                                  // ".csv" ; ".IGOR"
   filename = filename + "-" + str_phi + extension;
   plotSoln(filename, "TEC", plotTitle, gas, soln);     //  "XL"  ; "IGO"
   ```

NOTE: In file "example_utils.h" to calculate the reaction rates the chemical scheme is called two times:

```
IdealGasMix new_gas("gri30.cti","gri30_mix").
```

If a different chemical scheme is used (for example CM2), this line must be changed:

```
IdealGasMix new_gas("CM2.cti","CM2_mix").
```

3.3.2 Solutions for different equivalence ratio

Sometimes could be useful to memorize into an output file the values of some variables as function of the equivalence ratio ϕ. Modifying file `flamespeed.cpp` it is possible to memorize flame speed, adiabatic temperature and mole fractions of the equilibrium state for different ϕ. The idea is to solve the same problem with different equivalence ratio thanks a for-cicle. To obtain this:

1. modify the parameter of function `flamespeed`:

   ```
   int flamespeed(doublereal& p, ofstream& myresult) {
        ...
   }
   ```

First parameter is the value of equivalence ratio, despite the second parameter is the name of input file where results as function of ϕ are memorized;

2. set the variable `phi` equal to the parameter `p` and delete the first if-state:

```
vector_fp x;
x.resize(nsp);

double phi = p;
/*  if (np > 0) phi = *(double*)(p);
    if (phi == 0.0) {
        cout << "Enter phi: ";
        cin >> phi;
    }
*/
```

3. after the last for-cicle save in the output file some interesting variables (flame speed, adiabatic temperature and mole fractions):

```
for(int n=0;n<np;n++){
    ...
    }
//SAVE SOLUTION AS FUNCTION OF PHI
myresult<<phi<<" ";
myresult<<flame.value(flowdomain,flow.componentIndex("u"),0)<<" ";
myresult<<flame.value(flowdomain,flow.componentIndex("T"),np-1)<<" ";
for(int n=0; n<nsp;n++)
   myresult<<flame.value(flowdomain,n+4,np-1)<<" ";
myresult<<"\n";
```

4. modify the `main` function adding a for-cicle for equivalence ratio between `p_min` and `p_max`:

```
#ifndef CXX_DEMO
int main() {
    doublereal p_min=0.5;
    doublereal p_max=2.6;
    ofstream myresult;
    myresult.open("myresult_phi.dat");

    int error_flag;
    for(doublereal p=p_min;p<p_max; p+=0.1)
```

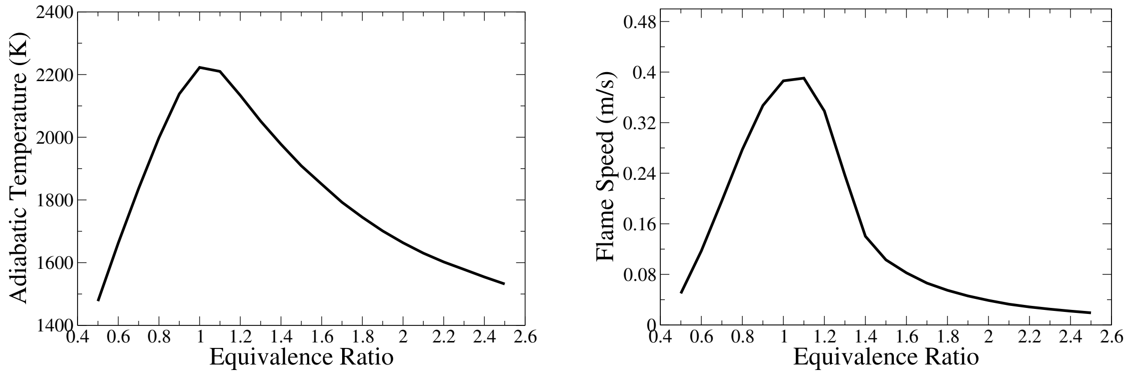


Figure 2: Adiabatic temperature and flame speed as function of ϕ

```
        error_flag=flamespeed(p, myresult);
    myresult.close();
    return error_flag;
}
#endif
```

At each iteration results are saved in file `"myresult_phi.dat"`.

4 Computing a 1D counterflow diffusion Flame

This second example uses python routine.

4.1 How to run the program

To run this example:

1. connect to nuage and work using bash shell
2. create a work directory

   ```
   mkdir MY_SECOND_1D_FLAME
   ```

3. copy /usr/local/cantera/bin/setup_cantera to his work directory

   ```
   cp /usr/local/cantera/bin/setup_cantera MY_SECOND_1D_FLAME
   ```

4. verify that `setup_cantera` is correctly filled. Normally, it must not be changed. It should looks like that:

   ```
   #!/bin/sh
   LD_LIBRARY_PATH=/usr/local/cantera/lib:$LD_LIBRARY_PATH
   export LD_LIBRARY_PATH
   PATH=/usr/local/cantera/bin:$PATH
   export PATH
   PYTHON_CMD=/usr/bin/python2
   export PYTHON_CMD
   alias ctpython=/usr/bin/python2
   ```

5. source the file `setup_cantera`

   ```
   source setup_cantera
   ```

6. copy the 1D nonpremixed counterflow flame example

   ```
   cp /usr/local/cantera/demos/python/flames/npflame1.py .
   ```

7. fill in properly the file `npflame1.py` (Section 4.2)
8. execute it by typing `python npflame1.py`. By default, solutions are written in CSV format, you can open it with Microsoft excel or OpenOffice Spreadsheet.

4.2 What program does

The program is composed by 2 part. Firstly, all the parameter values could be changed (see Section 4.2.1). Secondly, the gas object is created, the flame is solved and saved (see Section 4.2.2).

4.2.1 Parameter values

The parameter values are grouped at beginning to simplify changing flame conditions:

1. the good boundary conditions

```
p          =   OneAtm              # pressure
tin_f      =   300.0               # fuel inlet temperature
tin_o      =   300.0               # oxidizer inlet temperature
mdot_o     =   0.72                # kg/m^2/s
mdot_f     =   0.24                # kg/m^2/s

comp_o       =  'O2:1, N2:3.76';   # air composition
comp_f       =  'CH4:1';           # fuel composition
```

2. the initial grid. By default, distance between inlets is 2 cm and it starts with an evenly-spaced 6-point grid:

```
initial_grid = 0.02*array([0.0, 0.2, 0.4, 0.6, 0.8, 1.0],'d')
```

3. the tolerance values for steady-state problem and for time stepping

```
tol_ss    = [1.0e-5, 1.0e-9]       # [rtol, atol] for steady-state
                                   # problem
tol_ts    = [1.0e-3, 1.0e-9]       # [rtol, atol] for time stepping
```

4.2.2 Creating gas object and solution

Now to create the gas object and to solve the flame:

1. choose the good chemical scheme:

```
gas = GRI30('Mix')
```

By default, `GRI-Mech 3.0` with mixture-averaged transport properties is used. To use another mechanism, function `IdealGasMix('mech.cti')` could be used to read a mechanism in Cantera format.

2. create an object representing the counterflow flame configuration:

   ```
   f = CounterFlame(gas = gas, grid = initial_grid)
   ```

   It consists of a fuel inlet on the left, the flow in the middle, and the oxidizer inlet on the right.

3. set the state of the two inlets and the error tolerances:

   ```
   f.fuel_inlet.set(massflux = mdot_f,
                    mole_fractions = comp_f,
                    temperature = tin_f)
   f.oxidizer_inlet.set(massflux = mdot_o,
                        mole_fractions = comp_o,
                        temperature = tin_o)
   f.set(tol = tol_ss, tol_time = tol_ts)
   ```

4. construct the initial solution estimate. To do so, it is necessary to specify the fuel species:

   ```
   f.init(fuel = 'C2H6')
   ```

   If a fuel mixture is being used, a representative species must be specified here for the purpose of constructing an initial guess.

5. check the correctness of the starting estimate:

   ```
   f.showSolution().
   ```

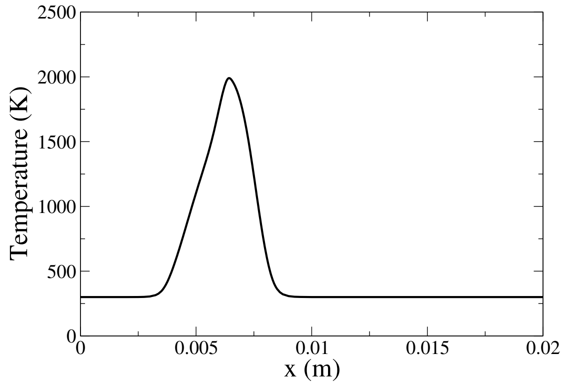
Then, to solve a diffusion flame:

1. the energy equation is disabled and the problem is solved without refining the grid:

   ```
   f.set(energy = 'off')
   f.solve(loglevel, 0)
   ```

2. grid refinement criteria are specified, the energy equation is turned on and problem is solved again:

   ```
   f.setRefineCriteria(ratio = 200.0, slope = 0.1, curve = 0.2, prune = 0.02)
   f.set(energy = 'on')
   f.solve(1)
   ```



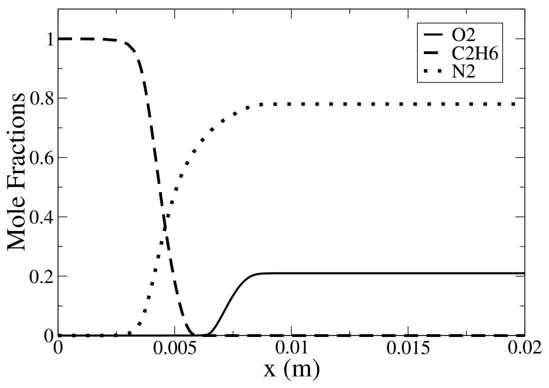


Figure 3: Temperature and mole fractions of O2, C2H6 and N2

3. the solution is saved in a `.xml` format and velocity, temperature and mole fractions are written into a CSV file:

