CERFACS

EUROPEAN CENTRE FOR RESEARCH AND ADVANCED TRAINING IN SCIENTIFIC COMPUTING

Cantera

Chemical Kinetics • Thermodynamics • Transport Processes

An object-oriented software toolkit for chemical kinetics, thermodynamics, and transport processes.

Content of the presentation

- CANTERA : Presentation
- Governing equations and numerical methods
- Structure of CANTERA
- Installing CANTERA
- CANTERA with Python
- Tutorials

Content of the presentation

- **CANTERA : Presentation**
  - **What is CANTERA ?**
  - **What can CANTERA do ?**
  - **Why is CANTERA so popular ?**
  - **How will it be helpful to you ?**
- Governing equations and numerical methods
- Structure of CANTERA
- Installing CANTERA
- CANTERA with Python
- Tutorials

What is CANTERA ?

- Cantera is a suite of **object-oriented software tools for problems involving reacting flows.**

- It's possibilities are **comparable to the CHEMKIN-II suite :**
  - You can **perform the same simulations**
  - The **governing equations are similar**
  - **The difference relies in the way the code is triggered:**
    - **CHEMKIN relies on *KEYWORDS***
    - **CANTERA is called through *functions* via *a script*** written in your favorite language (object-oriented)

What is CANTERA ?

- It's possibilities are **comparable to the CHEMKIN-II suite :**

CHEMKIN = a set of FORTRAN libraries	**CANTERA = a set of C++ libraries**
3 input files (thermo / transport / mechanism)	1 « data file » (everything)
« Interpreter step » to generate the binary input file	
A driver to stir the program towards simulations (***Keywords***)	A script to arrange « building blocks » into a simulation (***Interface objects and functions***)
Outputs written by the libraries	Outputs generated by the script's language (Python, C++, Matlab, FORTRAN)

What is CANTERA ?

- It's possibilities are **comparable to the CHEMKIN-II suite :**

CHEMKIN = a set of FORTRAN libraries	**CANTERA = a set of C++ libraries**
3 input files (thermo / transport / mechanism)	**1 « data file » (everything)**
« Interpreter step » to generate the binary input file	
A driver to stir the program towards simulations (***Keywords***)	A script to arrange « building blocks » into a simulation (***Interface objects and functions***)
Outputs written by the libraries	Outputs generated by the script's language (Python, C++, Matlab, FORTRAN)

The beginning of a data file

```
#
# Generated from file MecaUCSDsandiego.mec
# by ck2cti on Wed Jul 29 17:08:40 2009
#
# Transport data from file transUCSD.inp.

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


ideal_gas(name = "gas",
      elements = " N  Ar  He  H  O  C ",
      species = """ N2  AR  HE  H  O2  OH  O  H2  H2O  HO2
                    H2O2  CO  CO2  HCO  CH2O  CH4  CH3  T-CH2  S-CH2  C2H4
                    CH3O  C2H5  C2H6  CH  C2H2  C2H3  CH2CHO  C2H4O  CH2CO  HCCO
                    C2H  CH2OH  CH3OH  C3H4  C3H3  C3H5  C3H6  C3H8  I-C3H7  N-C3H7
                   """,
      reactions = "all",
      transport = "Mix",
      initial_state = state(temperature = 300.0,
                            pressure = OneAtm)    )

#-----------------------------------------------------------------------------
#  Species data
#-----------------------------------------------------------------------------

species(name = "N2",
    atoms = " N:2 ",
    thermo = (
       NASA( [  300.00,  1000.00], [  3.298677000E+00,   1.408240400E-03,
               -3.963222000E-06,   5.641515000E-09,  -2.444854000E-12,
               -1.020899900E+03,   3.950372000E+00] ),
       NASA( [ 1000.00,  5000.00], [  2.926640000E+00,   1.487976800E-03,
               -5.684760000E-07,   1.009703800E-10,  -6.753351000E-15,
               -9.227977000E+02,   5.980528000E+00] )
             ),
    transport = gas_transport(
                      geom = "linear",
                      diam =     3.62,
                      well_depth =    97.53,
                      polar =     1.76,
                      rot_relax =     4.00),
    note = "121286"
       )
```

The beginning of a data file

Phase data

```
#
# Generated from file MecaUCSDsandiego.mec
# by ck2cti on Wed Jul 29 17:08:40 2009
#
# Transport data from file transUCSD.inp.

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


ideal_gas(name = "gas",
      elements = " N  Ar  He  H  O  C ",
      species = """ N2  AR  HE  H  O2  OH  O  H2  H2O  HO2
                    H2O2  CO  CO2  HCO  CH2O  CH4  CH3  T-CH2  S-CH2  C2H4
                    CH3O  C2H5  C2H6  CH  C2H2  C2H3  CH2CHO  C2H4O  CH2CO  HCCO
                    C2H  CH2OH  CH3OH  C3H4  C3H3  C3H5  C3H6  C3H8  I-C3H7  N-C3H7
                   """,
      reactions = "all",
      transport = "Mix",
      initial_state = state(temperature = 300.0,
                            pressure = OneAtm)    )

#-----------------------------------------------------------------------------
#  Species data
#-----------------------------------------------------------------------------

species(name = "N2",
    atoms = " N:2 ",
    thermo = (
       NASA( [  300.00,  1000.00], [  3.298677000E+00,   1.408240400E-03,
               -3.963222000E-06,   5.641515000E-09,  -2.444854000E-12,
               -1.020899900E+03,   3.950372000E+00] ),
       NASA( [ 1000.00,  5000.00], [  2.926640000E+00,   1.487976800E-03,
               -5.684760000E-07,   1.009703800E-10,  -6.753351000E-15,
               -9.227977000E+02,   5.980528000E+00] )
             ),
    transport = gas_transport(
                      geom = "linear",
                      diam =     3.62,
                      well_depth =    97.53,
                      polar =     1.76,
                      rot_relax =     4.00),
    note = "121286"
       )
```

The beginning of a data file

```
#
# Generated from file MecaUCSDsandiego.mec
# by ck2cti on Wed Jul 29 17:08:40 2009
#
# Transport data from file transUCSD.inp.

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


ideal_gas(name = "gas",
      elements = " N  Ar  He  H  O  C ",
      species = """ N2  AR  HE  H  O2  OH  O  H2  H2O  HO2
                    H2O2  CO  CO2  HCO  CH2O  CH4  CH3  T-CH2  S-CH2  C2H4
                    CH3O  C2H5  C2H6  CH  C2H2  C2H3  CH2CHO  C2H4O  CH2CO  HCCO
                    C2H  CH2OH  CH3OH  C3H4  C3H3  C3H5  C3H6  C3H8  I-C3H7  N-C3H7
                   """,
      reactions = "all",
      transport = "Mix",
      initial_state = state(temperature = 300.0,
                            pressure = OneAtm)    )

#-----------------------------------------------------------------------------
#  Species data
#-----------------------------------------------------------------------------

species(name = "N2",
    atoms = " N:2 ",
    thermo = (
       NASA( [  300.00,  1000.00], [  3.298677000E+00,   1.408240400E-03,
               -3.963222000E-06,   5.641515000E-09,  -2.444854000E-12,
               -1.020899900E+03,   3.950372000E+00] ),
       NASA( [ 1000.00,  5000.00], [  2.926640000E+00,   1.487976800E-03,
               -5.684760000E-07,   1.009703800E-10,  -6.753351000E-15,
               -9.227977000E+02,   5.980528000E+00] )
             ),
    transport = gas_transport(
                      geom = "linear",
                      diam =     3.62,
                      well_depth =    97.53,
                      polar =     1.76,
                      rot_relax =     4.00),
    note = "121286"
       )
```

