Synthesis of rocket modeling:

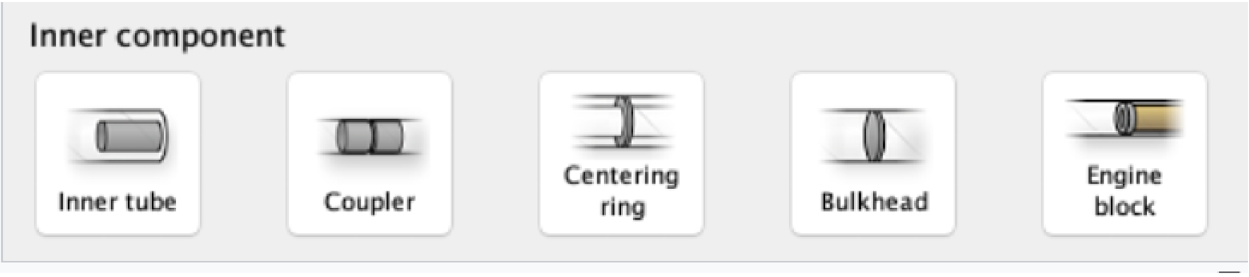
1. Mechanics
2. Variables
3. Parameters
4. Gasses

Mechanics

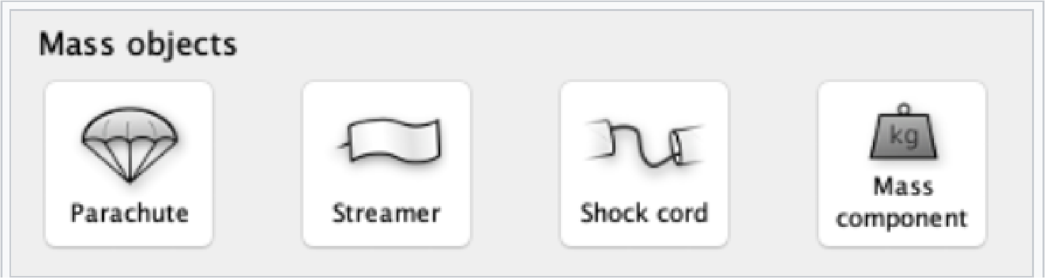
- **Assembly Components**
- **Body Components and Fin Sets**
- **Inner Components**
- **Mass Components**

Components;

- **Stage**: Every rocket has at least one stage, which is the basic framework element to which the rocket's physical components are attached. A Stage may be renamed, and has override and comment tabs.
- **Boosters**: A booster is a framework element to which physical components are attached, and may be used to build separate pieces of the rocket, such as a glider. Boosters may ONLY be attached to a body tube, and CAN separate during flight from the stage to which a booster is associated. Boosters may be renamed, and have separation, general, override, and comment tabs.
- **Pods**: A pod is a framework element to which physical components are attached, and may be used to build connected pieces of the rocket that are adjacent to the main airframe, such as side motors.



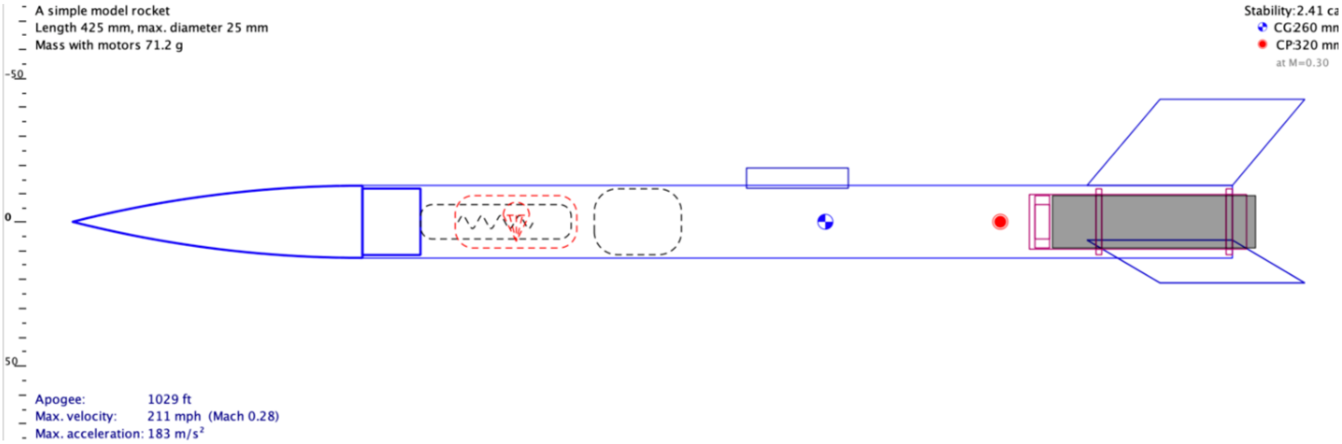
- **Parachute**: Like any good parachute, this component will stop your rocket from becoming scrap.
- **Streamer**: Another component for keeping your rocket safe, a streamer creates drag as your rocket falls down to earth.
- **Shock Cord**: A shock cord secures the nose cone to the body of the rocket so that it isn't lost when the nose is blown off to deploy the parachute/streamer.
- **Mass Component**: This is a block of mass used to adjust the rocket's Center of Gravity (CG).