```
f.save('npflame1.xml')
z = f.flame.grid()
T = f.T()
u = f.u()
V = f.V()
fcsv = open('npflame1.csv','w')
writeCSV(fcsv, ['z (m)', 'u (m/s)', 'V (1/s)', 'T (K)']
         + list(gas.speciesNames()))
for n in range(f.flame.nPoints()):
    f.setGasState(n)
    writeCSV(fcsv, [z[n], u[n], V[n], T[n]]+list(gas.moleFractions()))
fcsv.close().
```

4.3 A useful modification: write solution in TECPLOT format

The source of this example could be found in directory:

`/home/cfd2/cantera/CANTERA/EXAMPLES/DIFFUSION.`

To save velocity, temperature and mole fractions TECPLOT format:

1. copy directory MODULE_PYTHON in the work directory:

```
cp -r /home/cfd2/cantera/CANTERA/CERFACS_ADDED/include/MODULE_PYTHON .
```

2. add the directory MODULE_PYTHON to `npflame1.py` file and import the module which contains function `writeTEC()`:

   ```
   from MODULE_PYTHON import *

   from Cantera import *
   from Cantera.OneD import *
   from Cantera.OneD.CounterFlame import CounterFlame
   from Cantera.num import array
   ```

3. add these lines at the end of the code:

   ```
   # generate the file for visualization with TECPLOT
   fdat = open('diffusion_flame.dat','w')
   title = 'diffusion_flame plot'
   writeTEC(fdat,['TITLE     =','"', title,'"'] )
   writeTEC(fdat,['VARIABLES ='])
   writeTEC(fdat,['"space (m)"'])
   writeTEC(fdat,['"u (m/s)"'])
   writeTEC(fdat,['"V (1/s)"'])
   writeTEC(fdat,['"T (K)"'])
   for m in range(gas.nSpecies()):
       writeTEC(fdat,['"', gas.speciesName(m), '"'])
   for n in range(f.flame.nPoints()):
       f.setGasState(n)
       writeTEC(fdat, [z[n], u[n], V[n], T[n]]+list(gas.moleFractions()))
   fdat.close()

   print 'solution for plotting (TECPLOT) saved to diffusion_flame.dat'
   ```

Solution is saved in file `diffusion_flame.dat`.

5 About convergence

While Cantera calculates a solution, commonly three problems could occur:

1. calculation stops with error:

   ```
   Procedure: OneDim::timeStep
   Error: Time integration failed
   ```

2. calculation runs infinitely,

3. solutions are no-physicals.

Despite method `equilibrate` has generally no problem of convergence, suitable parameters must be chosen to make converge function `solve`. Unfortunately since these problems are highly nonlinear, there is no foolproof way to get a converged solution. Probably the best bet is to start with a simpler mechanism and generate a flame solution that can be used as the initial guess with the bigger mechanism. Cantera automatically takes care of changed species order in the arrays when it starts from a previous solution generated with a different mechanism as long as the corresponding species in each mechanism have the same name (i.e., if methane is called 'CH4' in one, it must be in the other too.) so it is easy to start with small mechanisms and work up. Moreover convergence could be reached holding the temperature fixed, or not refining the grid, or refining it more. One useful thing is to increase the value of `loglevel` to about 5, which will cause a great deal of information to be printed, including which solution component is not converging.
Concerning no-physical solutions with premixed flames, temperature in burnt gas obtained with `solve` must have nearly the same value that adiabatic temperature obtained with `equilibrate` method. If they are not nearly the same, it means that programs stops running before convergence is reached. Calculated values are necessarily fault.

5.1 Some parameters that rule convergence

Setting suitable parameters generally guarantee convergence. To improve convergence:

1. modify the length of domain `lz` or the number of initial points `nz`;

2. modify the grid refinement criteria (`ratio`, `slope`, `curve` and `prune`)

3. set relative and absolute tolerances for steady-state problem and time stepping (by default `rtol`=1.0e-8, `atol`=1.0e-15):

   ```
   setTolerancesSS(doublereal rtol, doublereal atol)
   setTolerancesTS(doublereal rtol, doublereal atol)
   ```

4. set minimum and maximum time step (by default `tmin`=1.0e-16, `tmax`=10.0):

   ```
   setMinTimeStep(doublereal tmin)
   setMaxTimeStep(doublereal tmax)
   ```

5. set time step (by default `stepsize`= 1.0e-5, `tsteps`= [1, 2, 5, 10]):

   ```
   setTimeStep(doublereal stepsize, int n, integer *tsteps)
   ```

6. set maximum number of grid points (by defaul `npoints` =3000):

   ```
   setMaxGridPoints(int dom, int npoints).
   ```

See how to use them reading C++ example of Section 6.2.3.

6 Simplified chemistry

The files `.cti` describe the thermochemical and transport properties of the mixture in a Cantera format.
These input files can be found in directory: /home/cfd2/cantera/CANTERA/INSTALL_DIR/cantera-1.80/data.
It is possible to introduce new chemical schemes:

1. converting chemical schemes in CHEMKIN format thanks to function `ck2cti`;
2. writing a new `.cti` file.

The Chemkin to Cantera input file converter `ck2cti.exe` could be found at `http://tech.groups.yahoo.com/group/cantera/`, the Cantera User's Group site.
In this section, firstly the organization of an input file is described and then some new chemical schemes are introduced and tested (MP1, CM2 and H2O2).

6.1 The organization of input file .cti

To get started, let's take a look at definition of air that could be found in file /home/cfd2/cantera/CANTERA/INSTALL_DIR/cantera-1.80/data/air.cti.
It is composed by four part:

1. The **default unit system** may be set with a `units` directive. The best choice to import an AVBP chemical schemes into Cantera (no unit conversion is required) is:

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

   The *length* and *time* units are used to construct the units for reaction pre-exponential factors. The *energy* units are used for molar thermodynamic properties, in combination with the units for quantity.
   Since activation energies are often specied in units other than those used for thermodynamic properties, a separate eld is devoted to the default units for activation energies.

   ```
   units(length = cm, quantity = mol, act_energy = kcal/mol)
   kf = Arrhenius(A = 1.0e14, b = 0.0, E = 54.0) # E is 54 kcal/mol
   ```

   Note that unit conversions are not done until the entire file has been read. Only one `units` directive should be present in a file, and the defaults it species apply to the entire file. If the file does not contain a `units` directive, the default units are meters, kilograms, kilomoles, and seconds. The allowed values for the fields of the units directive are listed in Table 2.

Field	Allowed values
`length`	'cm', 'm', 'mm'
`quantity`	'mol', 'kmol', 'molec'
`times`	's', 'min', 'hr', 'ms'
`energy`	'J', 'kJ', 'cal', 'kcal'
`act_energy`	'kJ/mol', 'J/mol', 'J/kmol'
	'kcal/mol', 'cal/mol', 'eV', 'K'

Table 2: Allowed values for the fields of the units directive.

2. A **model for a gas** that uses the ideal gas equation of state named `air`.

