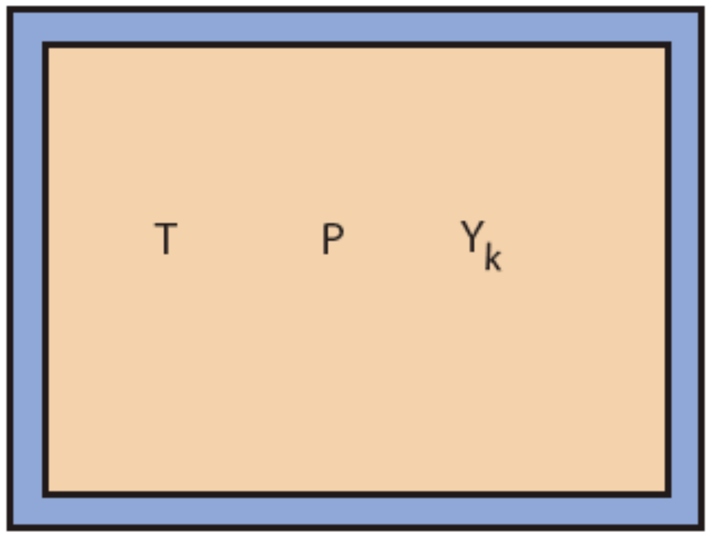
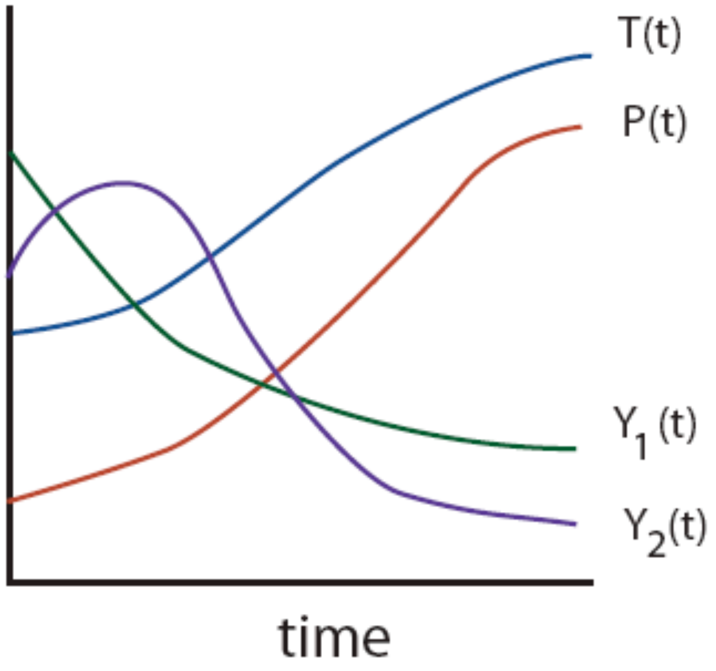
Reactor Networks

D. G. Goodwin
Division of Engineering and Applied Science
California Institute of Technology

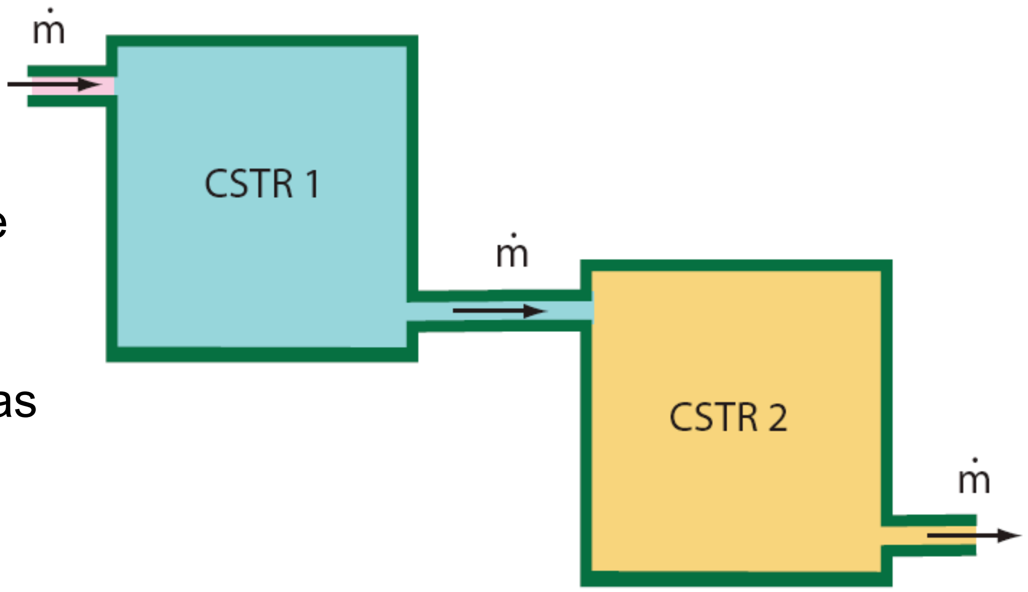
Cantera Workshop
July 25, 2004

A Batch Reactor





Continuously-Stirred Tank Reactors



- In a CSTR, fluid entering through the inlet is instantly mixed into the fluid in the reactor interior
- Fluid leaving through the outlet has the same composition as the reactor interior
- A CSTR is usually treated as a steady-state reactor, with equal inlet and outlet mass flow rates
- Objective is to compute reactor composition (and possibly temperature) as a function of residence time