Model-motors;

Estes A8-3 motor to your rocket. Now repeat these steps for the following motors, using a *New Configuration* for each new motor you add:

- **Estes B6-4**
- **Estes C6-3**
- **Estes C6-5**
- **Estes C6-7**



A simple model rocket
Length 425 mm, max. diameter 25 mm
Mass with motors 71.2 g

Stability:2.41 ca
CG:260 mn
CP:320 mn
at M=0.30

Apogee: 1029 ft
Max. velocity: 211 mph (Mach 0.28)
Max. acceleration: 183 m/s^2

After it has been added change the *Plus* value to **2**, *Packed length* to **5.2**, and *Packed diameter* to **1.2**. Again, we will add a comment to this component. Enter the following line to the *Comment* section: **The** shock cord does not need to be attached to anything in particular, as it functions only as a mass component.

Parachute. With the body tube highlighted, add a parachute component. Change *Plus* to **3.2**, *Packed length* to **4.2** and *Packed diameter* to **1.8**. That is everything we need to do to the parachute. Click *Close* to close the window. You can see what your rocket should now look like below.

Mass Component. Go ahead and add one to the body tube now. Adjust the *Mass* to **2** grams (g), the *Approximate density* to **0.16** g/cm^3, the *length* to **3.0**, the *diameter* to **2.3**