```
ideal_gas(name = "air",
      elements = " O  N  Ar ",
      species = """ O  O2  N  NO  NO2 N2O  N2  AR """,
      reactions = "all",
      transport = "Mix",
      initial_state = state(temperature = 300.0,
                        pressure = OneAtm,
                        mole_fractions = 'O2:0.21, N2:0.78, AR:0.01') )
```

A chemical_reacting ideal gas mixture of multiple species is described by

```
 ideal_gas(name,elements,species,reactions,
           kinetics,transport,initial_state,options)
```

where:

- `name` : a string to identify the phase. Must be unique among the phase names within the file
- `elements`: a string of element symbols
- `species` : a string or sequence of strings of species symbols
- `reactions` : the homogeneous reactions. If omitted, no reactions will be included. The string `'all'` includes all reactions defined locally in the input file
- `transport`: the transport property model. One of the strings `'none'`, `'multi'` or `'mix'`. Default = `'none'`.
- `initial-state`: initial thermodynamic state, specified with an embedded state entry
- `options`: special processing options. A string or sequence of strings in the format described in Table 3.

Option String	Meaning
`no_validation`	Turn off all validation. Use when the denition has been previously validated to speed up importing the denition into an application. Use with caution!
`skip_undeclared_elements`	When importing species, skip any containing undeclared elements, rather than flagging them as an error.
`skip_undeclared_species`	When importing reactions, skip any containing undeclared species, rather than flagging them as an error.

Table 3: The possible string for options

Apart from the `ideal_gas`, there are several other different types of phases. Currently these are `stoichiometric_solid`, `stoichiometric_liquid` and `ideal_solution`.

3. The **definition of species**:

```
species(name = "O",
    atoms = " O:1 ",
    thermo = (
       NASA( [  200.00,  1000.00], [  3.168267100E+00,  -3.279318840E-03,
                6.643063960E-06,  -6.128066240E-09,   2.112659710E-12,
                2.912225920E+04,   2.051933460E+00] ),
       NASA( [ 1000.00,  3500.00], [  2.569420780E+00,  -8.597411370E-05,
                4.194845890E-08,  -1.001777990E-11,   1.228336910E-15,
                2.921757910E+04,   4.784338640E+00] )
             ),
    transport = gas_transport(
                     geom = "atom",
                     diam =     2.75,
                     well_depth =    80.00),
    note = "L 1/90"
       )
...
```

To obtain the species data for a new chemical scheme just copy the species data from `gri30.cti` file that could be found in the same directory of `air.cti` file. For more details see [4].

4. The **reaction equations** which determine the reactant and product stoichiometry:

```
#  Reaction 1
three_body_reaction( "2 O + M <=> O2 + M",   [1.20000E+17, -1, 0],
         efficiencies = " AR:0.83 ")
#  Reaction 2
reaction( "N + NO <=> N2 + O",   [2.70000E+13, 0, 355])
...
#  Reaction 6
falloff_reaction( "N2O (+ M) <=> N2 + O (+ M)",
         kf = [7.91000E+10, 0, 56020],
         kf0   = [6.37000E+14, 0, 56640],
         efficiencies = " AR:0.625 ")
```

Cantera accepts various reaction types:

(a) A homogeneous chemical reaction with pressure-independent rate coefficient and mass-action kinetics:

`reaction(equation, rate_coeff, id, options)`

where:

- `equation`: A string specifying the chemical equation.
- `rate_coeff`: The rate coefficient for the forward direction. If a sequence of three numbers is given, these will be interpreted as [A, n, E] in the modied Arrhenius function $AT^n \exp(-E/RT)$.
- `id` An optional identication string. If omitted, it defaults to a four-digit numeric string beginning with 0001 for the first reaction in the file.
- `options` Processing options, as described in Table 4.

Option String	**Meaning**
`skip`	It can be used to temporarily remove this reaction from the phase or interface that imports it
`duplicate`	In some cases, it may be appropriate to include duplicate reactions, for example if a reaction can proceed through two distinctly different pathways, each with its own rate expression.
`negative_A`	If there are duplicate reactions such that the total rate is positive, then negative A parameters are acceptable

Table 4: The possible string for options

(b) A three-body reaction:

`three_body_reaction(equation, rate_coeff, efficiencies, id, options)` where:

- `equation`: a string specifying the chemical equation. The reaction can be written in either the association or dissociation directions, and may be reversible or irreversible.
- `rate_coeff`: the rate coefficient for the forward direction. If a sequence of three numbers is given, these will be interpreted as [A, n, E] in the modied Arrhenius function $AT^n \exp(-E/RT)$.
- `efficiencies`: a string specifying the third-body collision efficiencies. The efficiencies for unspecified species are set to 1.0.
- `id` An optional identication string. If omitted, it defaults to a four-digit numeric string beginning with 0001 for the first reaction in the file.
- `options` Processing options, as described in Table 4.

(c) A gas-phase falloff reaction:

```
falloff_reaction(equation, rate_coeff_inf, rate_coeff_0,
                 efficiencies, falloff, id, options)
```

where:

- `equation`: a string specifying the chemical equation.
- `rate_coeff_inf` : the rate coefficient for the forward direction in the high-pressure limit. If a sequence of three numbers is given, these will be interpreted as [A, n, E] in the modied Arrhenius function $AT^n \exp(-E/RT)$.
- `rate_coeff_0`: the rate coefficient for the forward direction in the low-pressure limit. If a sequence of three numbers is given, these will be interpreted as [A, n, E] in the modied Arrhenius function $AT^n \exp(-E/RT)$.
- `efficiencies`: a string specifying the third-body collision efficiencies. The efficiency for unspecified species is set to 1.0.
- `falloff`: an embedded entry specifying a falloff function. If omitted, a unity falloff function (Lindemann form) will be used.
- `id` An optional identication string. If omitted, it defaults to a four-digit numeric string beginning with 0001 for the first reaction in the file.
- `options` Processing options, as described in Table 4.

OBSERVATION: The pre-exponential of the Arrenhius law is expressed in `cgs`, that is the unity used by AVBP. No unity conversion is needed. Moreover, in AVBP a negative pre-exponential indicates that the backward reaction should be taken into account. However, the value used in the code is obviously positive. In Cantera the pre-exponential must be positive and to specify that the reaction should be treated as reversible => must be replaced with <=>.

For more details and for more complicate reactions see [4].

6.2 New chemical schemes

It is possible to solve with Cantera a 1D premixed flame described with the AVBP chemistry (see http://www.cerfacs.fr/ avbp/AVBP_V6.X/AVBPHELP/PARAM/chemistry/index.php). Unfortunally, some limitations exist:

1. it is not possible to give orders for reversible equation. It is possible to avoid this problem splitting reversible equation into two irreversible equation (see Section 7);
2. it is not possible to specify orders for non-reactant;
3. reaction order must be non-negative.

6.2.1 Methane-Air: 1 step mechanism MP1

The source of these examples could be found in directory:

`/home/cfd2/cantera/CANTERA/EXAMPLES/MP1.`

The MP1 mechanism has only one reaction equation:

$$CH_4 + 2O_2 => CO_2 + 2H_2O. \qquad (1)$$

To study it with Cantera:

1. open the `input_premix.dat` file for MP1 mechanism

```
  1                   ! nreac (number of chemical reactions)

*** Reaction #  1 : CH4 + 2 O2 => CO2 + 2 H2O
4                     ! nspec (number of species involved in the reaction)
'CH4'    -1.00     +1.00       0.0
'O2'     -2.00     +0.50       0.0
'CO2'    +1.00     +0.00       0.0
'H2O'    +2.00     +0.00       0.0
 1.1d10                 ! Pre-exponential factor (cgs)
 20000.0d0              ! Activation Energy (cal/mol)
0d0
0
```

2. create a directory to save all new chemical schemes:

```
mkdir MY_REACTIONS
```

3. copy and rename the gri30.cti file:

   ```
   cp /home/cfd2/cantera/CANTERA/INSTALL_DIR/cantera-1.80/data/gri30.cti MP1.cti
   ```

4. modify `ideal_gas` definition of MP1.cti file (just `name, elements` and `species`):

   ```
   ideal_gas(name = "MP1",
         elements = " O H C N ",
         species = """ O2 H2O CH4 CO2 N2 """,
         reactions = "all",
         initial_state = state(temperature = 300.0,
                          pressure = OneAtm)    )

   ideal_gas(name = "MP1_mix",
         elements = " O H C N ",
         species = """ O2 H2O CH4 CO2 N2 """,
         reactions = "all",
         transport = "Mix",
         initial_state = state(temperature = 300.0,
                          pressure = OneAtm)    )


   ideal_gas(name = "MP1_multi",
         elements = " O H C N",
         species = """ O2 H2O CH4 CO2 N2 """,
         reactions = "all",
         transport = "Multi",
         initial_state = state(temperature = 300.0,
                          pressure = OneAtm)    )
   ```

5. change not species data and delete reaction data;

6. write at the end the reaction:

   ```
   #  Reaction 1
   reaction( " CH4 + 2 O2 =>  CO2 + 2 H2O ",   [1.1E+10, 0.0, 20000],
              order="CH4:1.0 O2:0.50")
   ```

7. follow instructions of Section 3.1 to copy and run the `flamespeed.cpp` file (but before running it remember to modify it);

8. add the directory with the new reactions in `main` function:

```
#ifndef CXX_DEMO
int main() {
    addDirectory("/home/cfd2/cantera/CANTERA/MY_REACTIONS")
    return flamespeed(0, 0);
}
#endif
```

9. modify the definition of `IdealGasMix` in `flamespeed` function:

```
int flamespeed(int np, void* p) {
  try {
    int i;
    IdealGasMix gas("MP1.cti","MP1_mix");
    ...
  }
 ...
}
```

10. modify some parameters (length of domain and grid refinement criteria):

```
// create an initial grid
    int nz=5;
    doublereal lz= 0.012;
    ...
    int flowdomain=1;
    double ratio=1.1;
    double slope=0.0005;
    double curve=0.0005;
    double prune=0.005;
```

6.2.2 Methane-Air: 2 steps mechanism CM2

The source of these examples could be found in directory:

`/home/cfd2/cantera/CANTERA/EXAMPLES/CM2.`

The CM2 mechanism has two reaction equations:

$$\begin{gathered} CH_4 + 1.5O_2 => CO + 2H_2O \\ CO + 0.5O_2 <=> CO_2. \end{gathered} \tag{2}$$

To study it with Cantera:

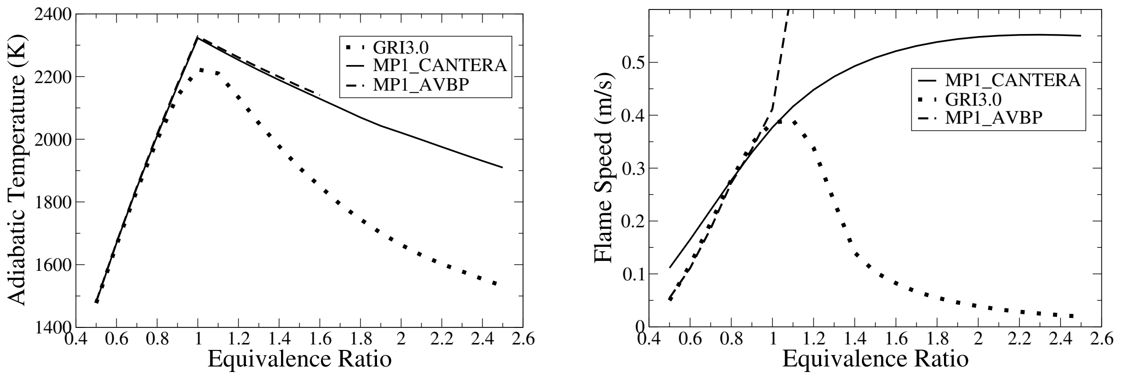


Figure 4: Adiabatic temperature and flame speed as function of ϕ for MP1 mechanism

1. open the `input_premix.dat` file for CM2 mechanism:

```
2                    ! nreac (number of chemical reactions)

*** Reaction #  1 : CH4+ 1.50O2=>CO+2H2O
4                      ! nspec (number of species involved in the reaction)
'CH4'    -1.00     +0.90       0.0
'O2'     -1.50     +1.10       0.0
'CO'     +1.00     +0.00       0.0
'H2O'    +2.00     +0.00       0.0
 2.0D15                    ! Pre-exponential factor (cgs)
 35000.0D0                 ! Activation Energy (cal/mol)

*** Reaction #  2 : CO+ 5.00E-01O2<=>CO2
3                      ! nspec (number of species involved in the reaction)
'CO'    -1.00     +1.00 0.0
'O2'    -0.50     +0.50 0.0
'CO2'   +1.00     +0.00 1.0
-2.0D9                 ! Pre-exponential factor (cgs)
 12000.0D0             ! Activation Energy (cal/mol)
```

2. copy and rename MP1.cti file directory MY_REACTIONS:

```
cp MP1.cti CM2.cti
```

3. modify `ideal_gas` definition of CM2.cti file (just `name` and `species`):

```
ideal_gas(name = "CM2",
      elements = " O H C N ",
      species = """ O2 H2O CH4 CO CO2 N2 """,
      reactions = "all",
      initial_state = state(temperature = 300.0,
                            pressure = OneAtm)    )

ideal_gas(name = "CM2_mix",
      elements = " O H C N ",
      species = """ O2 H2O CH4 CO CO2 N2 """,
      reactions = "all",
      transport = "Mix",
      initial_state = state(temperature = 300.0,
                            pressure = OneAtm)    )


ideal_gas(name = "CM2_multi",
      elements = " O H C N",
      species = """ O2 H2O CH4 CO CO2 N2 """,
      reactions = "all",
      transport = "Multi",
      initial_state = state(temperature = 300.0,
                            pressure = OneAtm)    )
```

4. change not species data and delete reaction data;

5. write at the end the reactions:

```
#  Reaction 1
reaction( " CH4 + 1.5 O2 =>  CO + 2 H2O ",   [0.2000E+16, 0.0, 35000],
          order="CH4:0.9 O2:1.10 ")
#  Reaction 2
reaction( "CO + 0.5 O2 <=> CO2",   [0.2E+10, 0.0, 12000])
```

6. follow instructions of Section 3.1 to copy and run the `flamespeed.cpp` file (but before running it remember to modify it);

7. add the directory with the new reactions in `main` function:

```
#ifndef CXX_DEMO
int main() {
    addDirectory("/home/cfd2/cantera/CANTERA/MY_REACTIONS")
    return flamespeed(0, 0);
```

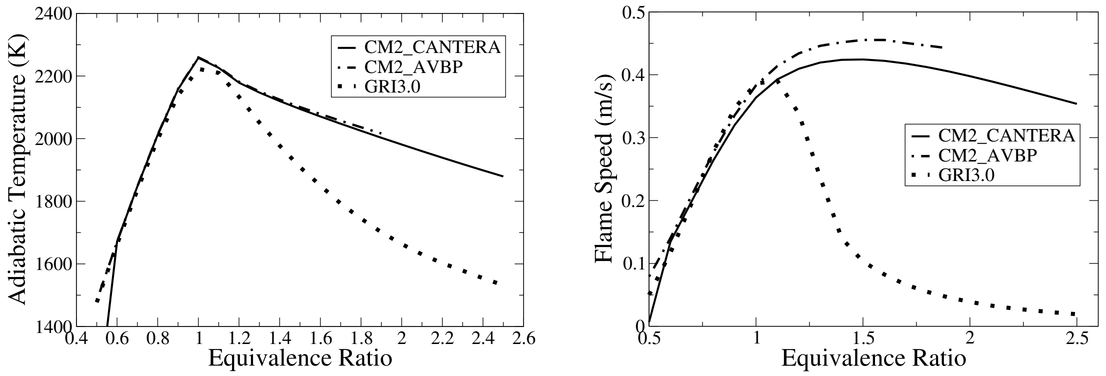


Figure 5: Adiabatic temperature and flame speed as function of ϕ for CM2 mechanism

```
}
#endif
```

8. modify the definition of `IdealGasMix` in `flamespeed` function:

```
int flamespeed(int np, void* p) {
  try {
    int i;
    IdealGasMix gas("CM2.cti","CM2_mix");
    ...
  }
 ...
}
```

9. modify some parameters (length of domain and grid refinement criteria):

```
// create an initial grid
    int nz=5;
    doublereal lz= 0.012;
    ...
    int flowdomain=1;
    double ratio=1.1;
    double slope=0.0005;
    double curve=0.0005;
    double prune=0.005;
```

6.2.3 Hydrogen-Oxygen: 7 steps mechanism H2O2

The source of these examples could be found in directory:

`/home/cfd2/cantera/CANTERA/EXAMPLES/H2O2.`

This mechanism has seven reaction equations:

$$\begin{gathered}
H + O_2 <=> O + OH \\
O + H_2 <=> H + OH \\
O_2 + H_2 <=> OH + OH \\
H_2 + OH <=> H_2O + H \\
OH + OH <=> H_2O + O \\
H + OH + M <=> H_2O + M \\
H + H + M <=> H_2 + M
\end{gathered} \tag{3}$$

To study it with Cantera:

1. open the `input_premix.dat` file for H2O2 mechanism;
2. copy and rename the MP1.cti file directory MY_REACTIONS:

```
cp MP1.cti H2O2.cti
```

3. modify `ideal_gas` definition of H2O2.cti file (just `name`, `elements` and `species`):