Constructing Reactor Networks

Creating a reactor

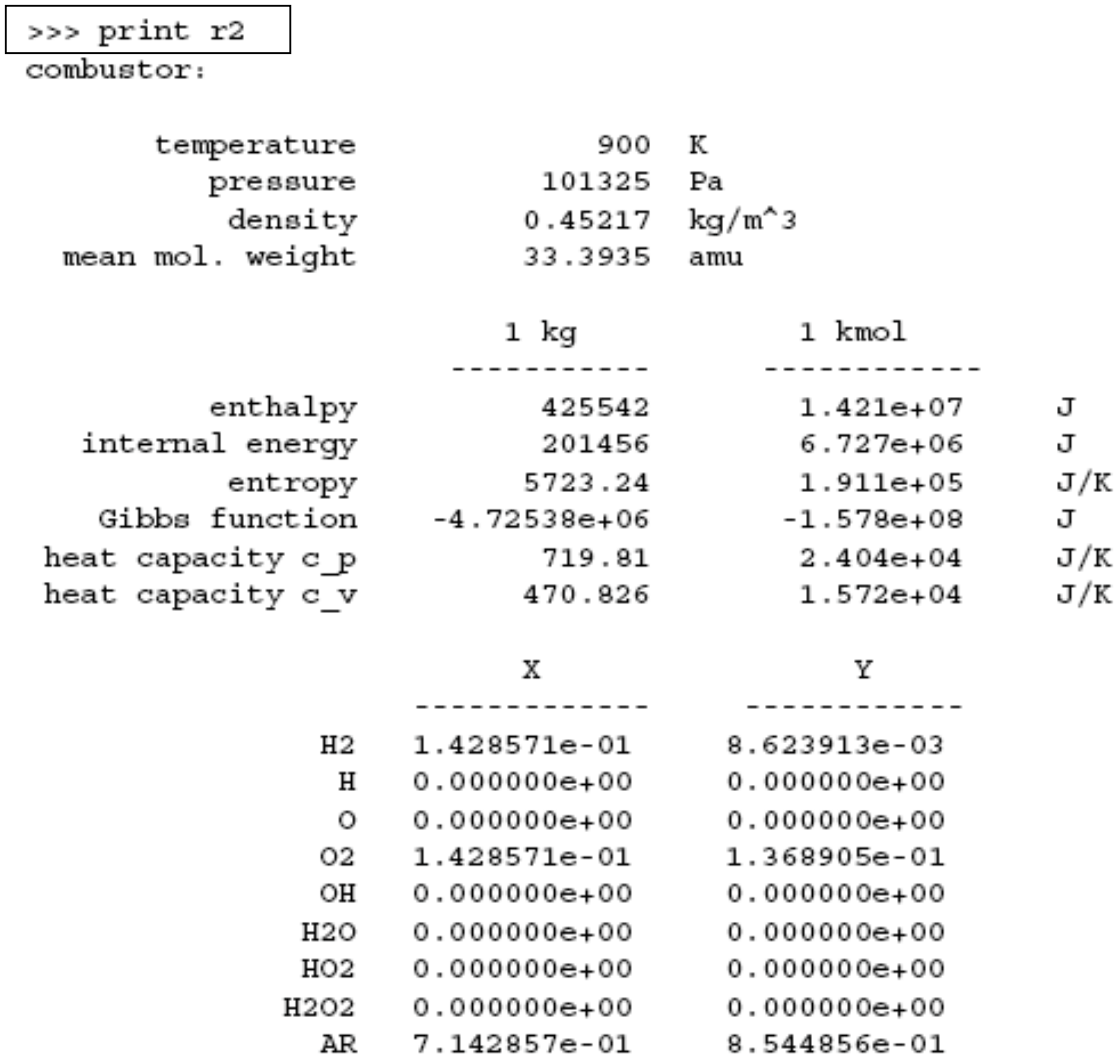
A reactor with default properties can be created like this:

```
>>> r1 = Reactor()
```

Or various attributes may be set at the time of creation by supplying arguments to the **Reactor** constructor:

```
>>> r2 = Reactor(contents = gas,
...                 name = ’combustor’,
...                 volume = 0.5)
```

Viewing the reactor state



```
>>> print r2
combustor:

       temperature             900  K
          pressure          101325  Pa
           density         0.45217  kg/m^3
  mean mol. weight         33.3935  amu

                          1 kg             1 kmol
                       -----------      ------------
          enthalpy          425542       1.421e+07     J
   internal energy          201456       6.727e+06     J
           entropy         5723.24       1.911e+05     J/K
    Gibbs function    -4.72538e+06      -1.578e+08     J
 heat capacity c_p          719.81       2.404e+04     J/K
 heat capacity c_v         470.826       1.572e+04     J/K

                           X                 Y
                     -------------     ------------
                H2   1.428571e-01     8.623913e-03
                 H   0.000000e+00     0.000000e+00
                 O   0.000000e+00     0.000000e+00
                O2   1.428571e-01     1.368905e-01
                OH   0.000000e+00     0.000000e+00
               H2O   0.000000e+00     0.000000e+00
               HO2   0.000000e+00     0.000000e+00
              H2O2   0.000000e+00     0.000000e+00
                AR   7.142857e-01     8.544856e-01
```

Accessing individual properties

```
>>> print r2.temperature()
900.0
>>> print r2.volume()
0.5
>>> print r2.moleFraction('H2')
0.142857142857
>>> print r2.mass()
0.226084973817
>>> print r2.massFractions()
[ 0.00862391  0.          0.          0.13689052  0.          0.
       0.          0.          0.85448557]
```

Creating multiple reactors

- If the same phase model is to be used for all reactors, one phase object can be used for all of them:

```
>>> gas.set(T = 500, P = 1000.0, X = '02:1, N2:3.76')
>>> r1 = Reactor(gas)
>>> gas.set(T = 1200, P = 2.0*OneAtm, X = 'C3H8:1, CO:2, H2:5')
>>> r2 = Reactor(gas)
>>> gas.set(T = 300.0, P = 1.0e4, X = 'AR:2, H2O:3')
>>> r3 = Reactor(gas)
```

- State information is stored locally in each **Reactor** object

- Each one uses the same phase object to compute properties, after first synchronizing the phase with its local state

- Note that the state of the phase object may change after computing any reactor property!

Using multiple phase models

- Using the same phase model for all reactors in a network may be very inefficient, since a large reaction mechanism may be required for some reactors (combustors), while other components may not require reactions at all (air preheaters) or can use a small mechanism or an approximate reduced one
- Each reactor can have its own phase model (reaction mechanism), tailored to its needs
- Large reaction mechanisms can be used selectively, only where needed
- Reduced mechanisms can be used elsewhere, or no reaction mechanism

Initialize each reactor with the appropriate phase object

Here GRI-3.0 is used for the combustor (325 reactions), and non-reactive gas mixtures are used for the two upstream reactors

```
>>> gas = GRI30()
>>> combustor = Reactor(gas)
>>> fuel, air = importPhases('gases.cti', ['methane', 'air'])
>>> fuel_heater = Reactor(fuel)
>>> air_heater = Reactor(air)
```

gases.cti

```
ideal_gas( name = "methane",
           elements = "C H",
           species = "gri30: CH4" )

ideal_gas( name = "air",
           elements = "N O Ar",
           species = "gri30: N2 O2 AR",
           initial_state = state( temperature = 300.0,
                                  pressure = OneAtm,
                                  mole_fractions = "N2:0.78, O2:0.21, AR:0.01"))
```

Reservoirs

- A **Reservoir** object has a state that never changes once set
- Useful to impose fixed upstream conditions, or to specify the environment
- Even if a *reacting* gas mixture is inserted into a reservoir, the state will not change - in effect, all chemistry is 'frozen' in reservoirs

```
>>> air = Air()
>>> env = Reactor(contents = air, name = 'environment')
```

Reactor Network Objects

- All reactors must be integrated in time together, since their evolution may be coupled
- Class ReactorNet provides container objects into which a set of Reactors may be placed
- The ReactorNet object handles all time integration for the Reactors it contains
- A ReactorNet object is created by supplying a sequence (note the square brackets) of Reactor objects to the constructor:

```
>>> net = ReactorNet([r1, r2, r3, r4])
```