Species « N2 » data

The beginning of a data file

```
#
# Generated from file MecaUCSDsandiego.mec
# by ck2cti on Wed Jul 29 17:08:40 2009
#
# Transport data from file transUCSD.inp.

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


ideal_gas(name = "gas",
      elements = " N  Ar  He  H  O  C ",
      species = """ N2  AR  HE  H  O2  OH  O  H2  H2O  HO2
                    H2O2  CO  CO2  HCO  CH2O  CH4  CH3  T-CH2  S-CH2  C2H4
                    CH3O  C2H5  C2H6  CH  C2H2  C2H3  CH2CHO  C2H4O  CH2CO  HCCO
                    C2H  CH2OH  CH3OH  C3H4  C3H3  C3H5  C3H6  C3H8  I-C3H7  N-C3H7
                   """,
      reactions = "all",
      transport = "Mix",
      initial_state = state(temperature = 300.0,
                            pressure = OneAtm)    )

#-------------------------------------------------------------------------------
#  Species data
#-------------------------------------------------------------------------------

species(name = "N2",
    atoms = " N:2 ",
    thermo = (
       NASA( [  300.00,  1000.00], [  3.298677000E+00,   1.408240400E-03,
              -3.963222000E-06,   5.641515000E-09,  -2.444854000E-12,
              -1.020899900E+03,   3.950372000E+00] ),
       NASA( [ 1000.00,  5000.00], [  2.926640000E+00,   1.487976800E-03,
              -5.684760000E-07,   1.009703800E-10,  -6.753351000E-15,
              -9.227977000E+02,   5.980528000E+00] )
             ),
    transport = gas_transport(
                      geom = "linear",
                      diam =     3.62,
                      well_depth =    97.53,
                      polar =     1.76,
                      rot_relax =     4.00),
    note = "121286"
       )
```

Species « N2 » data

Thermo data

Transport data

The end of a data file

```
#-----------------------------------------------------------------------------
#  Reaction data
#-----------------------------------------------------------------------------

#  Reaction 1
reaction(  "H + O2 <=> OH + O",  [3.52000E+16, -0.7, 17069.8])

#  Reaction 2
reaction(  "H2 + O <=> OH + H",  [5.06000E+04, 2.67, 6290.63])

#  Reaction 3
reaction(  "H2 + OH <=> H2O + H",  [1.17000E+09, 1.3, 3635.28])

#  Reaction 4
reaction(  "H2O + O <=> 2 OH",  [7.60000E+00, 3.84, 12779.6])

#  Reaction 5
three_body_reaction( "H + O + M <=> OH + M",  [4.71000E+18, -1, 0],
         efficiencies = " AR:0.75  CO:1.9  CO2:3.8  H2:2.5  H2O:12  HE:0.75 ")
```

Reaction data

What is CANTERA ?

- It's possibilities are **comparable to the CHEMKIN-II suite :**

CHEMKIN = a set of FORTRAN libraries	**CANTERA = a set of C++ libraries**
3 input files (thermo / transport / mechanism)	1 « data file » (everything)
« Interpreter step » to generate the binary input file	
A driver to stir the program towards simulations (***Keywords***)	A script to arrange « building blocks » into a simulation (***Interface objects and functions***)
Outputs written by the libraries	Outputs generated by the script's language (Python, C++, Matlab, FORTRAN)

What is CANTERA ?

- It's p… uite :

CHE…	…aries
3 in…	
« Interp…	
A dri…	» into a …nd
Ou…	…nguage …N)

CHEMKIN KEYWORD INPUT

```
/ flame configuration, burner stabilized with specified
temperature
BURN
TGIV
/ in the event of a Newton failure, take 100 timesteps
of 1.E-6
TIME 100 1.00E-6
/ begin on a uniform mesh of 6 points
NPTS 6
/ definition of the computational interval
XEND 10.0
XCEN 5.0
WMIX 10.0
/ pressure and inlet mass flow rate
PRES 0.0329 (atmospheres)
FLRT 4.63E-3 (g/cm**2-sec)
```

KEYWORDS

What is CANTERA ?

- It's possibilities are **comparable to the CHEMKIN-II suite :**

CHEMKIN = a set of FORTRAN libraries	**CANTERA = a set of C++ libraries**
3 input files (thermo / transport / mechanism)	1 « data file » (everything)
« Interpreter step » to generate the binary input file	
A driver to stir the program towards simulations (***Keywords***)	A script to arrange « building blocks » into a simulation (***Interface objects and functions***)
Outputs written by the libraries	Outputs generated by the script's language (Python, C++, Matlab, FORTRAN)

What is CANTERA ?

- It’s possibilities are **comparable to the CHEMKIN-II suite :**

CHEMKI	es
3 input fi	
« Interpreter	
A driver to simu	nto a ***ions***)
Outputs	uage

```
"""  Burner Stabilized flame ‘.py’ script. """”

initial_grid    = [0.0, 0.0025, 0.005, 0.0075, 0.0099, 0.01]

f               = ct.BurnerFlame(gas, initial_grid)
f.burner.T      = 400
f.burner.X      = reactants
f.burner.mdot   = 0.04
```

OBJECTS

FUNCTIONS

What is CANTERA ?

- CANTERA, an object-oriented code :

What is CANTERA ?

- **CANTERA has the advantages of an object-oriented code :**
  - *Think of a Lego box* : it allows you to **form complex** kinetic/thermodynamic systems, or **networks from a set of « building blocks »**.
  - Each « building block » (or object) represents a well-defined small component of the global structure.
  - Those « building blocks » are defined and coded in C++, the core language of CANTERA …
  - **… But CANTERA provides a user-friendly interface in Python** (or matlab, or FORTRAN)

An example of application: **the constant pressure batch reactor**

Initial state :

Separating moveable wall, to keep a constant P

A « piston-like » system

Time evolution of the state with CANTERA :

What is CANTERA ?

- **CANTERA's history :**
  - We owe it all to **Dave Goodwin**, the original developer of Cantera whom made it available under BSD licence.
  - He started off with an extensive overhaul of the CHEMKIN suite …
  - … To steer the modeling software towards an **object-oriented structure** with multiple interfaces (C++, Python, Matlab, FORTRAN)
  - Currently, **we are at the version 2.2.0** ***(but we will use version 2.1.1 in this training)***

Version 2.1 came with a new Python interface. Functions and objects' names follow a new convention : PEP8.