```
ideal_gas(name = "H2O2",
       elements = " O  H ",
      species = """ H O O2 OH H2 H2O """,
      reactions = "all",
      initial_state = state(temperature = 300.0,
                        pressure = OneAtm)    )

ideal_gas(name = "H2O2_mix",
       elements = " O  H ",
      species = """ H O O2 OH H2 H2O """,
      reactions = "all",
      transport = "Mix",
      initial_state = state(temperature = 300.0,
                        pressure = OneAtm)    )
```

```
ideal_gas(name = "H2O2_multi",
       elements = " O  H ",
      species = """ H O O2 OH H2 H2O """,
      reactions = "all",
      transport = "Multi",
      initial_state = state(temperature = 300.0,
                        pressure = OneAtm)    )
```

4. change not species data and delete reaction data;

5. write at the end the reactions:

```
#  Reaction 1
reaction( " H + O2 <=>  O + OH ",   [3.62E+17, -0.91, 16530])
#  Reaction 2
reaction( " H2 + O <=> H + OH ",   [1.53E+05, 2.67, 6296])
#  Reaction 3
reaction( " H2 + O2 <=> 2 OH ",   [5.13E+13, 0.0, 48050])
#  Reaction 4
reaction( " OH + H2 <=> H2O + H ",   [6.64E+13, 0.0, 5155])
#  Reaction 5
reaction( " 2 OH <=> H2O + O ",   [1.9E+13, 0.0, 1091])
#  Reaction 6
three_body_reaction( " H + OH + M <=> H2O + M ", [6.67E+22, -2.0, 0.0],
            efficiencies = "H:1.0 H2:2.5 O:1.0 O2:1.0 OH:1.0 H2O:16.0 ")
#  Reaction 7
three_body_reaction( " 2 H + M <=> H2 + M ", [2.2E+18, -1.0, 0.0],
            efficiencies = "H:1.0 H2:2.5 O:1.0 O2:1.0 OH:1.0 H2O:16.0 ")
```

6. follow instructions of Section 3.1 to copy and run the `flamespeed.cpp` file (but before running it remember to modify it);

7. add the directory with the new reactions in `main` function:

```
#ifndef CXX_DEMO
int main() {
    addDirectory("/home/cfd2/cantera/CANTERA/MY_REACTIONS")
    return flamespeed(0, 0);
}
#endif
```

8. modify the definition of `IdealGasMix` in `flamespeed` function:

```
int flamespeed(int np, void* p) {
  try {
    int i;
    IdealGasMix gas("H2O2.cti","H2O2_mix");
    ...
  }
 ...
}
```

9. delete line (actually CH4 and N2 are not defined in H2O2.cti file and this line cause an error):

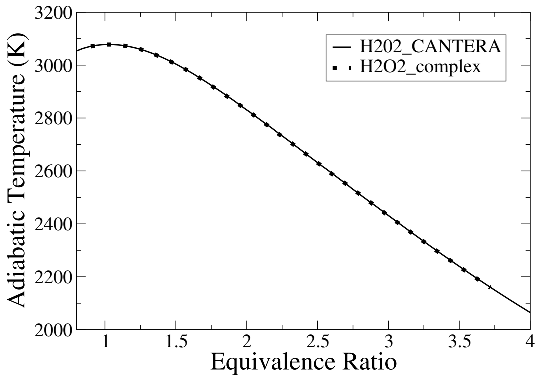
   ```
   gas.setState_TPX(temp, pressure, "CH4:1.0, O2:2.0, N2:7.52");
   ```

10. modify the initial composition of gas:

    ```
       double phi = 0.0;
       if (np > 0) phi = *(double*)(p);
       if (phi == 0.0) {
         ...
       }
    //VALUE MOLAR FRACTIONS
       doublereal nu_F=2;
       doublereal nu_O=1;
       doublereal fa_stoic=nu_O/nu_F;
       doublereal X_F=phi/(phi+fa_stoic);
       doublereal X_O2=fa_stoic/(phi+fa_stoic);
       for(int k=0;k<nsp;k++){
          if(k==gas.speciesIndex("H2")){ x[k]=X_F; }
          else if(k==gas.speciesIndex("O2")){ x[k]=X_O2; }
          else{ x[k]=0; }
        }
    ```

11. modify some parameters (length of domain and grid refinement criteria):

    ```
    // create an initial grid
        int nz=10;
        doublereal lz= 0.5;
        ...
        int flowdomain=1;
        double ratio=1.1;
    ```



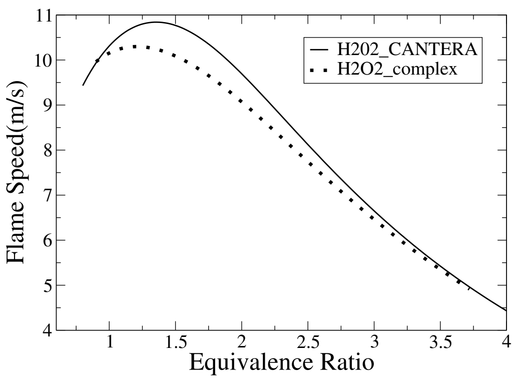


Figure 6: Adiabatic temperature and flame speed as function of ϕ for H2O2 mechanism

```
double slope=0.01;
double curve=0.01;
double prune=-0.005;
```

12. set new parameters to improve convergence as explained in Section 5 :

```
double prune=-0.005;
int npmax=300;
doublereal timestep=1.0e-7;
int num_steps=4;
integer* tsteps=new int[num_steps];
tsteps[0]=1;
tsteps[1]=2;
tsteps[2]=5;
tsteps[3]=10;
doublereal tmin=1.0e-8;
doublereal tmax=1.0e-2;