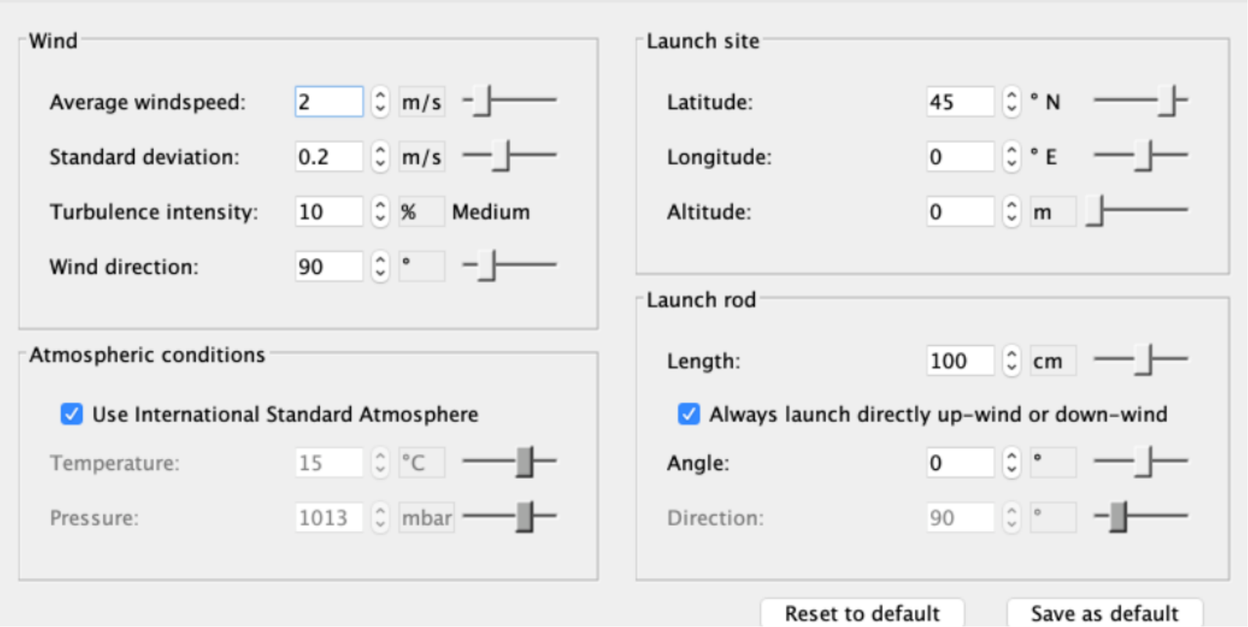
Python-scripts

X and Y axes any one of over 50 parameters:

```
Time
Altitude
Vertical velocity
Vertical acceleration
Total velocity
Total acceleration
Lateral distance
Lateral direction
Lateral velocity
Lateral acceleration
Latitude
Longitude
Gravitational acceleration
Angle of attack
Roll rate
Pitch rate
Yaw rate
Mass
Propellant mass
Longitudinal moment of inertia
Rotational moment of inertia
CP location
CG location
Stability margin calibers
Mach number
Reynolds number
Thrust
Drag force
Drag coefficient
Axial drag coefficient
```

Components;

Motor ignition, **Motor burnout**, **Apogee**, **Recovery device deployment**, and **Ground hit**.
the three Y-axis parameters described above: **Altitude**, **Vertical velocity**, and **Vertical acceleration.**



Wind

Average windspeed: 2 m/s
Standard deviation: 0.2 m/s
Turbulence intensity: 10 % Medium
Wind direction: 90 °

Atmospheric conditions

☑ Use International Standard Atmosphere
Temperature: 15 °C
Pressure: 1013 mbar

Launch site

Latitude: 45 ° N
Longitude: 0 ° E
Altitude: 0 m

Launch rod

Length: 100 cm
☑ Always launch directly up-wind or down-wind
Angle: 0 °
Direction: 90 °

Scripting-python to describe kinetics

```
gas1 combustion

>>> gas1.TPX = 1200, 101325, 'CH4:1, O2:2, N2:7.52'
>>> gas1()

>>> import cantera as ct
>>> import numpy as np
When using Cantera, the first thing you usually need is an object
representing: some phase of matter. Here, we'll create a gas mixture

>>>
>>> gas1 = ct.Solution('gri30.yaml')
To view the state of the mixture, call the gas1 object as if it were a
function:
>>> gas1()
```

You should see something like this:

```
 gri30:

      temperature   300 K
         pressure   1.0133e+05 Pa
          density   0.081894 kg/m^3
 mean mol. weight   2.016 kg/kmol
  phase of matter   gas
```

Calculate fractions-moles with phi:

```
>>> gas1.TP = 1200, 101325           # temperature, pressure
>>> gas1.TD = 1200, 0.0204723        # temperature, density
>>> gas1.HP = 1.32956e7, 101325      # specific enthalpy, pressure
>>> gas1.UV = 8.34619e6, 1/0.0204723 # specific internal energy, specific
volume
>>> gas1.SP = 85227.6, 101325        # specific entropy, pressure
>>> gas1.SV = 85227.6, 1/0.0204723   # specific entropy, specific volume

>>> II = [i for i,r in enumerate(g.reactions())
...       if 'CO' in r.reactants and 'CO2' in r.products]
>>> for i in II:
...     print(g.reaction(i).equation)
...
CO + O (+M) <=> CO2 (+M)
CO + O2 <=> CO2 + O
CO + OH <=> CO2 + H
CO + HO2 <=> CO2 + OH
```

We should also include reactions where the reaction is written such that CO2 is a reactant and CO is a product, but for this example, we'll just stick to this smaller set of reactions.

Reactions-chemical Kinetics

```
rxns = '''
  - equation: O + H2 <=> H + OH
    rate-constant: {A: 3.87e+04, b: 2.7, Ea: 6260.0}
  - equation: O + HO2 <=> OH + O2
    rate-constant: {A: 2.0e+13, b: 0.0, Ea: 0.0}
'''
```

combustion efficiencies

default_efficiency
Get/Set the default third-body efficiency for this reaction, used for species not in.

net_production_rates. The list of selected species can be set by name or index. This property returns the species by index.:

```
gas.selected_species = ["H2", "O2"]
print(gas.molecular_weights)
[ 2.016 31.998]
```