The 'advance' method

- To allow the reactor network to evolve in time, use the 'advance' method of class ReactorNet

  >>> net.advance(1.0)

- Argument is the final time (not time interval)
- 'advance' only needs to be called for times when output is desired
- Example:

```
>>> t = 0.1
>>> while t < 2.5:
...       net.advance(t)
...       print t, r1.temperature(), r2.temperature()
...       t += 0.1
```

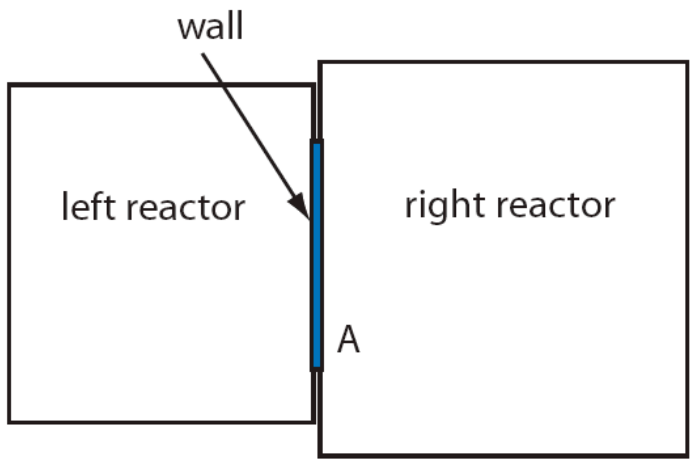
The 'step' method

- Sometimes a fast event (an explosion) may occur between the times specified to 'advance'
- As a result, output file will not capture the fast event, (even though the integrator handled it properly, and the solution after the event is accurate)
- Remedy: use method 'step' instead.
- Method 'step' takes one internal timestep and returns

```
>>> tfinal = 2.5
>>> while tnow < tfinal:
...     tnow = net.step(tfinal)
...     print tnow, r1.temperature(), r2.temperature()
```

Walls

- Walls may be installed between any two reactors and/or reservoirs
- By default, walls are rigid, insulating, and non-reactive
- But wall attributes may be set so that they:
  - move in response to a pressure difference
  - move with a prescribed velocity
  - conduct heat
  - are reactive, with separate reaction mechanisms on each side



Creating and installing a wall

```
w1 = Wall(left = r1, right = r2,
               name = 'wall_1',
               A = 0.5, U = 2.0, K = 100.0)
```

- Wall is installed between r1 on the left, and r2 on the right
- Area = 0.5 m^2 (default: 1.0 m^2)
- Heat transfer coefficient = 2.0 $W/m^2/K$ (default: 0.0)
- Expansion rate parameter = 100 (default: 0.0)

Heat transfer through walls

$$Q(t) = A\left[U(T_\ell - T_r) + \epsilon_{\text{eff}}\sigma(T_\ell^4 - T_r^4) + q_0(t)\right].$$

- A = wall area
- U = overall heat transfer coefficient (conduction / convection
- ε_{eff} = effective emissivity (radiation)
- $q_0(t)$ = specified heat flux function

Some useful limiting cases

conduction through a wall of thickness L, with convection on each side:

$$\boxed{\frac{1}{U} = \frac{1}{h_\ell} + \frac{L}{\kappa} + \frac{1}{h_r}}$$

an evacuated gap with N radiation shields in the gap

$$\boxed{\frac{1}{\epsilon_{\text{eff}}} = \frac{1}{\epsilon_\ell} + \frac{N}{\epsilon_s} + \frac{1}{\epsilon_r}.}$$

Specified heat flux

```
from Cantera.Func import Gaussian
heat_flux = Gaussian(A = 1.0, t0 = 5.0, FWHM = 0.2)
w2 = Wall(left = env, right = r1, heatflux = heat_flux)
```

This wall will deliver a Gaussian heat pulse to reactor r1, with peak value of 1 W/m^2 at t = 5 s, with full-width at half max = 0.2 s.

The heat flux is nominally extracted from the left-hand reactor, but here it is a reservoir with constant state.

Wall motion

$$\dot{V}_{\text{left}} = A\left[K\Delta P + \dot{F}_V(t)\right].$$

- K = constant determining how fast wall moves in response to a unit pressure difference
- Fv = specified time-dependent velocity

Constant-pressure simulations

- To create a constant-pressure reactor
  - Create a reservoir at the desired pressure
  - Set the reactor initial pressure also to this value
  - Install a wall between the reactor and the reservoir, with K sufficiently large to hold P constant to within the desired tolerance (some experimentation with K values is needed)

A piston with sinusoidal velocity

- Wall objects with specified velocity can be used to represent pistons in engine simulations

- This wall represents a piston executing harmonic motion:

```
from Cantera.Func import Fourier
omega = 2.0*Pi*1000.0
coeffs = [(0.0, 0.0), (0.0, 10.0)]    # 10*sin(omega*t)
piston_rate = Fourier(omega, coeffs)
w3 = Wall(left = r1, right = r2, velocity = piston_rate)
```

- Note that it is the piston velocity, not displacement, that is specified.

Surface Chemistry

- A wall object may have reactive surfaces on each side:

```
pt_surf = importInterface('pt.cti','Pt_Surface')
w1 = Wall(left = r1, right = r2, kinetics = [pt_surf, pt_surf])
```

- Note that a *pair* of two interface objects is supplied to the **Wall** constructor. Here they are the same, but in general may differ. The first specifies the reaction mechanism for the side facing the left reactor, and the second the reaction mechanism for the side facing the right reactor.

- If one surface is inert, enter 'None' (without the quotes) in place of the interface object

Flow Controllers

- Reactors and reservoirs may be connected by lines that allow fluid to flow between them.

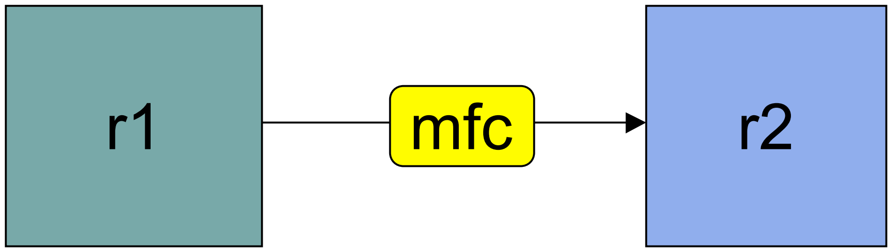
- Each line must contain a device to regulate the flow
  - All devices allow only one-way flow from upstream to downstream

- Three classes of devices are available
  - **MassFlowController** devices
    - Specified mass flow rate, either constant or time-dependent
  - **Valve** devices
    - Mass flow rate is a function of pressure difference
  - **PressureController** devices
    - Designed to be used on reactor outlets to control pressure