flow.setTolerancesTS(1.0e-4,1.0e-5);
flow.setTolerancesSS(1.0e-5,1.0e-15);
flame.setMinTimeStep(tmin);
flame.setMaxTimeStep(tmax);
flame.setTimeStep(timestep,num_steps,tsteps);
flame.setMaxGridPoints(flowdomain,npmax);
flame.setRefineCriteria(flowdomain,ratio,slope,curve,prune);
```

6.3 Kerosene-air: 2 steps mechanism 2S_KERO_BFER

An example of a calculation of a kerosene flame with the 2S_KERO_BFER mechanism can be found in the directory: `/home/cfd2/cantera/CANTERA/2S_KERO_BFER/`. Please note that BFER mechanisms (the present kerosene and the 2 steps methane-air) require input_pea.dat files to get correct flame speeds with rich mixtures (see Section 12 for more details on input_pea.dat and PEA formalism).

7 Reverse equilibrium reaction

Sometimes knowing the kinetic parameters of a reversible reaction written in the reverse direction could be necessary. As seen in Section 6.2, it is not possible to define orders for a reversible reaction.
To set orders for a reversible reaction in Cantera:

1. calculate the kinetic parameters in the reverse direction using function `reverse`;
2. modify file .cti splitting reversible reaction into two irreversible reactions;
3. solve flame.

The source of these examples could be found in directory:

`/home/cfd2/cantera/CANTERA/EXAMPLES/REVERSE.`

7.1 Calculate the kinetic parameters in the reverse direction

The first step is realized following what is done in [6]:

1. values of the equilibrium constant K are determined. The variation of equilibrium constant against the temperature can be determined thanks to a 1D-ame:

$$K = \frac{\Pi_j (X_j)^{\nu_j}}{\Pi_i (X_i)^{\nu_i}} \tag{4}$$

   Where X_i and X_j are respectively the molar fractions of the reactant i and product j. ν_i and ν_j are respectively the stoichiometric coefficients of the reactant i and product j.

2. reaction rates are modeled using an Arrhenius law:

$$kf = AT^{\beta} \exp\left(\frac{-E_a}{RT}\right) \tag{5}$$

   where A is the pre-exponential factor, E_a the activation energy, R the universal gas constant and T the temperature.

3. values of $T \ln(\frac{kf}{K})$ as function of T are calculated;

4. from the plotting of values of $T \ln(\frac{kf}{K})$ as function of T, the kinetic parameters for the reverse direction are obtained with XMGRACE starting from equation 6:

$$T \ln\left(\frac{kf}{K}\right) = T \ln(A_2) - \frac{E_{a2}}{R} \tag{6}$$

   where A_2 is the pre-exponential factor and E_{a2} the activation energy for the reverse direction.

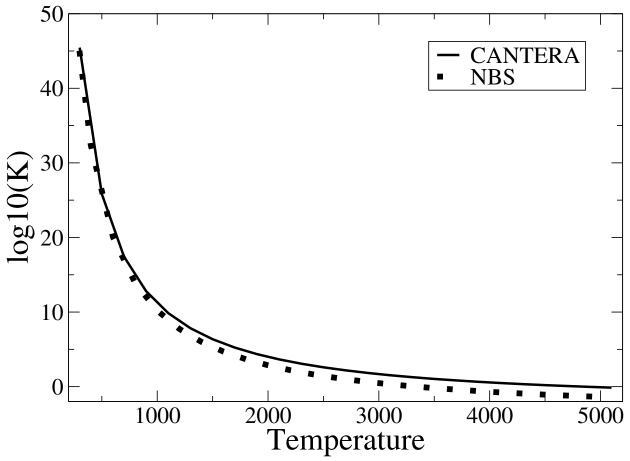


Figure 7: Logarithms to the base 10 of the equilibrium constant K calculated with Cantera and based on information provided by the National Bureau of Standards

To calculate the kinetic parameters with Cantera (CM2 mechanism is chosen as an example):

1. create a work directory:

   ```
   mkdir REVERSE
   ```

2. copy file `CM2_reverse.cpp`:

   ```
   cp /home/cfd2/cantera/CANTERA/CERFACS_ADDED/demos/CM2_reverse.cpp
   ```

3. copy /usr/local/cantera/bin/setup_cantera to your work directory:

   ```
   cp /usr/local/cantera/bin/setup_cantera MY_FIRST_1D_FLAME
   ```

4. source the file `setup_cantera`:

   ```
   source setup_cantera
   ```

5. compile this c++ file. To do that, fill properly `CM2_reverse.mak`:

6. plot with XMGRACE values of $T \ln \left(\frac{kf}{K}\right)$ (memorized into `reverse.dat` with $\log 10(K)$):

   ```
   300 45.4503 -32045.8
   500 25.8264 -26791.1
   ...
   4900 -0.034825 82370.7
   5100 -0.136757 87176.3
   ```

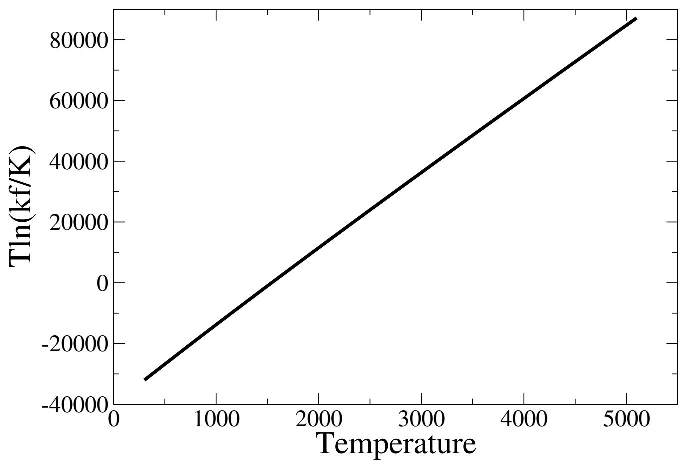


Figure 8: $T \ln\left(\frac{kf}{K}\right)$ against T

7. use option Data → Transformation → Regression, obtaining:

   ```
   y = -38452 + 24.773 * x
   ```

   which means $E_{a2} = 76404$ and $A_2 = 5.7e10$.

7.2 Modify file .cti splitting reversible reaction into two irreversible reactions

To split the reversible reaction:

1. copy chemical schemes `CM2.cti`:

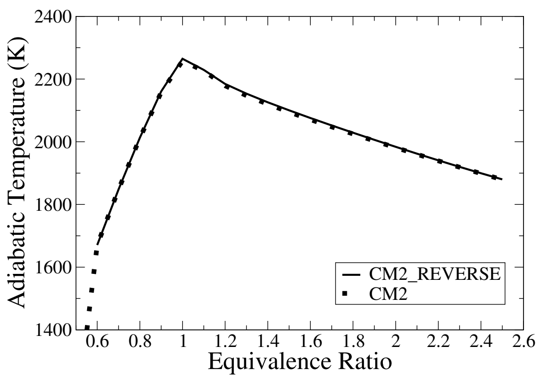
   ```
   cp CM2.cti CM2_reverse.cti
   ```

2. about species, modify only `name`:

   ```
   ideal_gas(name = "CM2_reverse",...)
   ideal_gas(name = "CM2_reverse_mix",..)
   ideal_gas(name = "CM2_reverse_multi",..)
   ```

3. modify reaction data:

   ```
   #  Reaction 1
   reaction( " CH4 + 1.5 O2 =>  CO + 2 H2O ",
             [0.2000E+16, 0.0, 35000], order="CH4:0.9 O2:1.10 ")
   ```



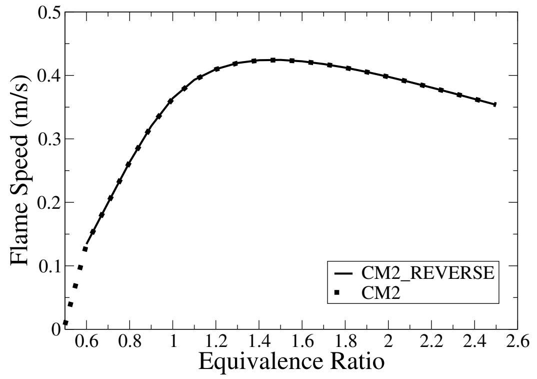


Figure 9: Adiabatic temperature and flame speed as function of ϕ for CM2 mechanism with splitting of reversible reaction

```
#  Reaction 2
reaction( " CO + 0.5 O2 => CO2",   [0.2E+10, 0.0, 12000])
#  Reaction 2bis
reaction( " CO2 => CO + 0.5 O2", [5.7E+10, 0.0, 76404])
```

To check if parameters obtained are correct just check if flame speed of a 1D premixed flame governed by the new mechanism are equal to results of CM2 mechanism. To do this, follow Section 6.2.2 changing in `flamespeed.cpp`:

```
IdealGasMix('CM2_reverse.cti','CM2_reverse_mix').
```

8 Creating an initial solution for AVBP

To create an initial solution for AVBP, similar instructions to the Section 3.3.1 are necessary:

1. copy file `initial_sol_AVBP.h` in the work directory

   ```
   cp /home/cfd2/cantera/CANTERA/CERFACS_ADDED/include/initial_sol_AVBP.h .
   ```

2. open the `flamespeed.cpp` file and insert the command `#include "initial_sol_AVBP.h"`:

   ```
   #include <cantera/Cantera.h>
   #include <cantera/onedim.h>
   ..
   #include <cantera/transport.h>
   #include "initial_sol_AVBP.h"
   ```

3. modify function `flamespeed`. After the calculation of the flame, solution is saved in a file .csv :

   (a) define the name of the file and of the plot:

   ```
   string plotTitle = "1D flame";
   string filename;
   filename = "Initial_solution_AVBP.csv";
   ```

   (b) Values of the spatial variable, of velocity and species mass fractions in a format that can be used by moulinette CAN2AV.

   ```
   Array2D soln(nsp+4,1);
   for(int n=0;n<np;n++){
      flow.setGas(flame.solution()+flame.start(flowdomain),n);
      (n==0) ? saveSolnAVBP(0, flow.grid(n),
                            flame.value(flowdomain,flow.componentIndex("u"),n),
                            gas,soln)
             : saveSolnAVBP(flow.grid(n),
                            flame.value(flowdomain,flow.componentIndex("u"),n),
                            gas,soln);
   }
   plotSolnAVBP(filename, plotTitle, gas, soln);
   cout<<endl<<"Solution for AVBP saved as: "<<filename;
   cout<<endl<<"Remember to convert it with moulinette CAN2AV"<<endl;
   ```

4. create file `can2av.choices`:

```
ofstream choices;
choices.open("can2av.choices");
choices<<"'./Initial_solution_AVBP.csv'     ! CANTERA last or restart file(1)"<<endl;
choices<<"'./Init.sol_0000000.h5'          ! The AVBP solution"<<endl;
choices<<"'./mesh.coor'                    ! The AVBP mesh coordinate"<<endl;
choices<<"'./input_species.dat'            ! The AVBP species file"<<endl;
choices<<"'./input_premix.dat'             ! The AVBP premix file"<<endl;
choices<<gas.pressure()<<"                 ! Pressure of CANTERA flame"<<endl;
choices<<nsp<<"                            ! Number of species "<<endl;
choices<<np<<"                             ! Number of CANTERA grid points"<<endl;
choices.close();

cout<<endl<<"File can2av.choices created."<<endl;
```

NOTE: The output file must be converted to AVBP format using CAN2AV Moulinette. Please make attention: species must be in the same order in `input_species.dat` of AVBP and in file .cti of CANTERA.

9 Continuation method

In this section it is explained how to realize the following operations:

1. create a premixed flame solution for a given pressure p, equivalence ratio ϕ and temperature for the fresh gas T_{fresh};
2. use this solution as initial guess for a flame with a value of pressure, equivalence ratio or temperature slightly different;
3. iterate this procedure for different values of p, ϕ and T_{fresh};
4. saving into a file the values of flame speed and adiabatic temperature at each iteration.

This method reduces the computational time and improves convergence .
First, follow the instructions of section 3.1 to create the initial flame solution. Then modify `flamespeed` function:

1. delete the for-cycle that prints at screen results (or check that all species for which mass fraction is saved are actually defined in file .cti):

