Warsaw University of Technology | Faculty of Power and Aeronautical Engineering

Institute of Heat Engineering

Semester project
for "Computer Methods in Combustion"

Comparison of fuel efficiency and feasibility study of a GDI engine

Rodak Michał
Wąsowski Witold

supervisor
PhD. Eng. Mateusz Żbikowski

Warsaw, 2020

Abstract

A project for "Computer Methods in Combustion" classes, where students get familiar with the principles of simulating the chemical processes taking place inside internal combustion engines. This paper summarizes the authors' attempt to compare various fuels efficiency in two different environments in GDI engine and the numerical study was conducted with open source Cantera software.

Keywords: Cantera, combustion, GDI, simulation

Contents

1 Introduction

In this project, the basic assumption is to test an engine adapted for petrol supply, with a system of direct fuel injection into the combustion chamber (GDI). Engines equipped with such system are generally more efficient and achieve better performance than engines fuelled indirectly (with fuel injected in the inlet mainfold). Since most of the engines currently produced in the world are fuelled directly, we have chosen this as a more modern solution.

In the combustion chamber of a directly fuelled engine, the fuel-air mixture can be either homogeneous or stratified, with a higher fuel density around the injector and spark plug and leaner in the remaining volume. Since the simulation is based on a zero-dimensional model, we can assume that the mixture takes the first form.

HCCI (Homogeneous Charge Compression Ignition) is a form of internal combustion in piston engines. This ignition type combines the characteristics of gasoline and diesel engines because a homogeneous mixture is injected into the cylinder in the intake stroke, but ignites spontaneously after compression to the auto-ignition point. According to source HCCI engines are fuel-lean, so they can operate at diesel-like compression ratios (>15), thus achieving 30% higher efficiencies than conventional SI gasoline engines. Because of the homogeneous mixture, it is possible to achieve cleaner combustion and thus, lower maximum temperature in the cylinder and lower emissions. On the other hand, auto-ignition is difficult to control and if it occurs too early or with too much chemical energy, combustion is too fast and cylinder pressure can destroy an engine. Additionally, high heat release and pressure rise rates contribute to faster engine wear.

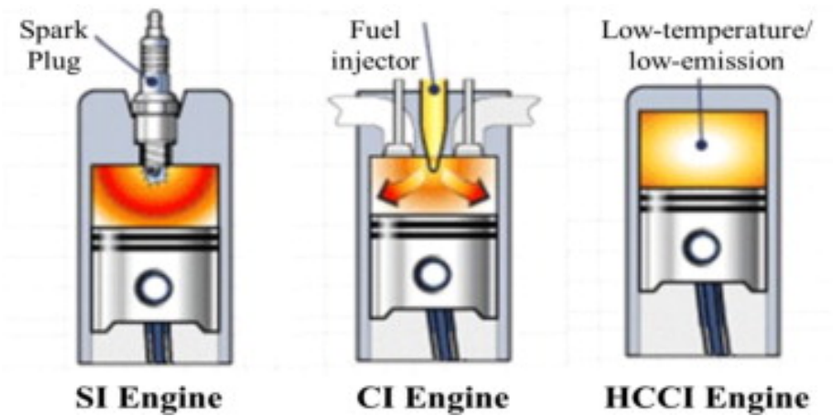

Figure 1: Comparison of Spark Ignition, Compression Ignition and Homogeneous Charge Compression Ignition engine.

Source: https://www.sciencedirect.com/science/article/abs/pii/S1364032114004717

A few global automotive manufacturers, including Mercedes-Benz and Mazda have experimented with HCCI engines, with the first prototype being demonstrated in 2007. In 2017, Mazda has launched first mass-produced engine with HCCI capability, SKYACTIV-X.

2 Aim of the project

The main objective of the project is to analyse the feasibility by various fuels' theoretical efficiency in two different environments as well as to study maximum pressure and temperature levels inside the cylinder during the combustion.

The reference will be the information obtained for the combustion of the mixture of octane and air, due to the assumption that the simulated engine is factory-adapted for such supply.