What can CANTERA do?

- 0D calculations:
  - **Equilibrium state**
- 0D calculations with time evolution:
  - **Constant Pressure/Volume batch reactor**
  - Steady-state Plug Flow Reactor (PFR)
- 1D calculations:
  - **Burner stabilized and propagating premixed flat flame**
  - Diffusion flame in counterflow configuration
  - Premixed flame in counterflow configuration (strained)

+ Data extraction and post processing:
  - Temperature, Pressure, Mass Fractions …
  - Path Flow analysis, sensitivity analysis

+ Add surface chemistry and heat losses …

Why is CANTERA so popular ?

- ✓ CANTERA has **most of the functionality of the familiar CHEMKIN-II**, plus additional capabilities (non-ideal phases, multiphase equilibrium, electrochemistry…)
- ✓ **CANTERA can (re)use CHEMKIN files** (libraries, mechanisms …)
- ✓ Users can interface with Cantera through Python, C++, the Matlab toolbox and Fortran as well : **no language excuse !**
- ◆ Those interfaces are only front ends; calculations are done in an optimized, compiled code that is really efficient and fast
- ✓ **Basically, it is possible to create anything that pops into your head …**

Why is CANTERA so popular ?

- **On the downside :** CANTERA does not provide any robust documentation. Basically, everything that exists can be accessed through : http://www.cantera.org
- ✓ **But** there is **a big community of users**, they will help solve all bugs and scripting issues : **cantera google group**
- ✓ **CANTERA comes with a series of scripts and examples** in all of the interface languages : currently **in the folder 'samples'**

How will it be helpful to you ?

Validation phase :

- ✓ **Compare different chemistries with CANTERA**, including skeletal and reduced chemistries, in a fast and efficient way :

Example of laminar flame speed comparison between different mechanism for C2H4/Air oxydation

[1] K. Narayanaswamy, G. Blanquart, H. Pitsch "A consistent chemical mechanism for oxidation of substituted aromatic species ". Combustion and Flame, Vol.157 pp. 1879–1898, 2010

How will it be helpful to you ?

Validation phase :

- ✓ **Reduce chemistries**
  - **Implement your own via the main script** (Sensitivity Analysis, …)
  - **Extract relevant data for post processing** with reduction tools (KINALC, YARC, …)
- ✓ **Simulate chemistry-related combustion features** directly
  - Evaluate the impact of simplified transport model (constant Sch, fitted viscosity …)
  - Develop and test various combustion models. At CERFACS, we use a flame front thickening option (DTFLES) in our CFD code AVBP
  - Evaluate the effects of strain on flame behavior: axisymmetric stagnation flows reduce to pseudo 1D simulations (counterflow flames)

Initialization phase:

- ✓ Use 1D profiles to initialize a simulation (CFD codes such as AVBP at CERFACS)
- ✓ Use 1D profiles to generate flamelet tables for use in CFD codes (Mutagen at CORIA)

Content of the presentation

- CANTERA : Presentation
- **Governing equations and numerical methods**
  - **(0D) Equilibrium equations**
  - **(pseudo 1D) Usual reactors equations**
  - **(1D) Laminar premixed flame equations**
  - **Numerical resolution of differential equations**
- Structure of CANTERA
- The joy of installing CANTERA
- CANTERA with Python
- Tutorials

Equilibrium equations[1]:

"The equilibrium state is that corresponding to a minimum of a property called the energy function under specified conditions"

- Several methods exist, based on different energy functions
- Usually, **the Gibbs free energy function is used, at constant T and P**
- 2 different methods exist :
  - **Non-stoichiometric methods** where the conservation of mass is treated separately
  - Stoichiometric methods, more robust but slower

- CANTERA tries a non-stoichiometric method first, and turns to a stoichiometric method (VCS algorithm[2]) if it fails.

[1] C.H.Wong ,"Chemical equilibrium analysis of combustion porudcts at constant volume", 2001
[2] Smith, W.R. and Missen, R.W. "Chemical Reaction Equilibrium Analysis: Theory and Algorithms", 1982

Equilibrium equations:

example with the Gibbs function

"The equilibrium state is that corresponding to a minimum of a property called the energy function under specified conditions"

Use the Gibbs energy function G:

$$G = G(T, P, N_k)$$

So that, when **P and T are constant**:

$$\boxed{dG = \sum_{k=1}^{K} \mu_k dN_k}$$

with

$$\mu k = \frac{\partial U}{\partial Nk}$$

Equilibrium equations:

example with the Gibbs function

"The equilibrium state is that corresponding to a minimum of a property called the energy function under specified conditions"

Use the Gibbs energy function G:

$$G = G(T, P, N_k)$$

So that, when **P and T are constant**:

At equilibrium, we want to minimize G

$$\boxed{dG = \sum_{i=1}^{K} \mu_k dN_k = 0}$$

with

$$\mu k = \frac{\partial U}{\partial Nk}$$

Equilibrium equations: The non-stoichiometric method, example with the Gibbs function

Use the Gibbs energy function G:

$$G = G(T, P, N_k)$$

So that, when **P and T are constant**:

At equilibrium, we want to minimize G

$$\boxed{dG = \sum_{i=1}^{K} \mu_k dN_k = 0}$$

with

$$\mu k = \frac{\partial U}{\partial Nk}$$

With the constraint that the number of moles p_l of every element l (N, O, H, …) is conserved:

$$p_l = \sum_{k=1}^{K} n_{kl} N_k$$

Equilibrium equations: The non-stoichiometric method, example with the Gibbs function

$$dG = \sum_{k=1}^{K} \mu_k dN_k = 0$$

This becomes an optimisation problem where :

$$p_l^* = p_l - \sum_{k=1}^{K} n_{kl} N_k = 0$$

Equilibrium equations: The non-stoichiometric method, example with the Gibbs function

$$dG = \sum_{k=1}^{K} \mu_k dN_k = 0$$

$$p_l^* = p_l - \sum_{k=1}^{K} n_{kl} N_k = 0$$

This becomes an optimisation problem where :

Illustration in 2D **:**

- G(x,y), represented by the surface

- the condition $p_l^*(x,y)$=smthg represented by the red line

Equilibrium equations: The non-stoichiometric method, example with the Gibbs function

$$dG = \sum_{k=1}^{K} \mu_k dN_k = 0$$

$$p_l^* = p_l - \sum_{k=1}^{K} n_{kl} N_k = 0$$

This becomes an optimisation problem where :

Illustration in 2D **:**

- Find an extremum of the function G(x,y), represented by the blue lines

- that satisfies the condition $p_l^*(x,y)$=smthg represented by the red line

Equilibrium equations: The non-stoichiometric method, example with the Gibbs function

$$dG = \sum_{k=1}^{K} \mu_k dN_k = 0$$

$$p_l^* = p_l - \sum_{k=1}^{K} n_{kl} N_k = 0$$

This becomes an optimisation problem where :