This method is often used implicitly by using an indexing expression on a

Solution object:

```
print(gas["H2", "O2"].molecular_weights)
[ 2.016 31.998]
```

gas-solution:

```
>>> q1.volume
7.110450052249231
>>> q1.enthalpy
1032229.2296838651
>>> q1.moles = 3
>>> q1.mass
86.55292436974788
>>> q1.volume
123.08604912143952
>>> q1 *= 2
>>> q1.moles
6.0
```

Constants-gasses;

```
>>> q3.mole_fraction_dict()
{'CH4': 0.2645277305363934, 'N2': 0.5809612884838573, 'O2':
0.15451098097974925}

q1.constant = q2.constant = 'HP'
q3 = q1 + q2 # combine at constant HP
q3.T
436.03320
q3.P
101325.0
```

Phases-coefficients gas

```
phase.Y = [0.1, 0, 0, 0.4, 0, 0, 0, 0, 0.5]
phase.Y = {'H2':0.1, 'O2':0.4, 'AR':0.5}
phase.Y = 'H2:0.1, O2:0.4, AR:0.5'
phase.Y
array([0.1, 0, 0, 0.4, 0, 0, 0, 0, 0.5])
```

Density

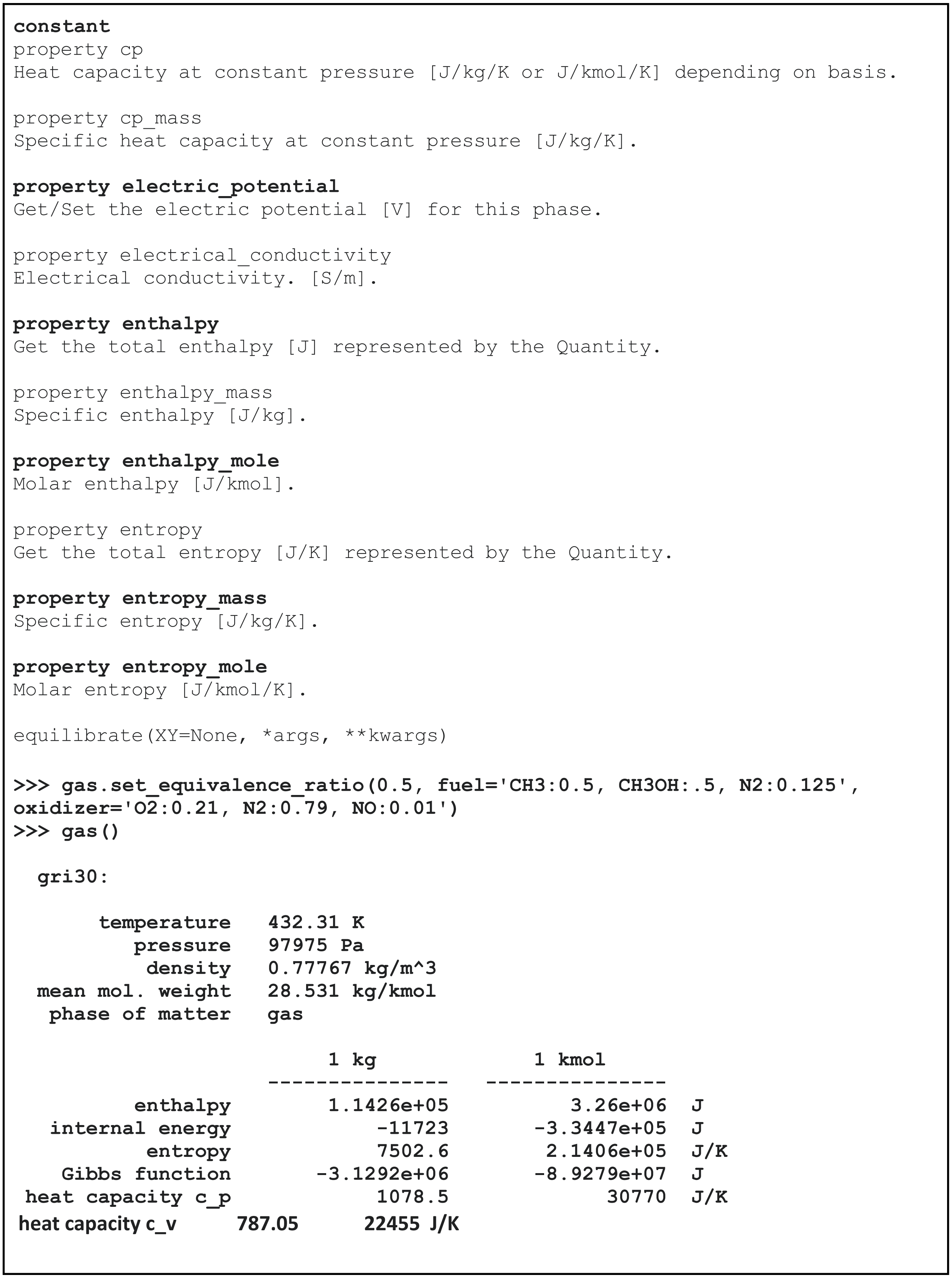
property binary_diff_coeffs
Binary diffusion coefficients [m^2/s].