```
vector<doublereal> zvec,Tvec,COvec,CO2vec,Uvec;
printf("\n%9s\t%8s\t%5s\t%7s\n","z (m)", "T (K)", "U (m/s)", "Y(CO)");
for(int n=0;n<np;n++){
      Tvec.push_back(flame.value(flowdomain,flow.componentIndex("T"),n));
      COvec.push_back(flame.value(flowdomain,flow.componentIndex("CO"),n));
      CO2vec.push_back(flame.value(flowdomain,flow.componentIndex("CO2"),n));
      Uvec.push_back(flame.value(flowdomain,flow.componentIndex("u"),n));
      zvec.push_back(flow.grid(n));
      printf("%9.6f\t%8.3f\t%5.3f\t%7.5f\n",flow.grid(n),Tvec[n],Uvec[n],COvec[n]);
    }
cout << endl<<"Adiabatic flame temperature from equilibrium is: "<<Tad<<endl;
cout << "Flame speed for phi="<<phi<<" is "<<Uvec[0]<<" m/s."<<endl;
```

2. after having calculate the initial solution, define some new variables (that will be new values of temperature, pressure and ϕ):

```
flame.solve(loglevel,refine_grid);

// CALCULATE NEW SOLUTION WITH DIFFERENT PHI, TEMP or PRESSION
int np=flow.nPoints();
doublereal temp_new=temp;
doublereal press_new=pressure;
double phi_new=phi;
```

3. open file where flame speed,adiabatic temperature and equilibrium mass fractions are saved at each iteration and save initial solution:

   ```
   ofstream myresult;
   myresult.open("my_result.dat");
   myresult<<phi_new<<" ";
   myresult<<flame.value(flowdomain,flow.componentIndex("u"),0)<<" ";
   myresult<<flame.value(flowdomain,flow.componentIndex("T"),np-1)<<" ";
   for(int n=0; n<nsp;n++)
         myresult<<flame.value(flowdomain,n+4,np-1)<<" ";
   myresult<<"\n";
   ```

   Note that in this case equivalence ratio is varied and so results are recorded in function of `phi_new`. If pressure or temperature are varied `press_new` or `temp_new` must be recorded.

4. start the for-cycle that calculate new flame with different values of p, ϕ and T_{fresh} (in this case 10 iterations are performed):

   a. define the new values (in this case only equivalence ratio is changed):

   ```
   for(int k=0;k<10;k++){
      temp_new=temp_new;//*1.01;
      press_new=press_new;//*1.01;
      phi_new=phi_new+0.05;
      X_F=1/(1+4.76*fa_stoic/phi_new);
      X_O2=1/((phi_new/fa_stoic)+4.76);
      X_N2=3.76*X_O2;
      for(int k=0;k<nsp;k++){
         if(k==gas.speciesIndex("CH4")){ x[k]=X_F; }
         else if(k==gas.speciesIndex("O2")){ x[k]=X_O2; }
         else if(k==gas.speciesIndex("N2")){ x[k]=X_N2; }
         else{ x[k]=0.0; }
      }
   ```

   b. impose the new values and calculate new solution:

   ```
      inlet.setTemperature(temp_new);
      inlet.setMoleFractions(DATA_PTR(x));
      flow.setPressure(press_new);
      flame.solve(loglevel,refine_grid);
   ```

c. save solution (as function of equivalence ratio `phi_new`), end for-cycle and close the solution file:

```
   myresult<<phi_new<<" ";
   myresult<<flame.value(flowdomain,flow.componentIndex("u"),0)<<" ";
   myresult<<flame.value(flowdomain,flow.componentIndex("T"),np-1)<<" ";
   for(int n=0; n<nsp;n++)
      myresult<<flame.value(flowdomain,n+4,np-1)<<" ";
   myresult<<"\n";
}
myresult.close();
```

This procedure calculates flames at different equivalence ratio faster than procedure of 3.3.2. Actually, last procedure generates gas and domain at each iteration, on the contrary new method generates them only one time. Even more important, in this method at each new equivalence ratio calculation the previous solution is used as initial solution, on the contrary in section 3.3.2 the initial condition is built at each calculation.

10 Using a previous solution as initial guess

A Cantera solutio could also be saved in specific format (.xml) that could be read by Cantera itself, that means that it is possible to save a Cantera solution and afterwards use it as initial guess for another calculation. This possibility could be used to refine the previous solution, for example, or perform continuation calculations. Remeber always that reading a previous solution from a file could make you loose time at a previous moment, but it could make you gain time and accurancy during the calculation. To save a solution in XML format in `flamespeed.cpp` use (obviously after having calculate your solution with `flame.solve(flowdomain, refine_grid)`):

```
flame.save("previous_solution.xml","1","phi=1.0");
```

where first parameter is file name, second parameter is a string to identify the solution (in fact, different solutions could be saved in the same file and called using the identifyer string) and the third parameter is a comment string. Note that all three parameters have to be given (even if you don't want to comment your flame, you have do do it!!). Note that when saving solution with `flame.save("previous_solution.xml","1","phi=1.0")` if a solution in file `"previous_solution.xml"` with id "1" still exits the new solution will be automatically saved with id "1_1" and so on ("1_2", "1_3",...).

Now it is possible to use the solution as an initil guess for a new calculation. From `flamespeed.cpp` write command `flame.restore("previous_solution.xml","1")` after having creating the initial solution in the usual way:

```
double z1=0.7;
double uout;
uout=inlet.mdot()/rho_out;
uin=inlet.mdot()/rho_in;
locs[0]=0.0; locs[1]=z1; locs[2]=1.0;
value[0]=uin; value[1]=uout; value[2]=uout;
flame.setInitialGuess("u",locs,value);
value[0]=temp; value[1]=Tad; value[2]=Tad;
flame.setInitialGuess("T",locs,value);
for(i=0;i<nsp;i++){
      value[0]=yin[i]; value[1]=yout[i]; value[2]=yout[i];
      flame.setInitialGuess(gas.speciesName(i),locs,value);
}
flame.restore("previous_solution.xml","1");
```

Note that the first parameter is the name of file containing the initial solution and the second parameter is the id of solution to import. The initial guess will be supplied with the solution saved into the file.

11 How to use the AVBP simplified transport properties

Handling with complex transport and thermodynamic properties when constructing a reduced chemical scheme is useless. Thus, simpler properties have been derived under the assumptions of unity Lewis numbers for all species and constant Prandtl number:

$$Le_k = \frac{\lambda}{\rho c_p D_k} = 1 \ , \tag{7}$$

$$Pr = \frac{\mu c_p}{\lambda} = Pr_0 \ , \tag{8}$$

where ρ is the gas mixture density, c_P is the gas mixture specific heat capacity at constant pressure, λ is the gas mixture thermal conductivity, D_k is the diffusion coefficient for species k, and μ is the gas mixture dynamic viscosity following a Second Sutherland law or a power law. In Cantera, it is possible to do this using a transport class named "AVBP". Before run your computation, you must verify in the .cpp file that the transport class has been activated:

```
Transport* tr = newTransportMgr("AVBP", &gas);
```

Moreover,the canteraAVBP version must be used. Verify it in your makefile:

```
 # the directory where the Cantera libraries are located
CANTERA_LIBDIR=/home/cfd2/cantera/CANTERA/INSTALL_DIR/canteraAVBP/lib

# the directory where Cantera include files may be found.
CANTERA_INCDIR=/home/cfd2/cantera/CANTERA/INSTALL_DIR/canteraAVBP/include
```

The transport properties are read by the input_premix.dat and input_thermo.dat files. These files have the same format as the AVBP files, but be carefull: notation form 1.0d-5 is not correctly read by Cantera, 1.0e-5 form must be used ('e' instead of 'd'). Finally, remind you to add them to your directory. An example on how to use the AVBP transport class could be found at the directory:

```
/home/cfd2/cantera/CANTERA/2S_KERO_BFER/
```

12 How to use the PEA

Reduced chemical mechanisms are usually not able to predict flame speed in rich regimes. The so-called PEA (Pre-Exponential Adjustment) is a way to get correct flame speed on the rich side. Basically, the pre-exponential constant is adjusted versus local equivalence ratio to reproduce the proper flame speed dependency with rich mixtures (Fig. 10).

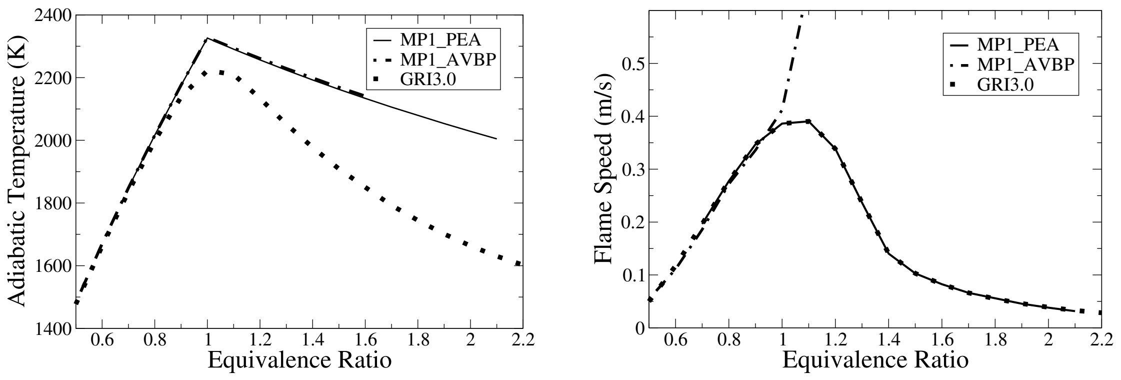


Figure 10: Adiabatic temperature and flame speed as function of ϕ for MP1 mechanism with PEA enabled

PEA is currently available with the trunk version of Cantera and is implemented for 1D domains. Its is thus available for both the premixed and diffusion flame examples presented in the previous sections of the current tutorial. Usage is very similar to AVBP: an input_pea.dat file is required and follows the exact same format as AVBP's input_pea.dat (shown here for the two steps methane BFER mechanism):

```
2                      ! ipea: =0 -> no PEA; =1 -> PEA_1 -> =2 -> PEA_2
'CH4'                  ! name of the fuel species
1 4 0                  ! number of carbon and hydrogen atoms in fuel
'C'                    ! atom retained for mixture fraction calculation
'N2'  0.e0 0.767e0     !
'O2'  0.e0 0.233e0     !
'H2O' 0.e0 0.e0        !
'CO2' 0.e0 0.e0        !
'CO'  0.e0 0.e0        !
'CH4' 1.e0 0.e0        ! species, mass fractions in fuel tank and oxydizer tank

-- PEA on fuel oxidation reaction only ---------------------------------------
-- f=f1+f2
-- f1 = 1/2*{1+tanh[(phi0-phi)/sigma0]}
-- f2 = B*1/4*{1+tanh[(phi-phi1)/sigma1]}*{1+tanh[(phi2-phi)/sigma2]}
PEA_1
3.00e0                  ! phi0
2.50e0                  ! sigma0
0.00e0                  ! B
9.00e-5                 ! phi1
0.00e0                  ! sigma1
8.00e0                  ! phi2
9.00e0                  ! sigma2


-- PEA on fuel oxidation (1) and CO-CO2 equilibrium (2) reactions -------------------------
-- f1 = 2/[{1+tanh[(phi0,1-phi)/sigma0,1]}+B1{1+tanh[(phi-phi1,1)/sigma1,1]}+C1{1+tanh[(phi
-- f2 = 0.5*{1+tanh[(phi0,2-phi)/sigma0,2]}+B2/2*{1+tanh[(phi-phi1,2)/sigma1,2]}+C2/2*{1+ta
PEA_2
1.10e0                  ! phi01
0.09e0                  ! sigma01
0.37e0                  ! B1
1.13e0                  ! phi11
0.03e0                  ! sigma11
6.7e0                   ! C1
1.6e0                   ! phi21
0.22e0                  ! sigma21
0.95e0                  ! phi02
0.08e0                  ! sigma02
2.5e-5                  ! B2
1.3e0                   ! phi12
0.04e0                  ! sigma12
0.0087e0                ! C2
1.2e0                   ! phi22
0.04e0                  ! sigma22
1.2e0                   ! phi32
0.05e0                  ! sigma32
```

The first line contains a switch to choose which formalism to apply:

- 0 : No PEA is applied
- 1 : PEA formalism number 1, mostly used for one-step mechanisms
- 2 : PEA formalism number 2, mostly used for two-steps mechanisms

The second line specifies which species is the fuel. Please note that Cantera will use this line to calculate the mixture fraction required by the PEA formalism.

For more details, please refer to AVBP website :

```
http://www.cerfacs.fr/~avbp/AVBP_V6.X/AVBPHELP/PARAM/pea.php
```

NOTE: When the input_pea.dat file is missing, no correction is applied (ipea set to 0) and the fuel is arbitrarily set to methane (CH4). As a consequence, mixture fraction will be corrupted if the considered fuel is not methane. However it should be noticed that this quantity is needed only in a PEA calculation. If your mechanism does not require PEA, results such as temperature, mass fractions or flame speed will be correct even if the mixture fraction is not correctly calculated. In most cases (including the examples in the present document), mixture fraction is never used.

To retrieve mixture fraction, the getMixFrac() class has been implemented and can be accessed through both Python and C++ programs, as in the examples below:

- Python

```
Z_mixFrac = []
for n in range(f.flame.nPoints()):
    Z_mixFrac.append(0.0)
    Z_mixFrac[n] = f.flame.getMixFrac(n)
    f.setGasState(n)
```

- C++

```
for(int n=0;n<np;n++){
    Tvec.push_back(flame.value(flowdomain,flow.componentIndex("T"),n));
    COvec.push_back(flame.value(flowdomain,flow.componentIndex("CO"),n));
    CO2vec.push_back(flame.value(flowdomain,flow.componentIndex("CO2"),n));
    CH4vec.push_back(flame.value(flowdomain,flow.componentIndex("CH4"),n));
    Uvec.push_back(flame.value(flowdomain,flow.componentIndex("u"),n));
    zvec.push_back(flow.grid(n));
    Z_mix_frac.push_back(flow.getMixFrac(n));
}
```

13 How to compile Cantera

Before compiling, some scripts need to be updated to your configuration. Open the sources folder and do these changes.
If you compile on the CERFACS network, you will have to do these changes:

- file preconfig, line 31

```
CANTERA_CONFIG_PREFIX=${CANTERA_CONFIG_PREFIX:="INSTALL_DIRECTORY"}
```

- file preconfig, line 78

```
USE_NUMERIC=${USE_NUMERIC:="y"}
```

- file tools/src/finish_install.py.in, line 35, 55, 58, 74

```
replace /lib/python by /lib64/python
```

If you compile on a MAC, you will have to do these change:

- file preconfig, line 31

```
CANTERA_CONFIG_PREFIX=${CANTERA_CONFIG_PREFIX:="INSTALL_DIRECTORY"}
```

- file preconfig, line 84

```
USE_NUMPY=${USE_NUMPY:="y"}
```

- file preconfig, line 84

```
NUMPY_HOME=${NUMPY_HOME:="NUMPY_PACKAGE_DIRECTORY"}
```

To get the right numpy home address (NUMPY_PACKAGE_DIRECTORY), look for the file "arrayobject.h" on your MAC. It will be at the address:

```
NUMPY_PACKAGE_DIRECTORY/numpy/arrayobject.h
```

- file configure, line 84

```
NUMPY_HOME=${NUMPY_HOME:="NUMPY_PACKAGE_DIRECTORY"}
```

References

[1] D.G.Goodwin, *Cantera Fortran Users Guide*, http://sourceforge.net/projects/cantera, 2001.

[2] Gregory P. Smith, David M. Golden, Michael Frenklach, Nigel W. Moriarty, Boris Eiteneer, Mikhail Goldenberg, C. Thomas Bowman, Ronald K. Hanson, Soonho Song, William C. Gardiner, Jr., Vitali V. Lissianski, and Zhiwei Qin http://www.me.berkeley.edu/gri_mech/

[3] D.G.Goodwin, *Cantera C++ Users Guide*, http://sourceforge.net/projects/cantera, 2002.

[4] D.G.Goodwin, *Defining Phases and Interfaces*, http://sourceforge.net/projects/cantera, 2003 .

[5] S.Roux, *Influence de la modelisation du melange air/carburant et de l'etendue du domaine de calcul dans la simulation aux grandes echelles des instabilites de combustion. Application des foyers aeronautiques - TH/CFD/07/38.* PhD thesis, InstitutNational Polytechnique de Toulouse, 2007.

[6] G.Lacaze, *Arrenhius kinetic parameters for a reversed equilibrium reaction* WN/CFD/07/39, CERFACS, November 20 2006.