Illustration in 2D **:**

- Find an extremum of the function G(x,y), represented by the blue lines

- that satisfies the condition $p_l^*(x,y)$=smthg represented by the red line

Equilibrium equations: The non-stoichiometric method, example with the Gibbs function

This becomes an optimisation problem where :

$$dG = \sum_{k=1}^{K} \mu_k dN_k = 0$$

$$p_l^* = p_l - \sum_{k=1}^{K} n_{kl} N_k = 0$$

Which is solved by **introducing Lagrange multipliers** λ_l such that :

$$G^* = G + \sum_{l=1}^{L} \lambda_l p_l^*$$

Equilibrium equations: The non-stoichiometric method, example with the Gibbs function

This becomes an optimisation problem where :

$$dG = \sum_{k=1}^{K} \mu_k dN_k = 0$$

$$p_l^* = p_l - \sum_{k=1}^{K} n_{kl} N_k = 0$$

Which is solved by introducing Lagrange multipliers λ_l such that :

$$G^* = G + \sum_{l=1}^{L} \lambda_l p_l^*$$

And the problem can be posed as a solution of a set of K + l nonlinear equations :

$$\frac{\partial G^*}{\partial N_k} = \mu_k - \sum_{l=1}^{L} \lambda_l n_{kl} = 0$$

$$\frac{\partial G^*}{\partial \lambda_l} = p_l^* = 0$$

Equilibrium equations: The non-stoichiometric method

Once the λ_l are determined, since **T & P are constant**, the mole fractions are automatically deduced:

$$\mu_k = \sum_{l=1}^{L} \lambda_l n_{kl} \Rightarrow X_k = \frac{P_o}{P} \exp\left(-\frac{g_k^0(T)}{RT} + \sum_{l=1}^{L} n_{kl} \frac{\lambda_l}{RT}\right)$$

- **General procedure** (***Note:*** *no need to provide reactions information !*):
  - The g_k^0 are tabulated
  - The user provides a guess for enough (L) X_k -with the knowledge that $\sum_{k=1}^{K} X_k = 1$
  - The λ_l can then be deduced from the previous K equations
  - The unkown X_k are calculated with those estimated λ_l and $\sum_{k=1}^{K} X_k$ is evaluated
  - If $\sum_{k=1}^{K} X_k$ is « too far » from 1, a new guess for the X_k is provided and the procedure reiterates with well chosen L X_k

Equilibrium equations: adiabatic flame temperature

- Equilibrium calculation are also possible maintaining other quantities constant
  Ex: **H and P constant, through a loop on T**

$$X_k = \frac{P_o}{P}\exp\left(-\frac{g_k^0(T)}{RT} + \sum_{l=1}^{L} n_{kl}\frac{\lambda_l}{RT}\right)$$

$$and$$

$$\Delta H = 0$$

Using H at initial conditions

$$H = \int_{T_0}^{T} C_p dT + \sum_{k=1}^{K} X_k H_{f,k}^{0,m}, \quad C_p = \sum_{k=1}^{K} X_k C_{p,k}^m$$

Equilibrium equations: adiabatic flame temperature

- Equilibrium calculation are also possible maintaining other quantities constant
  Ex: **H and P constant, through a loop on T**

$$X_k = \frac{P_o}{P}\exp\left(-\frac{g_k^0(T)}{RT} + \sum_{l=1}^{L} n_{kl}\frac{\lambda_l}{RT}\right)$$

$$and$$

$$\Delta H = 0$$

Using H at initial conditions

$$H = \int_{T_0}^{T} C_p dT + \sum_{k=1}^{K} X_k H_{f,k}^{0,m}, \quad C_p = \sum_{k=1}^{K} X_k C_{p,k}^m$$

→ **Gives the gas composition and the adiabatic flame temperature**

Batch and Stirred Reactor equations[3]:

Conceptual illustration of a continuously stirred tank reactor.

[3] Picture from Robert J. Kee, Michael E. Coltrin and Peter Glarborg ,"Chemically Reacting Flow: Theory and Practice", 2003 – chap 16

Governing equations

Batch and Stirred Reactor equations[3,4]:

The **state variables** for Cantera's general reactor model are

- m, the mass of the reactor's contents
- V, the reactor's volume
- U, the total internal energy of the reactors contents
- Yk, the mass fractions for each species

[3] Robert J. Kee, Michael E. Coltrin and Peter Glarborg ,"Chemically Reacting Flow: Theory and Practice", 2003 – chap 16
[4] http://cantera.github.io/docs/sphinx/html/reactors.html

Batch and Stirred Reactor equations[34]:

The **state variables** for Cantera's general reactor model are

- m, the mass of the reactor's contents
- V, the reactor's volume
- U, the total internal energy of the reactors contents
- Yk, the mass fractions for each species

- **Reactor Volume:**

$$\frac{dV}{dt} = \sum_{w} f_w A_w v_w(t)$$

Where

A_w, **is the wall Area**

$v_w(t)$, **is the velocity of the wall**

- Mass Conservation:

$$\frac{dm}{dt} = \sum_{in} \dot{m}_{in} - \sum_{out} \dot{m}_{out} + \dot{m}_{wall}$$

Where $\dot{m}_{in,out}$ are inlets and outlets mass flow rate and $\dot{m}_{wall}$ stands for the production of homogeneous phase species on the reactor walls.

[3] Robert J. Kee, Michael E. Coltrin and Peter Glarborg ,"Chemically Reacting Flow: Theory and Practice", 2003 – chap 16

[4] http://cantera.github.io/docs/sphinx/html/reactors.html

Batch and Stirred Reactor equations[34]:

The **state variables** for Cantera's general reactor model are

- m, the mass of the reactor's contents
- V, the reactor's volume
- U, the total internal energy of the reactors contents
- Yk, the mass fractions for each species

- Reactor Volume:

$$\frac{dV}{dt} = \sum_{w} f_w A_w v_w(t)$$

Where
A_w, is the wall Area
$v_w(t)$, is the velocity of the wall

- **Mass Conservation:**

$$\frac{dm}{dt} = \sum_{in} \dot{m}_{in} - \sum_{out} \dot{m}_{out} + \dot{m}_{wall}$$

Where $\dot{m}_{in,out}$ are inlets and outlets mass flow rate and $\dot{m}_{wall}$ stands for the production of homogeneous phase species on the reactor walls.