property case_sensitive_species_names
Enforce case-sensitivity for look up of species names

property charges
Array of species charges [elem. charge].

property chemical_potentials
Array of species chemical potentials [J/kmol].

property concentrations
Get/Set the species concentrations. Units are kmol/m^3 for bulk phases, kmol/m^2 for surface **phases**



constant

property cp

Heat capacity at constant pressure [J/kg/K or J/kmol/K] depending on basis.

property cp_mass

Specific heat capacity at constant pressure [J/kg/K].

property electric_potential

Get/Set the electric potential [V] for this phase.

property electrical_conductivity

Electrical conductivity. [S/m].

property enthalpy

Get the total enthalpy [J] represented by the Quantity.

property enthalpy_mass

Specific enthalpy [J/kg].

property enthalpy_mole

Molar enthalpy [J/kmol].

property entropy

Get the total entropy [J/K] represented by the Quantity.

property entropy_mass

Specific entropy [J/kg/K].

property entropy_mole

Molar entropy [J/kmol/K].

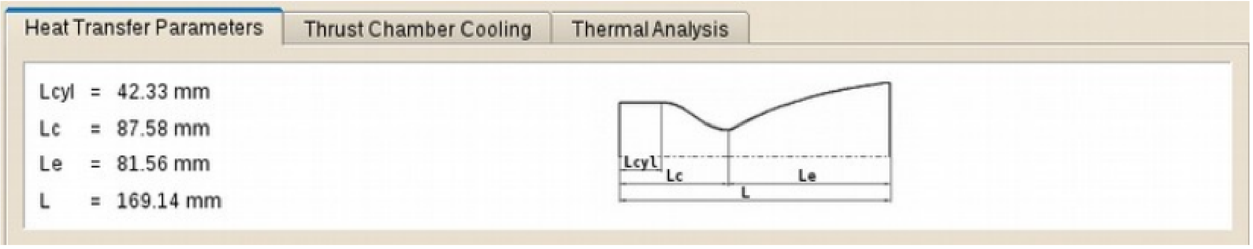
equilibrate(XY=None, *args, **kwargs)

```
>>> gas.set_equivalence_ratio(0.5, fuel='CH3:0.5, CH3OH:.5, N2:0.125',
oxidizer='O2:0.21, N2:0.79, NO:0.01')
>>> gas()

  gri30:

       temperature   432.31 K
          pressure   97975 Pa
           density   0.77767 kg/m^3
  mean mol. weight   28.531 kg/kmol
   phase of matter   gas

                          1 kg             1 kmol
                     ---------------   ---------------
          enthalpy        1.1426e+05         3.26e+06  J
   internal energy            -11723      -3.3447e+05  J
           entropy            7502.6       2.1406e+05  J/K
    Gibbs function       -3.1292e+06      -8.9279e+07  J
 heat capacity c_p            1078.5            30770  J/K
 heat capacity c_v      787.05       22455 J/K
```