The information collected will allow an analysis to be made of what technical requirements an engine running on a given fuel would have to meet and whether this would be cost-effective in terms of performance.

3 Model description

Model used in this project is a simple network of reservoirs and reactor created using Cantera, very similar to the example one provided by software creators. Engine cylinder is a zero-dimensional ideal gas reactor, thus it follows the theoretical assumption that the mixture of air and fuel inside is homogeneous. Inlet, injector, outlet and ambient air are implemented as reservoirs and piston is modeled as a wall. To simplify the simulation, inlet and outlet valves do not overlap. The fuel injector is modeled as a mass flow controller device.

All of the model elements are shown on a figure below:

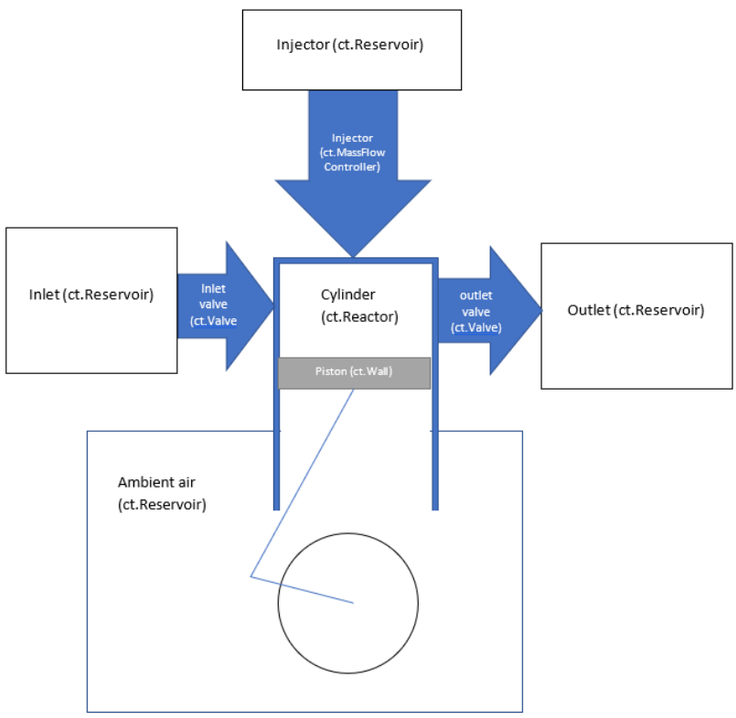

Figure 2: Block diagram of model used in simulation.

In this study there are three fuels compared: propane (C3H8), hydrogen (H2) and octane (C8H18), with the last one being used as a standard gasoline representative. In the first series of simulations all fuels have been mixed with air and in the second one with nitrous oxide (N2O).

Unfortunately, the most-commonly used GRI 3.0 thermodynamics mechanism designed to model natural gas combustion does not provide reactions for octane. They have been added basing on information provided by French Basic and Applied Research Center (CERFACS) source and the new mechanism has been implemented in the model as “gri30-22”.

The simulation was performed at two engine speeds: 1500 and 3000 rpm. All data, including geometrical dimensions, are in the python code attached to this paper.

4 Results

To check if the model simulates proper engine cycle, the following pressure-volume plot was obtained:

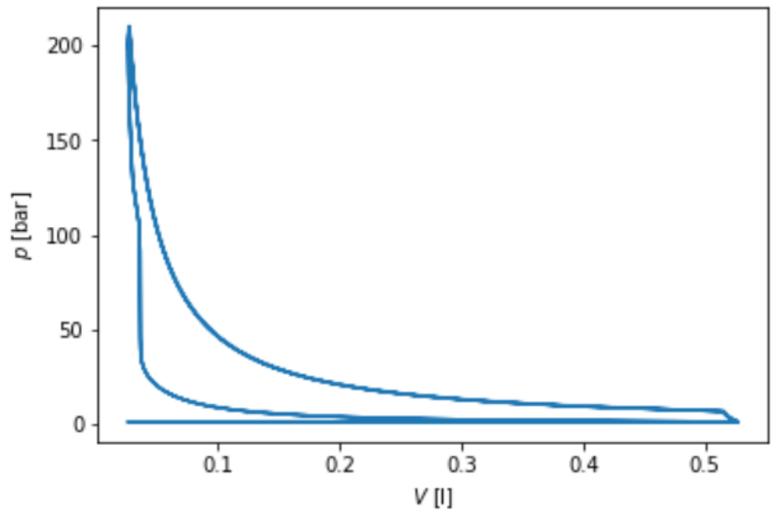

Figure 3: Example pressure-volume plot obtained, with octane used as fuel and nitrous oxide as oxidiser, at crank speed 1500 rpm.

As a result of calculations, following results were obtained:

4.1 Results at 1500 RPM

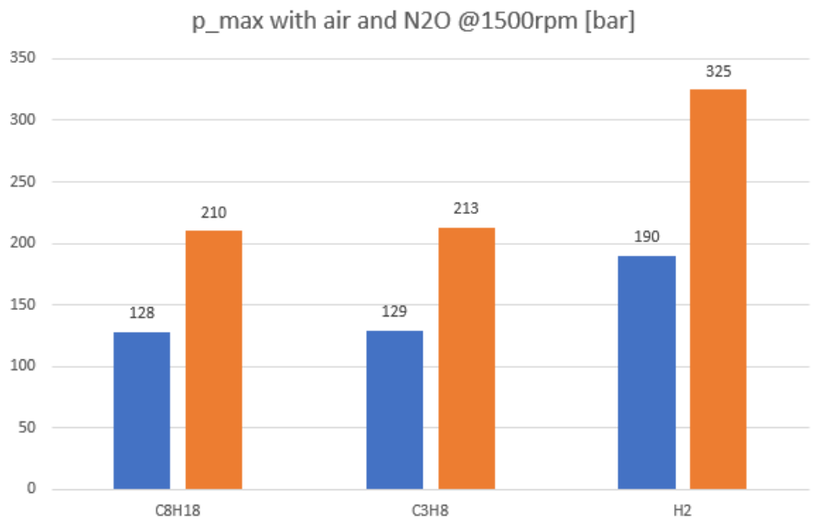

Figure 4: Comparison of maximum pressure [bar] in the cylinder for all used fuels and oxidisers, at crank speed 1500 rpm.

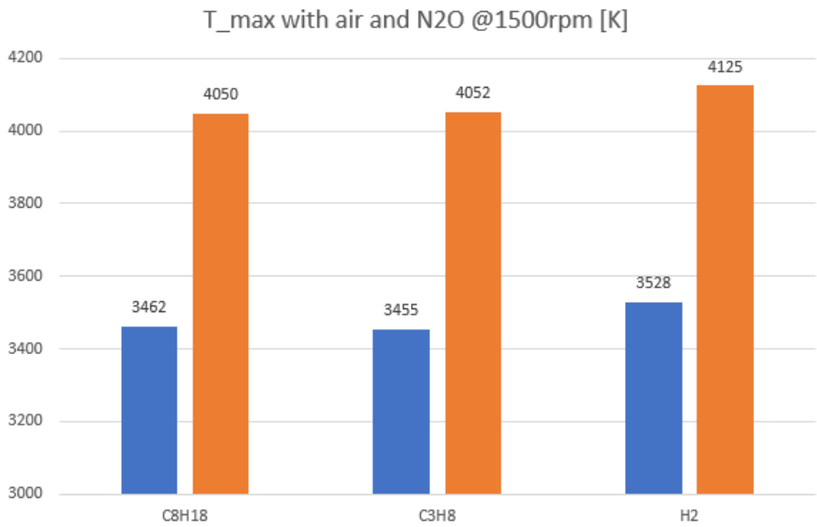

Figure 5: Comparison of maximum temperature [K] in the cylinder for all used fuels and oxidisers, at crank speed 1500 rpm.