[3] Robert J. Kee, Michael E. Coltrin and Peter Glarborg ,"Chemically Reacting Flow: Theory and Practice", 2003 – chap 16
[4] http://cantera.github.io/docs/sphinx/html/reactors.html

Batch and Stirred Reactor equations[3,4]:

The **state variables** for Cantera's general reactor model are

- m, the mass of the reactor's contents
- V, the reactor volume
- U, the total internal energy of the reactors contents
- Yk, the mass fractions for each species

- **Species conservation:**

$$\frac{d(mY_k)}{dt} = \sum_{in} \dot{m}_{in} Y_{k,in} - \sum_{out} \dot{m}_{out} Y_k + \dot{m}_{k,gen}$$

Where

$\dot{m}_{k,gen}$ **is the rate at which species k is generated through homogeneous phase reactions in the reactor and on the walls**

- Energy Conservation:

$$\frac{dU}{dt} = -p\frac{dV}{dt} - \dot{Q} + \sum_{in} \dot{m}_{in} h_{in} - h\sum_{out} \dot{m}_{out}$$

Where $\dot{Q}$ is the total rate of heat transfer through all walls

[3] Robert J. Kee, Michael E. Coltrin and Peter Glarborg ,"Chemically Reacting Flow: Theory and Practice", 2003 – chap 16

[4] http://cantera.github.io/docs/sphinx/html/reactors.html

Batch and Stirred Reactor equations[3,4]:

The **state variables** for Cantera's general reactor model are

- m, the mass of the reactor's contents
- V, the reactor volume
- U, the total internal energy of the reactors contents
- Yk, the mass fractions for each species

- **Species conservation:**

$$\frac{d(mY_k)}{dt} = \sum_{in} \dot{m}_{in} Y_{k,in} - \sum_{out} \dot{m}_{out} Y_k + \dot{m}_{k,gen}$$

Where

$\dot{m}_{k,gen}$ **is the rate at which species k is generated through homogeneous phase reactions in the reactor and on the walls**

- Energy Conservation:

$$\frac{dU}{dt} = -p\frac{dV}{dt} - \dot{Q} + \sum_{in} \dot{m}_{in} h_{in} - h\sum_{out} \dot{m}_{out}$$

Where $\dot{Q}$ is the total rate of heat transfer through all walls

[3] Robert J. Kee, Michael E. Coltrin and Peter Glarborg ,"Chemically Reacting Flow: Theory and Practice", 2003 – chap 16

[4] http://cantera.github.io/docs/sphinx/html/reactors.html

Batch and Stirred Reactor equations:

The resulting system of Ordinary Differential Equations are usually stiff, and they are integrated for a user-specified timestep by the Sundials' **solver CVODE**.

Batch and Stirred Reactor equations:

The resulting system of Ordinary Differential Equations are usually stiff, and they are integrated for a user-specified timestep by the Sundials' **solver CVODE**.

→ **Gives the temporal evolution of the quantities inside a vessel**

Laminar premixed flame:

Computed solution to an atmospheric-pressure, freely propagating, stoichiometric, premixed, hydrogen-air, flat flame.

Picture from Robert J. Kee, Michael E. Coltrin and Peter Glarborg ,"Chemically Reacting Flow: Theory and Practice", 2003 – chap 3

Laminar premixed flame equations:

For a steady laminar premixed flame, **we consider 3+K equations, derived from the general conservation equations for perfect gas** in cylindrical coordinates[3]:

- State

$$\rho = \frac{p\bar{W}}{RT}$$

- Continuity

$$\frac{\partial \rho u}{\partial z} = 0$$

- Species continuity

$$\rho \frac{\partial Yk}{\partial t} + \rho u \frac{\partial Yk}{\partial z} = -\frac{\partial j_{k,z}}{\partial z} + \dot{\omega} Wk$$

- Energy

$$\rho c_p \frac{\partial T}{\partial t} + \rho u c_p \frac{\partial T}{\partial z} = \frac{\partial}{\partial z}\left(\lambda \frac{\partial T}{\partial z}\right) - \sum_{k=1}^{K} c_{pk} j_{k,z} \frac{\partial T}{\partial z} - \sum_{k=1}^{K} h_k \dot{\omega}_k W_k$$

$\overline{W}$, is the mean molecular weight of the mixture
$\dot{\omega}_k$, is the molar production rate of species k
$j_{k,z}$, is the diffusive mass flux vector of species k
c_p , is the mass averaged specific heat
$c_{p,k}$, is the specific heat of species k
λ , is the thermal conductivity
z , is the spatial coordinate

[3] Robert J. Kee, Michael E. Coltrin and Peter Glarborg ,"Chemically Reacting Flow: Theory and Practice", 2003 – chap 16

Laminar premixed flame equations:

For a steady laminar premixed flame, **we consider 3+K equations, derived from the general conservation equations for perfect gas** in cylindrical coordinates[3]:

- State

$$\rho = \frac{p\bar{W}}{RT}$$

- Continuity

$$\frac{\partial \rho u}{\partial z} = 0$$

- Species continuity

$$\rho \frac{\partial Yk}{\partial t} + \rho u \frac{\partial Yk}{\partial z} = -\frac{\partial j_{k,z}}{\partial z} + \dot{\omega} Wk$$

- Energy

$$\rho c_p \frac{\partial T}{\partial t} + \rho u c_p \frac{\partial T}{\partial z} = \frac{\partial}{\partial z}\left(\lambda \frac{\partial T}{\partial z}\right) - \sum_{k=1}^{K} c_{pk} j_{k,z} \frac{\partial T}{\partial z} - \sum_{k=1}^{K} h_k \dot{\omega}_k W_k$$

$\bar{W}$ is the [illegible] mixture

W_k [illegible] k

h_k [illegible] cies k

λ , is the thermal conductivity

z , is the spatial coordinate

Steady state version

[3] Robert J. Kee, Michael E. Coltrin and Peter Glarborg ,"Chemically Reacting Flow: Theory and Practice", 2003 – chap 16

Numerical resolution of differential equations:

- The ***steady state*** version of the previous equations (1D flames) are discretized on a generally non uniform mesh, if needed :

Example with the windward difference in space :

$$\left[\frac{dT}{dx}\right]_j = \frac{T_j - T_{j-1}}{x_j - x_{j-1}}$$

Numerical resolution of differential equations:

- The ***steady state*** version of the previous equations (1D flames) are discretized on a generally non uniform mesh, if needed :

Example with the windward difference in space :

$$\left[\frac{dT}{dx}\right]_j = \frac{T_j - T_{j-1}}{x_j - x_{j-1}}$$

- Introduce **the discretized solution vector y** (density, velocity, mole fractions and temperature at each space point) and **the equation vector F** (so the 3+K conservation equations).
- Introduce **the discretized residual F(y)** at each space point.

Numerical resolution of differential equations:

- The computational problem consists of **finding the smallest residual F**, (at all space points j …) :

$$F(y) = 0 \qquad y = (solution_vector_pt1, \ldots, solution_vector_ptJ)$$

Numerical resolution of differential equations:

- The computational problem consists of **finding the smallest residual F**, (at all space points j …) :

$$F(y) = 0 \qquad y = (solution_vector_pt1, \ldots, solution_vector_ptJ)$$

- **A Damped modified Newton solver with internal time integration** is used (but then, transient terms are needed)

Numerical resolution of differential equations:

- The computational problem consists of **finding the smallest residual F**, (at all space points j …) :

$$F(y) = 0 \qquad y = (solution_vector_pt1, \ldots, solution_vector_ptJ)$$

- **A Damped modified Newton solver with internal time integration** is used (but then, transient terms are needed)

- Convergence relies on the **initial estimate,** y^0 **usually provided by equilibrium calculation** at the inlet temperature T_1^0 and pressure P_1^0.