Mass Flow Controllers

```
r1 = Reservoir(gas)
r2 = Reactor(gas)
mfc = MassFlowController(upstream = r1,
             downstream = r2,  mdot = 0.1)
```

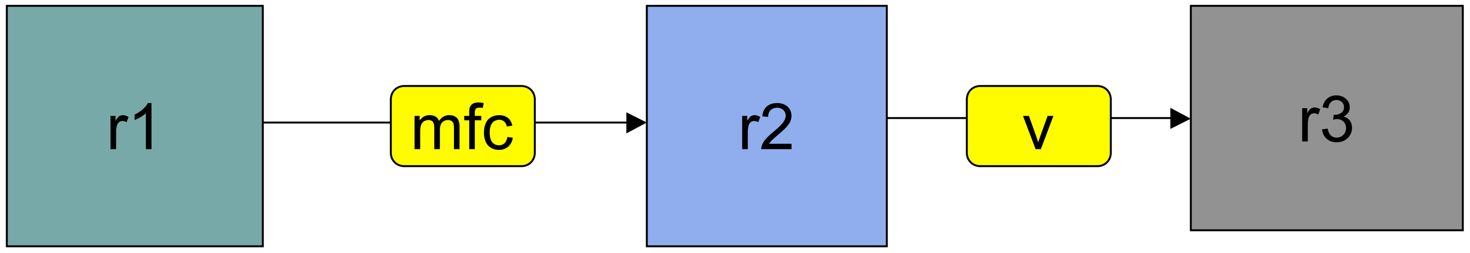
set to either a constant or to a time-dependent functor object



Valves

```
>>> r3 = Reservoir(gas)
>>> v = Valve(upstream = r2, downstream = r3,
                 Kv = 10.0)
```

set to a constant or to a functor object describing the function $K_v(\Delta P)$

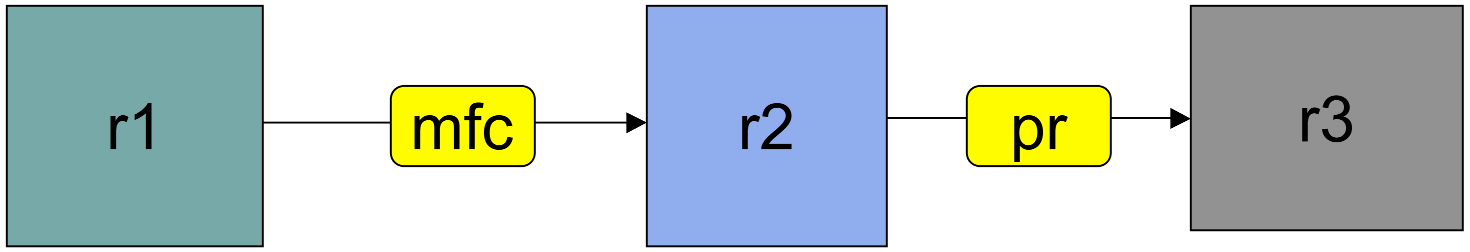


Pressure Controller

```
>>> r3 = Reservoir(gas)
>>> pr = PressureController(upstream = r2,
                 downstream = r3, master = mfc, Kv = 10.0)
```

mass flow rate is that of master, plus term linear in ΔP

use to hold reactor pressure close to downstream reservoir pressure with time-dependent flow into reactor



Mathematical Model

State Variables

For each reactor in the network:

- Total internal energy U [J]
- Volume V [m^3]
- Mass of each species M_k [kg]
- Coverage θ_j of each surface species on each surface facing the reactor

Intensive Variables

- Given the reactor state variables, the intensive variables needed to compute fluid properties may be determined

$$M = \sum_k M_k$$

$$u = U / M$$

$$\rho = M / V$$

$$Y_k = M_k / M$$

$$T = T(u, \rho, Y_k) \longleftarrow \text{invert } u(T, \rho, Y_k)$$

$$P = P(T, \rho, Y_k)$$

Volume and Energy Equations

Volume:

$$\boxed{\frac{dV}{dt} = \sum_{\text{walls}} f_w A_w \underbrace{v_w}_{\text{wall velocity}},}$$

Energy:

$$\boxed{\frac{dU}{dt} = \underbrace{-P\frac{dV}{dt}}_{\text{compression work}} - \underbrace{\sum_{\text{walls}} f_w A_w q_w}_{\text{heat transfer through walls}} + \underbrace{\sum_{\text{inlets}} \dot{m}_i h_{i,\text{in}} + (\sum_{\text{outlets}} \dot{m}_i)h}_{\text{advected energy + flow work}}.}$$

Species and Coverage Equations

Species

$$\frac{dM_k}{dt} = W_k \left[V \dot{\omega}_k + \sum_{\text{walls}} A_w \dot{s}_{k,w} \right] + \sum_{\text{inlets}} \dot{m}_i Y_{k,i,\text{in}} - \left(\sum_{\text{outlets}} \dot{m}_o \right) Y_k .$$

Surface Coverages

$$\frac{d\theta_\ell}{dt} = \frac{\dot{s}_\ell \sigma_\ell}{\Gamma}$$

Examples

An adiabatic, constant-volume batch reactor

```
gas = importPhase('h2o2.cti')
gas.set(T = 900.0, P = OneAtm, X = 'H2:2, O2:1, AR:50')

r1  = Reactor(gas)
sim = ReactorNet([r1])

for n in range(30):
    time = (n+1)*0.002
    sim.advance(time)
    print fmt % (time, r1.temperature(), r1.pressure(),
                 r1.moleFraction('OH'), r1.intEnergy_mass())
```