4.2 Results for 3000 RPM

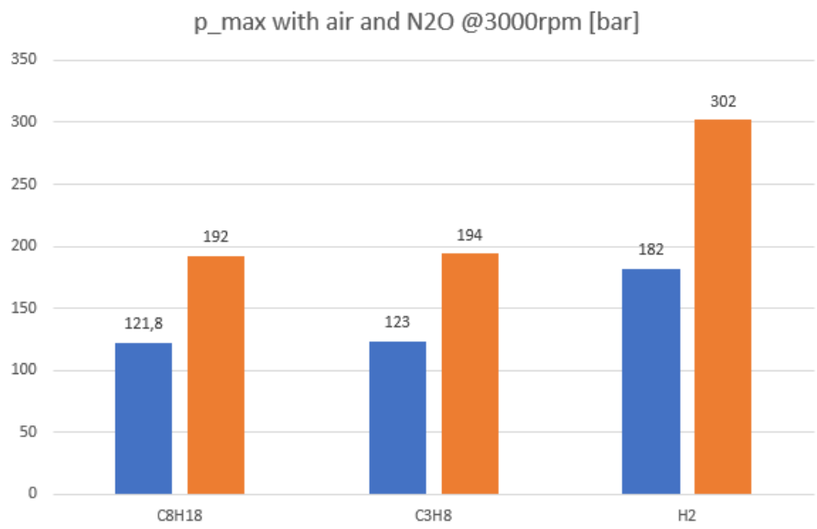

Figure 6: Comparison of maximum pressure [bar] in the cylinder for all used fuels and oxidisers, at crank speed 3000 rpm.

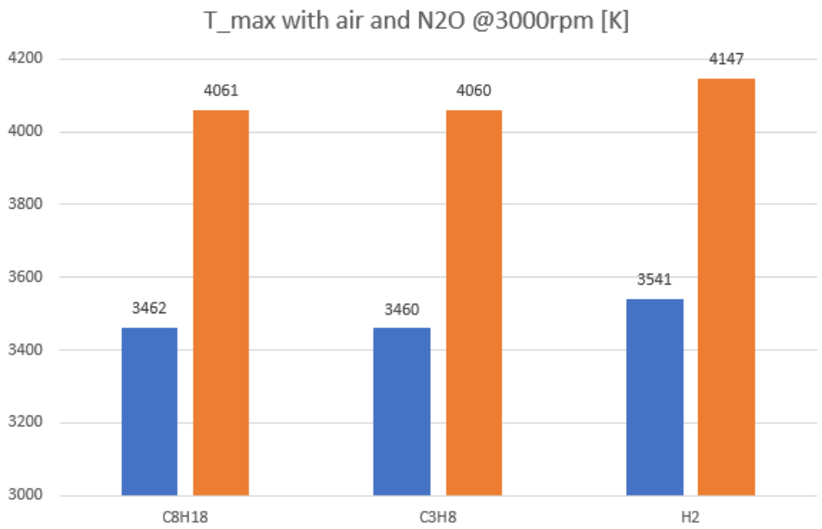

Figure 7: Comparison of maximum temperature [K] in the cylinder for all used fuels and oxidisers, at crank speed 3000 rpm.