Numerical resolution of differential equations:

An initial y^0 is required :

Numerical resolution of differential equations:

Newton solver:

- What we seek at point m :

$$F(y) = 0$$

Numerical resolution of differential equations:

Newton solver:

- What we seek at point m :

$$F(y) = 0$$

is used to iterate :

$$\frac{\partial F}{\partial y} = \frac{0 - F(y^m)}{y^{m+1} - y^m}$$

Or :

$$\boxed{y^{m+1} = y^m - \left[\frac{\partial F}{\partial y}\right]_{y^m}^{-1} F(y^m)}$$

Numerical resolution of differential equations:

Newton solver:

- What we seek at point m :

$$F(y) = 0$$

is used to iterate :

$$\frac{\partial F}{\partial y} = \frac{0 - F(y^m)}{y^{m+1} - y^m}$$

Or :

$$\boxed{y^{m+1} = y^m - \left[\frac{\partial F}{\partial y}\right]_{y^m}^{-1} F(y^m)}$$

- **Convergence is reached when** $\Delta y^m = y^{m+1} - y^m$ **becomes negligibly small.**
- The mesh might be automatically refined in the region of high gradients

Numerical resolution of differential equations:

Damped modified Newton solver:

- **Evaluating the Jacobian matrices** $\left(\frac{\partial F}{\partial y}\right)_{y^m} = J^m$ **is a time consuming process,** so it is not done at each iteration

Numerical resolution of differential equations:

Damped modified Newton solver:

- Evaluating the Jacobian matrices $\left(\frac{\partial F}{\partial y}\right)_{y^m} = J^m$ is a time consuming process, so it is not done at each iteration

- **The solver evaluates the Jacobian each specified number of iteration step and in between :**

$$(J^m)^{-1} = \lambda^m \left(\frac{\partial F}{\partial y}\right)^{-1}_{y^k} = \boxed{\lambda^m}(J^k)^{-1}; \quad 1 < k \leq m \quad 0 < \lambda^m \leq 1$$

Scaling parameter

Numerical resolution of differential equations:

Damped modified Newton solver:

- Evaluating the Jacobian matrices $\left(\frac{\partial F}{\partial y}\right)_{y^m} = J^m$ is a time consuming process, so it is not done at each iteration

- The solver evaluates the Jacobian each specified number of iteration step and in between :

$$(J^m)^{-1} = \lambda^m \left(\frac{\partial F}{\partial y}\right)_{y^k}^{-1} = \lambda^m (J^k)^{-1}; \quad 1 < k \leq m \quad 0 < \lambda^m \leq 1$$

Scaling parameter

- **So that the problem becomes :**

$$\boxed{(J^k)\Delta y^m = \boxed{-\lambda^m} F(y^m)}$$

Numerical resolution of differential equations:

Damped modified Newton solver:

$$F(y) = 0 \qquad (J^k)\Delta y^m = -\lambda^m F(y^m)$$

Illustration of the full Newton and modified Newton algorithm

Numerical resolution of differential equations:

Damped modified Newton solver:

$$\boxed{(J^k)\Delta y^m = -\lambda^m F(y^m)}$$

- **The λ^m must satisfy the condition that the subsequent undamped step be smaller :**

$$\boxed{\begin{aligned} &\left|\Delta y^{m+1}\right| \leq \left|\Delta y^m\right| \\ &\left|(J^m)^{-1} F(y^{m+1})\right| \leq \left|(J^m)^{-1} F(y^m)\right| \end{aligned}}$$

- If not verified, $\begin{cases} \text{another damping parameter is tested} \\ \text{a new Jacobian is computed} \end{cases}$

Numerical resolution of differential equations:

Damped modified Newton solver:

$$F(y) = 0 \qquad (J^k)\Delta y^m = -\lambda^m F(y^m)$$

Undamped step : blows up

$$y^{(m+1)} = y^{(m)} + \lambda^{(m)}\,\Delta y^{(m)}$$

Pick $\lambda^{(m)}$ such that

$$|\Delta y^{(m+1)}| < |\Delta y^{(m)}|$$

where

$$\Delta y^{(m)} = J^{-1}\,F\,(y^{(m)})$$

$$\Delta y^{(m+1)} = J^{-1}\,F\,(y^{(m+1)})$$

Graphical illustration of the damping strategy for Newton's method.

Numerical resolution of differential equations:

Damped modified Newton solver:

$$F(y) = 0 \qquad (J^k)\Delta y^m = -\lambda^m F(y^m)$$

$y^{(m+1)} = y^{(m)} + \lambda^{(m)}\,\Delta y^{(m)}$

Pick $\lambda^{(m)}$ such that

$|\Delta y^{(m+1)}| < |\Delta y^{(m)}|$

where

$\Delta y^{(m)} = J^{-1}\,F\,(y^{(m)})$

$\Delta y^{(m+1)} = J^{-1}\,F\,(y^{(m+1)})$

Graphical illustration of the damping strategy for Newton's method.

Numerical resolution of differential equations:

Damped modified Newton solver with internal time integration :

Whenever both damping parameters and the new Jacobian fail:

- ***Transient equations*** are used : $F(y_{t=n+1}) = \dfrac{y_{t=n+1} - y_{t=n}}{\Delta t}$

Numerical resolution of differential equations:

Damped modified Newton solver with internal time integration :

Whenever both damping parameters and the new Jacobian fail:

- ***Transient equations*** are used : $F(y_{t=n+1}) = \dfrac{y_{t=n+1} - y_{t=n}}{\Delta t}$
- Time derivatives are approximated by **first-order, implicit backwards finite differences schemes**

Numerical resolution of differential equations:

Damped modified Newton solver with internal time integration :

Whenever both damping parameters and the new Jacobian fail:

- ***Transient equations*** are used : $F(y_{t=n+1}) = \dfrac{y_{t=n+1} - y_{t=n}}{\Delta t}$
- Time derivatives are approximated by first-order, implicit backwards finite differences schemes
- Time steps are specified by the user $\Delta t = h$
- **Same Newton method to solve the system of equations for each time step**

$$G(y) = F(y) - \frac{dy}{dt} = 0 \qquad \boxed{\left(J - \frac{I}{h}\right)(y_{n+1} - y_n) = -G(y_n)}$$

Numerical resolution of differential equations:

Damped modified Newton solver with internal time integration :

Whenever both damping parameters and the new Jacobian fail:

- ***Transient equations*** are used : $F(y_{t=n+1}) = \dfrac{y_{t=n+1} - y_{t=n}}{\Delta t}$
- Time derivatives are approximated by first-order, implicit backwards finite differences schemes
- Time steps are specified by the user $\Delta t = h$
- Same Newton method to solve the system of equations for each time step

$$G(y) = F(y) - \frac{dy}{dt} = 0 \qquad \boxed{\left(J - \frac{I}{h}\right)(y_{n+1} - y_n) = -G(y_n)}$$

- The **new y is used as a new starting estimate for the steady state problem**

- CANTERA : Presentation
- Governing equations and numerical methods
- **Structure of CANTERA**
- The joy of installing CANTERA
- CANTERA with Python
- Tutorials

Cantera is a big lasagna, it has layers :

Cantera is a big lasagna, it has layers :

BUT don't worry : The code takes care of everything on its own

All external libraries called by CANTERA during a simulation: **After CANTERA is properly installed, nothing to worry about**

Cantera is a big lasagna, it has layers :

BUT don't worry : The code takes care of everything on its own

Cantera is a big lasagna, it has layers :

BUT don't worry : The code takes care of everything on its own

Cantera is a big lasagna, it has layers :

BUT don't worry : The code takes care of everything on its own

Cantera is a big lasagna, it has layers :

BUT don't worry : The code takes care of everything on its own

Cantera is a big lasagna, it has layers :

BUT, if you are an experienced user, you can act everywhere !

Let's rewind :

-The script « calls » CANTERA with functions

Let's rewind :

With information from the data file !

-The script « calls » CANTERA with functions

Structure of CANTERA

The « Bookkeeping » layer :

As we have just seen, it is the python script entry. **This layer contains all the methods that will :**

- **Initialize objects** defined in the script :
  - Ex : If a phase object is defined, it will calculate and set its thermodynamic state and implement their transport models
  - Ex: Set the inlet conditions of a “FreeFlame” object
- **Link all objects** together:
  - Ex : Link two reactors through a wall
- **Organize the simulation**:
  - Call the required solvers (so, the “solvers” layer)
  - Extract required output data

The « Solver » layer :

Is usually hidden from the user, and **borrows from famous « free » libraries** (LAPACK, BLAS, …) to perform:

- Equilibrium calculations
- Reactor equations integration
- 1D calculations
- …

- CANTERA : Presentation
- Governing equations and numerical methods
- Structure of CANTERA
- **Installing CANTERA**
- CANTERA with Python
- Tutorials

Official versions :

You can install CANTERA on both LINUX, Windows and Mac

To install the official version, instructions can be found here:

https://cantera.org/install/index.html

The current version is Cantera 2.4.0

CERFACS versions :

If you are from CERFACS get the sources with :

```
git clone git@nitrox.cerfacs.fr:cantera/cantera-avbp.git
```

From outside, you can get them from :

https://chemistry.cerfacs.fr/en/cantera-installation/

Installing CANTERA

The advised procedure for a local installation of the CERFACS Cantera version is :

```
cazeres@cesar:~/Programs/Codes/Cantera$ cd cantera-avbp/
cazeres@cesar:~/Programs/Codes/Cantera/cantera-avbp$ python install_cantera.py
```

The script is only valid for NFS machines installation and Mac OSX installation

- CANTERA : Presentation
- Governing equations and numerical methods
- Structure of CANTERA
- The joy of installing CANTERA
- **CANTERA with Python**
  - **Data files**
  - **Scripts**
  - **Useful links**
- Tutorials

CANTERA with Python

- What do you need to run Cantera ? Again :

- What do you need to run Cantera ?
  - **A data file** ('.cti', or '.xml')
  - **A simulation script** ('.py')

CANTERA with Python

- What do you need to run Cantera ?
  - **A data file** ('.cti', or '.xml')
  - **A simulation script** ('.py')
- **Where do you find the data file** ?
  - From CHEMKIN ('mech.inp', 'therm.dat' & 'trans.dat') via the tool ck2cti
  - From a dataset provided with CANTERA (air, GRIMech, …)
  - You can generate it by hand (see tutorials)

CANTERA with Python

- What do you need to run Cantera ?
  - **A data file** ('.cti', or '.xml')
  - **A simulation script** ('.py')
- Where do you find the data file ?
  - From CHEMKIN ('mech.inp', 'therm.dat' & 'trans.dat') via the tool ck2cti
  - From a dataset provided with CANTERA (air, GRIMech, …)
  - You can generate it by hand (see tutorials)
- **And the script** ?
  - You generate it from « building blocks »

Data files

- But **what is a data file** (format '.cti') ?

We have seen that it is a file which contains information about :

- Phases and interfaces (species involved, thermo and transport models …)
- Elements and species data
- Reaction data (expression, rate coefficients, pressure dependence, …)

Remember from the beginning : Ex of a data file

```
#
# Generated from file MecaUCSDsandiego.mec
# by ck2cti on Wed Jul 29 17:08:40 2009
#
# Transport data from file transUCSD.inp.

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


ideal_gas(name = "gas",
      elements = " N  Ar  He  H  O  C ",
      species = """ N2  AR  HE  H  O2  OH  O  H2  H2O  HO2
                    H2O2  CO  CO2  HCO  CH2O  CH4  CH3  T-CH2  S-CH2  C2H4
                    CH3O  C2H5  C2H6  CH  C2H2  C2H3  CH2CHO  C2H4O  CH2CO  HCCO
                    C2H  CH2OH  CH3OH  C3H4  C3H3  C3H5  C3H6  C3H8  I-C3H7  N-C3H7
                   """,
      reactions = "all",
      transport = "Mix",
      initial_state = state(temperature = 300.0,
                            pressure = OneAtm)    )

#-------------------------------------------------------------------------------
#  Species data
#-------------------------------------------------------------------------------

species(name = "N2",
    atoms = " N:2 ",
    thermo = (
       NASA( [  300.00,  1000.00], [  3.298677000E+00,   1.408240400E-03,
               -3.963222000E-06,   5.641515000E-09,  -2.444854000E-12,
               -1.020899900E+03,   3.950372000E+00] ),
       NASA( [ 1000.00,  5000.00], [  2.926640000E+00,   1.487976800E-03,
               -5.684760000E-07,   1.009703800E-10,  -6.753351000E-15,
               -9.227977000E+02,   5.980528000E+00] )
             ),
    transport = gas_transport(
                      geom = "linear",
                      diam =     3.62,
                      well_depth =    97.53,
                      polar =     1.76,
                      rot_relax =     4.00),
    note = "121286"
       )
```