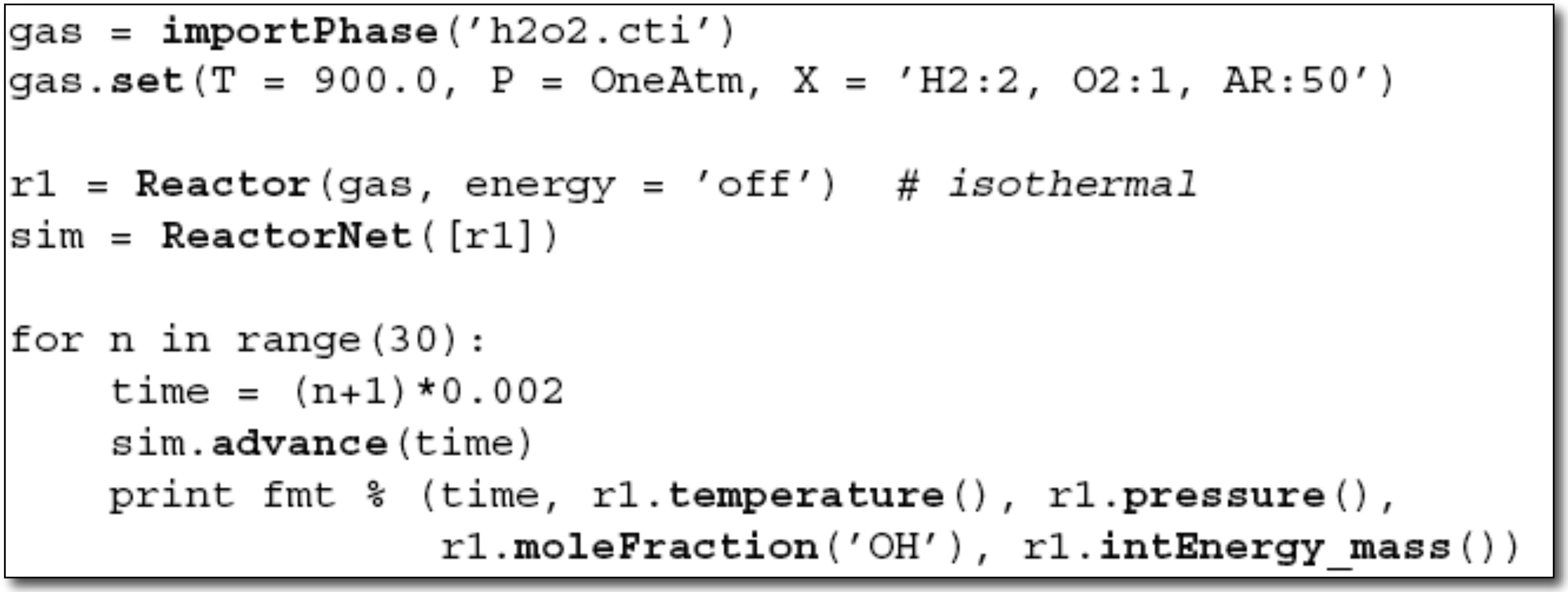
Results

```
 time [s]        T [K]      P [Pa]       X(OH)     u [J/kg]
     0.002       900.0  101325.0008  1.036e-12   139422.81
     0.004       900.0  101325.0037  2.573e-12   139422.81
     0.006       900.0  101325.0101  5.153e-12   139422.81
     0.008       900.0  101325.0226  9.748e-12   139422.81
     0.010       900.0  101325.0461  1.818e-11   139422.81
     0.012       900.0  101325.0900  3.405e-11   139422.81
     0.014       900.0  101325.1730  6.494e-11   139422.81
     0.016       900.0  101325.3345  1.287e-10   139422.81
     0.018       900.0  101325.6678  2.755e-10   139422.81
     0.020       900.0  101326.4462   6.98e-10   139422.81
     0.022       900.0  101328.8998  2.712e-09   139422.81
     0.024       900.2  101351.7403  5.254e-08   139422.81
     0.026      1078.7  120880.3132  0.0002294   139422.81
     0.028      1545.5  170966.9372  0.0003735   139422.81
     0.030      1566.7  173212.6694  0.0002594   139422.81
     0.032      1574.7  174062.0839  0.0002075   139422.81
...
     0.054      1590.2  175696.2023  8.979e-05   139422.81
     0.056      1590.6  175735.4877  8.662e-05   139422.81
     0.058      1590.9  175770.6258  8.376e-05   139422.81
     0.060      1591.2  175802.2837  8.118e-05   139422.81
```

u stays constant despite large changes in T, P, and composition

Isothermal simulations

- If a Reactor object is created with option
  `energy = 'off'`,
  then the energy equation will not be integrated for this reactor, and the temperature will be held constant.



```
gas = importPhase('h2o2.cti')
gas.set(T = 900.0, P = OneAtm, X = 'H2:2, O2:1, AR:50')

r1  = Reactor(gas, energy = 'off')   # isothermal
sim = ReactorNet([r1])

for n in range(30):
    time = (n+1)*0.002
    sim.advance(time)
    print fmt % (time, r1.temperature(), r1.pressure(),
                 r1.moleFraction('OH'), r1.intEnergy_mass())
```

convt.py

An adiabatic, constant-pressure reactor

```
gas = importPhase('h2o2.cti')
gas.set(T = 900.0, P = OneAtm, X = 'H2:2, O2:1, AR:50')

r1  = Reactor(gas)
r2  = Reservoir(gas)
wall = Wall(left = r1, right = r2, K = 1.0e5)
sim = ReactorNet([r1])

for n in range(30):
    time = (n+1)*0.002
    sim.advance(time)
    print fmt % (time, r1.temperature(), r1.pressure(),
                 r1.moleFraction('OH'), r1.enthalpy_mass())
```

file 'conp.py'

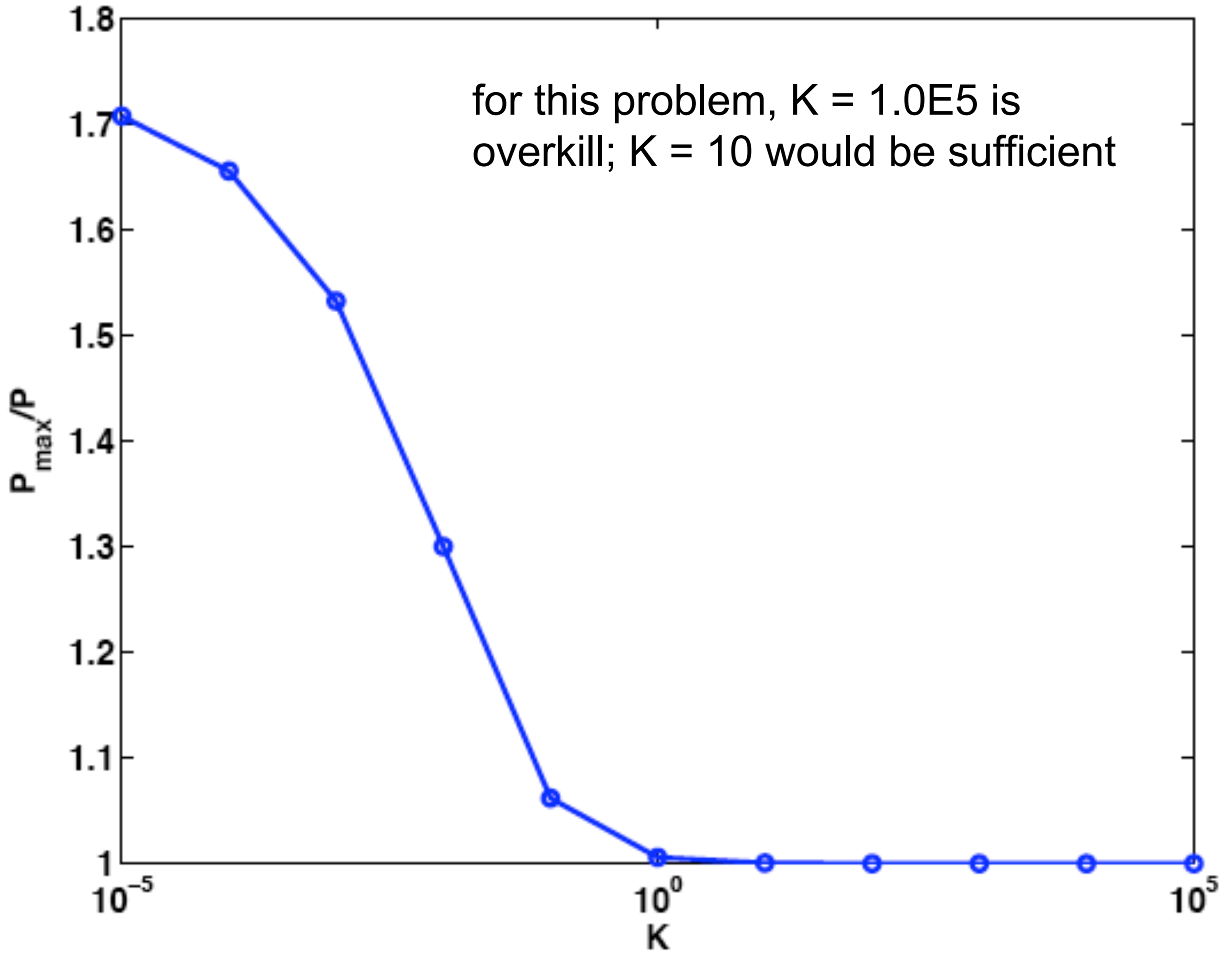
experiment with value
until P deviation is within
desired tolerance