4.3 Achieved power comparison

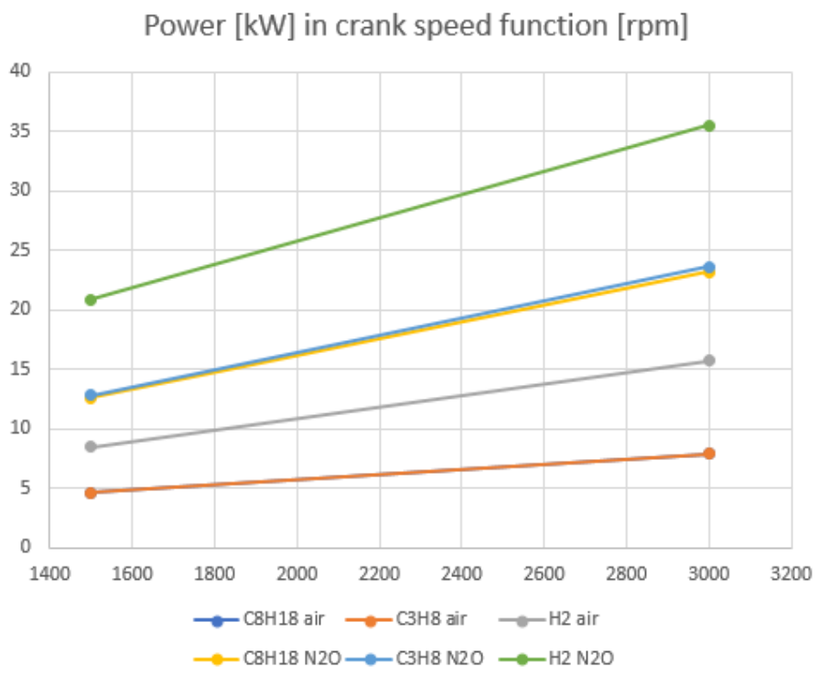

Figure 8: Comparison of maximum usable power [kW] in the cylinder for all used fuels and oxidisers, at different crank speeds.

5 Summary

As can be seen in the graphs in the "Results" section, the operating parameters and performance obtained during operation on oil-based fuels are very similar. The engine achieves higher efficiency when burning propane, but the power obtained in the cylinder is slightly higher when burning octane. However, the highest both efficiency and power is achieved when operating on hydrogen.

Series	Fuel	Oxidiser	Crank speed [rpm]	Heat release per cylinder [kW]	Exp. Power per cylinder [kW]
1	C8H18	air	1500	9.1	4.6
2	C8H18	air	3000	15.9	7.9
3	C3H8	air	1500	8.7	4.6
4	C3H8	air	3000	15.1	7.9
5	H2	air	1500	12.8	8.5
6	H2	air	3000	23.8	15.7
7	C8H18	N2O	1500	28.3	12.6
8	C8H18	N2O	3000	52.8	23.2
9	C3H8	N2O	1500	28.1	12.8
10	C3H8	N2O	3000	52.6	23.6
11	H2	N2O	1500	40	20.9
12	H2	N2O	3000	65.6	35.5

Table 1: Results obtained - part 1.

Series	Efficiency [%]	CO emission [ppm]	p_max [bar]	T_max [K]
1	50.5%	131693	128	3462
2	49.7%	154422	121.8	3462
3	52.9%	128141	129	3455
4	52.3%	148734	123	3460
5	66.4%	0	190	3528
6	66.0%	0	182	3541
7	44.5%	10707	210	4050
8	43.9%	20196	192	4061
9	45.6%	10997	213	4052
10	44.9%	21407	194	4060
11	52.3%	0	325	4125
12	54.1%	0	302	4147

Table 2: Results obtained - part 2.

The temperature in the cylinder depends mainly on the oxidant and is on average 17% higher when operating on a nitrogen oxide mixture than when operating on a mixture with air. However, a significant constructional obstacle can be the pressure, which for hydrogen combustion is even 57% higher than for octane combustion and exceeds 300 bar.

6 List of figures

List of Figures

7 Sources

- https://cantera.org/examples/python/reactors/ic_engine.py.html
- https://www.cerfacs.fr/cantera/docs/mechanisms/fuel-IFP/2S_C8H18_BE2/2S_C8H18_BE2.cti
- Zhao, Fuquan; Asmus, Thomas W.; Assanis, Dennis N.; Dec, John E.; Eng, James A.; Najt, Paul M. (2003). Homogeneous Charge Compression Ignition (HCCI) Engines: Key Research and Development Issues. Warrendale, PA, USA: Society of Automotive Engineers.
- Baumgarten, Carsten (2006). Mixture Formation in Internal Combustion Engines: Mixture Formation in Internal Combustion Engines. Birkhäuser.
- Stanglmaier, Rudolf H.; Roberts, Charles E. (1999). "Homogeneous Charge Compression Ignition (HCCI): Benefits, Compromises, and Future Engine Applications". SAE Technical Paper