Phase data

```
#
# Generated from file MecaUCSDsandiego.mec
# by ck2cti on Wed Jul 29 17:08:40 2009
#
# Transport data from file transUCSD.inp.

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


ideal_gas(name = "gas",
      elements = " N  Ar  He  H  O  C ",
      species = """ N2  AR  HE  H  O2  OH  O  H2  H2O  HO2
                    H2O2  CO  CO2  HCO  CH2O  CH4  CH3  T-CH2  S-CH2  C2H4
                    CH3O  C2H5  C2H6  CH  C2H2  C2H3  CH2CHO  C2H4O  CH2CO  HCCO
                    C2H  CH2OH  CH3OH  C3H4  C3H3  C3H5  C3H6  C3H8  I-C3H7  N-C3H7
                   """,
      reactions = "all",
      transport = "Mix",
      initial_state = state(temperature = 300.0,
                            pressure = OneAtm)    )

#-----------------------------------------------------------------------------
#  Species data
#-----------------------------------------------------------------------------

species(name = "N2",
    atoms = " N:2 ",
    thermo = (
       NASA( [  300.00,  1000.00], [  3.298677000E+00,   1.408240400E-03,
               -3.963222000E-06,   5.641515000E-09,  -2.444854000E-12,
               -1.020899900E+03,   3.950372000E+00] ),
       NASA( [ 1000.00,  5000.00], [  2.926640000E+00,   1.487976800E-03,
               -5.684760000E-07,   1.009703800E-10,  -6.753351000E-15,
               -9.227977000E+02,   5.980528000E+00] )
             ),
    transport = gas_transport(
                      geom = "linear",
                      diam =     3.62,
                      well_depth =    97.53,
                      polar =     1.76,
                      rot_relax =     4.00),
    note = "121286"
       )
```

```
#
# Generated from file MecaUCSDsandiego.mec
# by ck2cti on Wed Jul 29 17:08:40 2009
#
# Transport data from file transUCSD.inp.

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


ideal_gas(name = "gas",
      elements = " N  Ar  He  H  O  C ",
      species = """ N2  AR  HE  H  O2  OH  O  H2  H2O  HO2
                    H2O2  CO  CO2  HCO  CH2O  CH4  CH3  T-CH2  S-CH2  C2H4
                    CH3O  C2H5  C2H6  CH  C2H2  C2H3  CH2CHO  C2H4O  CH2CO  HCCO
                    C2H  CH2OH  CH3OH  C3H4  C3H3  C3H5  C3H6  C3H8  I-C3H7  N-C3H7
                   """,
      reactions = "all",
      transport = "Mix",
      initial_state = state(temperature = 300.0,
                            pressure = OneAtm)    )

#-----------------------------------------------------------------------------
#  Species data
#-----------------------------------------------------------------------------

species(name = "N2",
    atoms = " N:2 ",
    thermo = (
       NASA( [  300.00,  1000.00], [  3.298677000E+00,   1.408240400E-03,
               -3.963222000E-06,   5.641515000E-09,  -2.444854000E-12,
               -1.020899900E+03,   3.950372000E+00] ),
       NASA( [ 1000.00,  5000.00], [  2.926640000E+00,   1.487976800E-03,
               -5.684760000E-07,   1.009703800E-10,  -6.753351000E-15,
               -9.227977000E+02,   5.980528000E+00] )
             ),
    transport = gas_transport(
                      geom = "linear",
                      diam =     3.62,
                      well_depth =    97.53,
                      polar =     1.76,
                      rot_relax =     4.00),
    note = "121286"
       )
```

Species « N2 » data

Data files

- **So… how is it written** (format ‘.cti’) ?

It is **composed of « entries » and « directives »** recognized via keywords :

- **So… how is it written** (format ‘.cti’) ?

It is composed of « entries » and « directives » recognized via keywords :

- **A directive** will tell the code how the entry parameters are to be interpreted. For example, the ‘units’ directive :

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

Data files

- **So… how is it written** (format ‘.cti’) ?

It is composed of « entries » and « directives » recognized via keywords :

- **An entry** defines an object, for example, a falloff reaction :

```
# Reaction 174
falloff_reaction( "H + C3H6 (+ M) <=> N-C3H7 (+ M)",
        kf = [1.33000E+13, 0, 3260.04],
        kf0   = [6.26000E+38, -6.66, 7000.48],
        falloff = Troe(A = 1, T3 = 1000, T1 = 1310, T2 = 48100),
        efficiencies = " AR:0.7  C2H6:3  CH4:2  CO:1.5  CO2:2  H2:2  H2O:6 ")
```

Data files

- **So… how is it written** (format ‘.cti’) ?

It is composed of « entries » and « directives » recognized via keywords :

- **Entries are composed of a series of <u>fields</u>** that can be assigned values, for example, ‘name’ or ‘elements’ : for a phase :

```
ideal_gas name = "gri30",
      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",
      initial_state = state(temperature = 300.0,
                            pressure = OneAtm)    )
```

Scripts

- **Scripts can generate simulations**. Let's take an example : the constant pressure batch reactor

The piston-cylinder system

- The constant pressure batch reactor :

It is composed of 5 main « building blocks »:

- 2 Reactors
- Methane-air mixture
- Non reactive air mixture
- Moveable wall

Separating moveable wall

The piston-cylinder system

They form a constant pressure reactor !

- The constant pressure batch reactor :

What does it look like, « program-wise » ?

Building blocks

```
        """  Constant pressure reactor '.py' script. """

#Mechanism used for the methane-air mixture
cti = importPhase('gri30.cti')
#Set initial conditions
cti.set(T = 1200, P = 1, X = 'CH4:0.40, O2:1, N2:3.76')
# Create the batch reactor
r   = Reactor(cti)

#Create a reactor for the environment containing air at
#atmospheric conditions
env = Reservoir(Air())

# 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 = Wall(r,env)
w.set(K = 1.0e6) # set expansion parameter
w.set(A = 1.0)    # set wall area

# Now create a reactor network consisting of the single batch
reactor
# Reason: the only way to advance reactors in time is through a
sim = ReactorNet([r])
```

- The constant pressure batch reactor :

What will you observe ?

Helpful links

- Helpful links for Cantera

  - **Github** - https://github.com/Cantera/cantera

  - **Google Groups page for Cantera** – http://groups.google.com/group/cantera-users

  - **Cantera SourceForge Page** - http://sourceforge.net/projects/cantera/files/ (download all Cantera versions, source code or (Windows) binaries and find more documentation)

Helpful links

CERFACS CANTERA website : www.chemistry.cerfacs.fr

CERFACS knowledge and experience in chemistry for CFD in one website !!!

- **Chemical database** : Detailed, reduced and global kinetics mechanisms
- **Cantera** : CERFACS' version with installation walkthrough, scripts and tutorials
- **Private documentation** : ARCANE, YARC, AVBP
- **Events** : such as this training
- **CFD gallery** : nice pictures with great chemistry

Helpful links

CERFACS CANTERA website : www.chemistry.cerfacs.fr

As CANTERA lives only through its user community, CERFACS has launched a specific CANTERA website where you will find :

- **Detailed and reduced mechanisms**
- Further documentation and **tutorials** (including this presentation)
- Graphs and data files for comparisons
- Links to all CANTERA's websites

The aim of this website is to make *your* life easier, so all contributions are welcome !

CANTERA

- CANTERA : Presentation
- Governing equations and numerical methods
- Structure of CANTERA
- The joy of installing CANTERA
- CANTERA with Python
- **Tutorials …**