Results

```
 time [s]        T [K]      P [Pa]       X(OH)    h [J/kg]
     0.002       900.0  101325.0000  1.036e-12   334462.82
     0.004       900.0  101325.0000  2.573e-12   334462.82
     0.006       900.0  101325.0000  5.153e-12   334462.82
     0.008       900.0  101325.0000  9.748e-12   334462.82
...
     0.020       900.0  101325.0000  6.973e-10   334462.82
     0.022       900.0  101325.0000  2.705e-09   334462.82
     0.024       900.2  101325.0000  5.174e-08   334462.82
     0.026       955.6  101325.0024  5.559e-05   334462.82
     0.028      1288.9  101325.0002  0.0002682   334462.82
     0.030      1309.9  101325.0001  0.0001901   334462.82
     0.032      1317.6  101325.0000  0.0001527   334462.82
     0.034      1321.7  101325.0000  0.0001302   334462.82
     0.036      1324.3  101325.0000  0.0001149   334462.82
     0.038      1326.1  101325.0000  0.0001036   334462.82
...
     0.056      1332.2  101325.0000   6.17e-05   334462.82
     0.058      1332.5  101325.0000   5.95e-05   334462.82
     0.060      1332.7  101325.0000  5.751e-05   334462.82
```

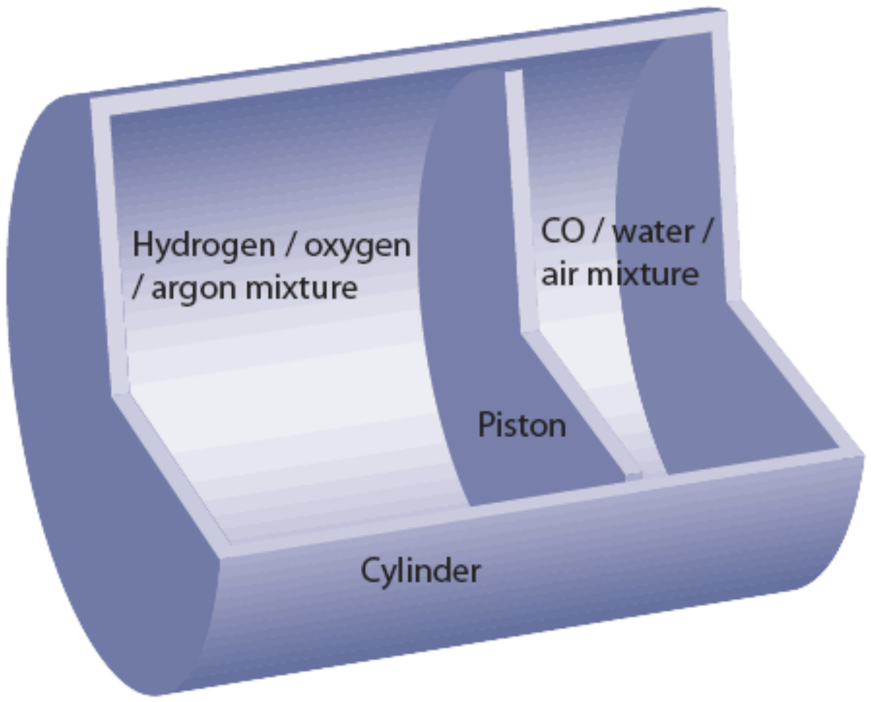
note h stays constant despite large changes in T, P, and composition

Pressure Error vs. K



Coupled Reactors: a Cylinder Divided by a Free Piston

- Each side modeled as a Reactor object
- Free piston is represented by a Wall object installed between the reactors
- Note that different reaction mechanisms are used on each side