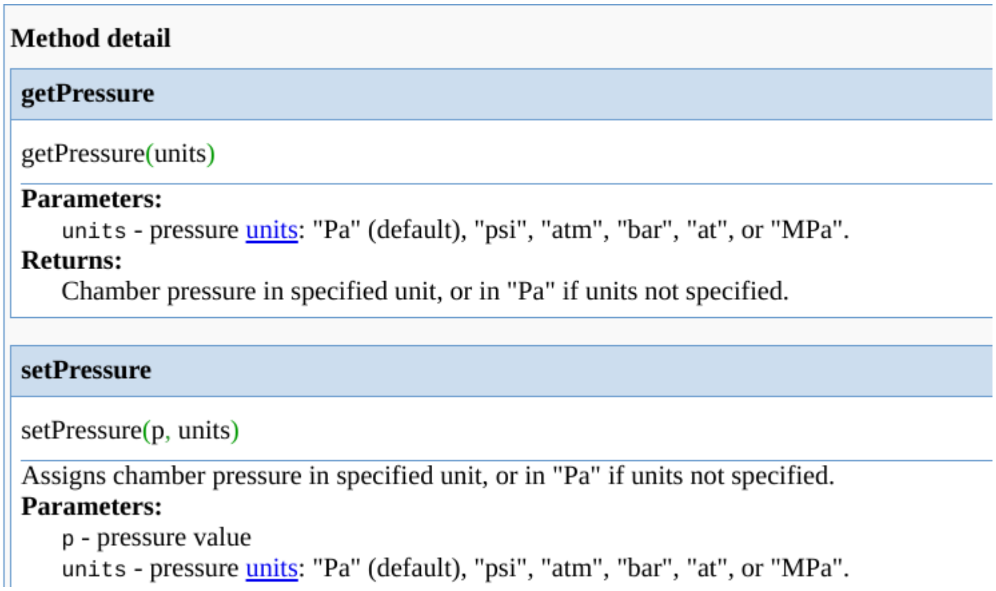


Lcyl = 42.33 mm
Lc = 87.58 mm
Le = 81.56 mm
L = 169.14 mm

Conditions-parameters;

Ambient condition By default the program calculates performance of the rocket engine at the sea level conditions (pa=1 atm, or 14.7 psi), optimum nozzle expansion (pe=pa), and vacuum conditions (pa=0). To calculate the performance at desired ambient conditions, you can also explicitly specify either the specific ambient pressure or the range of ambient pressures given as high and low range values.

The pressure is an absolute pressure and can be entered using one of the following units: MPa, atm, kg/sm2 , bar, psi, Pa.



Method detail

getPressure

getPressure(units)

Parameters:
`units` - pressure units: "Pa" (default), "psi", "atm", "bar", "at", or "MPa".
Returns:
Chamber pressure in specified unit, or in "Pa" if units not specified.

setPressure

setPressure(p, units)

Assigns chamber pressure in specified unit, or in "Pa" if units not specified.
Parameters:
`p` - pressure value
`units` - pressure units: "Pa" (default), "psi", "atm", "bar", "at", or "MPa".

The gas-side heat transfer rates: Ievlev approach for calculation of convective heat transfer in the nozzle and Bartz semi empirical correlation for gas-side heat transfer coefficient.

Propellant analysis in code

```
mix = Mixture();
mix.addSpecies("O2(L)", 0.8);    // Add 1st component at it's normal
temperature and atmospheric pressure
mix.addSpecies("H2(L)", 0.15);   // Add 2nd component at it's normal
```

```
temperature and atmospheric pressure
mix.addSpecies("RP-1", 0, "K", 3, "atm", 0.03); // Add 3rd component at it's
normal temperature and pressure 3 atm
mix.addSpecies("AL(cr)", 0.02);  // Add 4th component at it's normal
temperature and atmospheric pressure

// Total mass fraction of components #2 (RP-1) and #3 (AL(cr))
sf = mix.getFraction(2) + mix.getFraction(3);

// Array with different values of AL(cr) mass fraction
m = Array();
for (i=0; i<=1.0; i+=0.2) {
    m[m.length] = sf*i;
}

// Print out table header
printf("#%6s %6s %8s %8s %8s", "RP-1", "AL(cr)", "Is_v,s", "Is_opt,s",
"Is_sl,s");

for (i=0; i<m.length; ++i) {
    // Change mass fractions of components #2 (RP-1) and #3 (AL(cr))
    mix.setFraction(2, sf - m[i]);
    mix.setFraction(3, m[i]);

    chamber = Chamber(mix);
    chamber.setP(10, "MPa");    // Chamber pressure
    chamber.setFcr(3);          // Nozzle inlet contraction area ratio
    chamber.solve(true);        // finiteChamberSection=true

    nozzleExit = NozzleSectionConditions(chamber, 40, "A/At", true);

    printf(" %6.3f %6.3f %8.2f %8.2f %8.2f",
        mix.getFraction(2),
        mix.getFraction(3),
        nozzleExit.getIs_v("s"),
        nozzleExit.getIs("s"),
        nozzleExit.getIs_H(1, "atm", "s")
    );
```