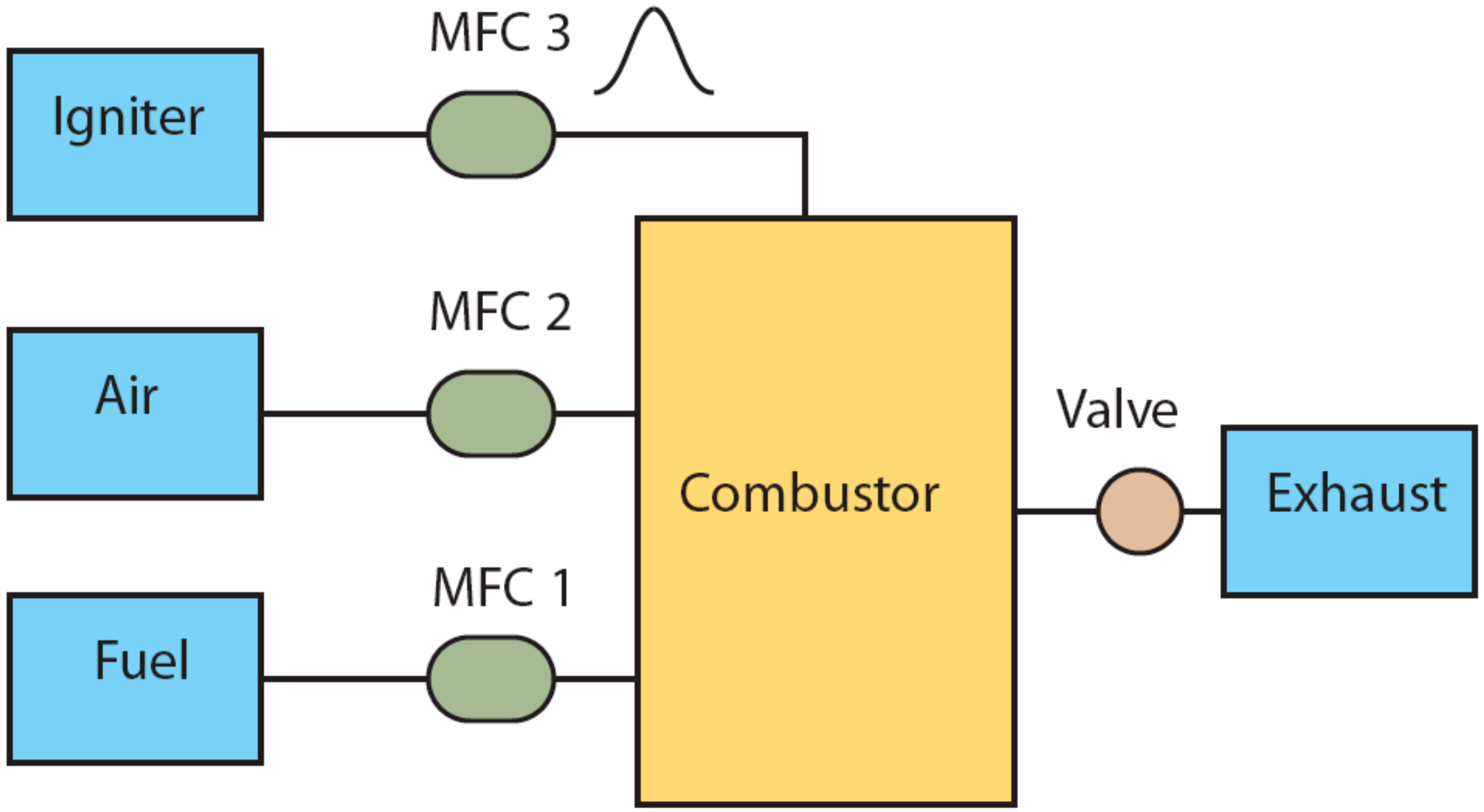


Python Script for Free Cylinder Problem

file 'piston.py'

```
gas1 = importPhase('h2o2.cti')
gas1.set(T = 900.0, P = OneAtm, X = 'H2:2, O2:1, AR:20')

gas2 = GRI30()
gas2.set(T = 900.0, P = OneAtm, X = 'CO:2, H2O:0.01, O2:5')

r1 = Reactor(gas1, volume = 0.5)
r2 = Reactor(gas2, volume = 0.1)
w = Wall(left = r1, right = r2, K = 1.0e3)

reactors = ReactorNet([r1, r2])


for n in range(30):
    time = (n+1)*0.002
    reactors.advance(time)
    print fmt % (time, r1.temperature(), r2.temperature(),
                 r1.volume(), r2.volume(), r1.volume() + r2.volume(),
                 r2.moleFraction('CO'))
```

A Steady-Flow Example: A Combustor



Combustor Python Script: Part I

```
from Cantera.Func import *

# use reaction mechanism GRI-Mech 3.0
gas = GRI30()

# set the fuel inlet state
gas.set(T = 300.0, P = OneAtm, X = 'CH4:1.0')
fuel_in = Reservoir(gas)
fuel_mw = gas.meanMolarMass()

# create a reservoir of H atoms to use to ignite the mixture
gas.set(T = 300.0, P = OneAtm, X = 'H:1.0')
igniter = Reservoir(gas)
```

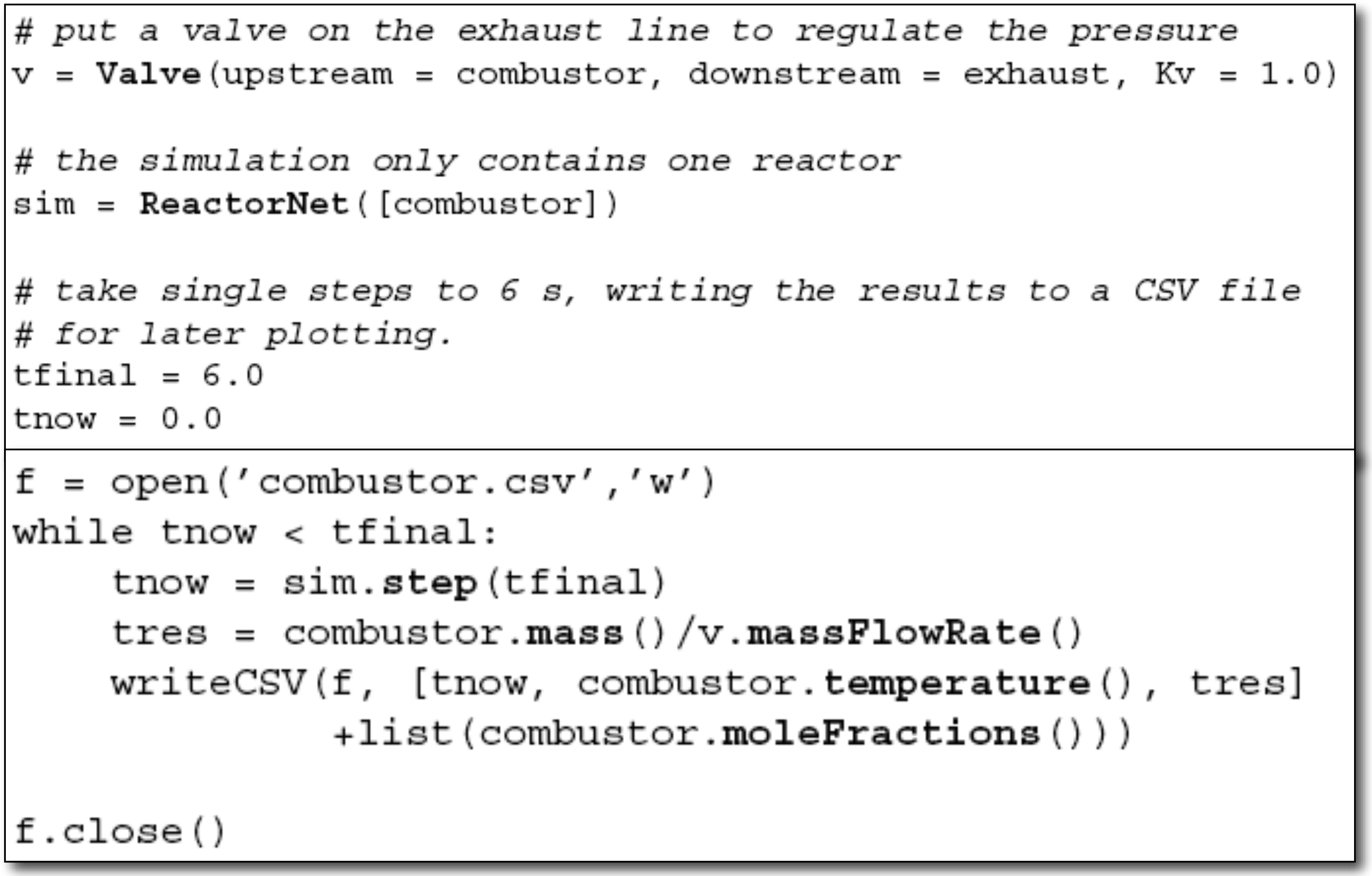
Combustor Python Script: Part II

```
# use predefined function Air() for the air inlet
air = Air()
air_in = Reservoir(air)
air_mw = air.meanMolarMass()

# create the combustor, and fill it in initially with N2
gas.set(T = 300.0, P = OneAtm, X = 'N2:1.0')
combustor = Reactor(contents = gas, volume = 1.0)

# create a reservoir for the exhaust
exhaust = Reservoir(gas)
```

Combustor Python Script: Part III

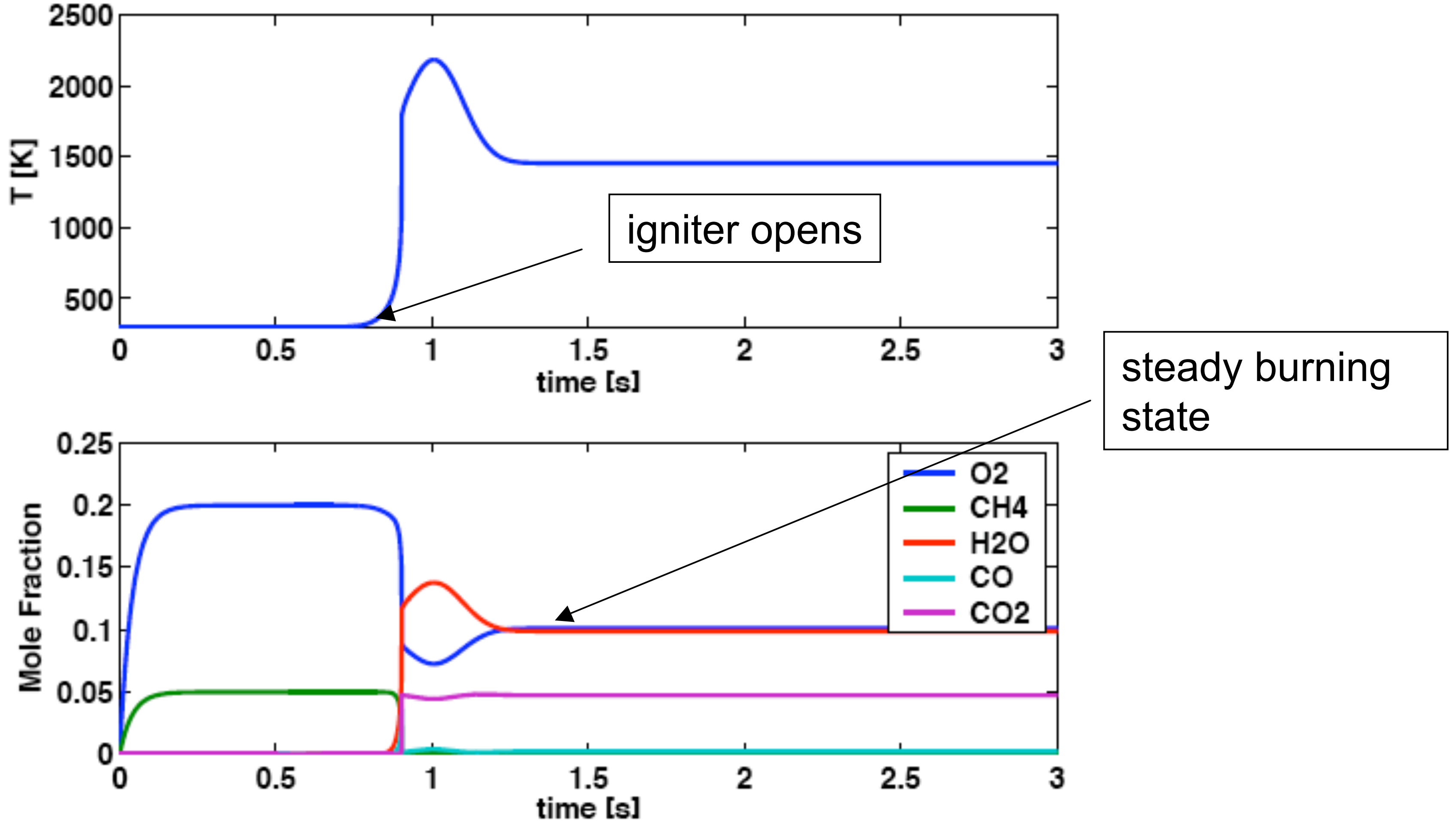
```
# lean combustion, phi = 0.5
equiv_ratio = 0.5

# compute fuel and air mass flow rates
factor = 0.1
air_mdot = factor*9.52*air_mw
fuel_mdot = factor*equiv_ratio*fuel_mw

# create and install the mass flow controllers. Controllers
# m1 and m2 provide constant mass flow rates, and m3 provides
# a short Gaussian pulse only to ignite the mixture
m1 = MassFlowController(upstream = fuel_in,
                        downstream = combustor, mdot = fuel_mdot)

m2 = MassFlowController(upstream = air_in,
                        downstream = combustor, mdot = air_mdot)

igniter_mdot = Gaussian(t0 = 1.0, FWHM = 0.2, A = 0.1)
m3 = MassFlowController(upstream = igniter,
                        downstream = combustor, mdot = igniter_mdot)
```

Combustor Python Script: Part IV



```
# put a valve on the exhaust line to regulate the pressure
v = Valve(upstream = combustor, downstream = exhaust, Kv = 1.0)

# the simulation only contains one reactor
sim = ReactorNet([combustor])

# take single steps to 6 s, writing the results to a CSV file
# for later plotting.
tfinal = 6.0
tnow = 0.0
f = open('combustor.csv','w')
while tnow < tfinal:
    tnow = sim.step(tfinal)
    tres = combustor.mass()/v.massFlowRate()
    writeCSV(f, [tnow, combustor.temperature(), tres]
             +list(combustor.moleFractions()))

f.close()
```

Combustor Example Results


igniter opens
steady burning state
T [K]
time [s]
Mole Fraction
O2
CH4
H2O
CO
CO2