8 Example python code

```
# -*- coding: utf-8 -*-
"""
Simulation of a (gaseous) Diesel-type internal combustion engine.
"""

import cantera as ct
import numpy as np

#########################################################################
# Input Parameters
#########################################################################

f = 3000. / 60.  # engine speed [1/s] (3000 rpm)
V_H = .5e-3  # displaced volume [m**3]
epsilon = 20.  # compression ratio [-]
d_piston = 0.083  # piston diameter [m]

# turbocharger temperature, pressure, and composition
T_inlet = 800.  # K
p_inlet = 1.3e5  # Pa
comp_inlet = 'N2O:1'

# outlet pressure
p_outlet = 1.2e5  # Pa

# fuel properties (gaseous!)
T_injector = 300.  # K
p_injector = 1600e5  # Pa
comp_injector = 'C3H8:1'

# ambient properties
T_ambient = 300.  # K
p_ambient = 1e5  # Pa
comp_ambient = 'O2:1, N2:3.76'

# Reaction mechanism name
reaction_mechanism = 'gri3022.cti'

# Inlet valve friction coefficient, open and close timings
inlet_valve_coeff = 1.e-6
```

```
inlet_open = -18. / 180. * np.pi
inlet_close = 198. / 180. * np.pi

# Outlet valve friction coefficient, open and close timings
outlet_valve_coeff = 1.e-6
outlet_open = 522. / 180 * np.pi
outlet_close = 18. / 180. * np.pi

# Fuel mass, injector open and close timings
injector_open = 350. / 180. * np.pi
injector_close = 365. / 180. * np.pi
injector_mass = 3.2e-5  # kg
injector_t_open = (injector_close - injector_open) / 2. / np.pi / f

# Simulation time and resolution
sim_n_revolutions = 8.
sim_n_timesteps = 100000.

########################################################################

# load reaction mechanism
gas = ct.Solution(reaction_mechanism)

# define initial state
gas.TPX = T_inlet, p_inlet, comp_inlet
r = ct.IdealGasReactor(gas)
# define inlet state
gas.TPX = T_inlet, p_inlet, comp_inlet
inlet = ct.Reservoir(gas)
# define injector state (gaseous!)
gas.TPX = T_injector, p_injector, comp_injector
injector = ct.Reservoir(gas)
# define outlet pressure (temperature and composition don't matter)
gas.TPX = T_ambient, p_outlet, comp_ambient
outlet = ct.Reservoir(gas)
# define ambient pressure (temperature and composition don't matter)
gas.TPX = T_ambient, p_ambient, comp_ambient
ambient_air = ct.Reservoir(gas)

# set up connecting devices
inlet_valve = ct.Valve(inlet, r)
injector_mfc = ct.MassFlowController(injector, r)
outlet_valve = ct.Valve(r, outlet)
piston = ct.Wall(ambient_air, r)

# convert time to crank angle
def crank_angle(t):
    return np.remainder(2 * np.pi * f * t, 4 * np.pi)

# set up IC engine parameters
V_oT = V_H / (epsilon - 1.)
A_piston = .25 * np.pi * d_piston ** 2
stroke = V_H / A_piston
r.volume = V_oT
piston.area = A_piston
def piston_speed(t):
    return - stroke / 2 * 2 * np.pi * f * np.sin(crank_angle(t))
piston.set_velocity(piston_speed)
```

```
# create a reactor network containing the cylinder
sim = ct.ReactorNet([r])

# set up output data arrays
states = ct.SolutionArray(r.thermo)
t_sim = sim_n_revolutions / f
t = (np.arange(sim_n_timesteps) + 1) / sim_n_timesteps * t_sim
V = np.zeros_like(t)
m = np.zeros_like(t)
test = np.zeros_like(t)
mdot_in = np.zeros_like(t)
mdot_out = np.zeros_like(t)
d_W_v_d_t = np.zeros_like(t)
heat_release_rate = np.zeros_like(t)

# set parameters for the automatic time step refinement
n_last_refinement = -np.inf  # for initialization only
n_wait_coarsening = 10

# do simulation
for n1, t_i in enumerate(t):
    # define opening and closing of valves and injector
    if (np.mod(crank_angle(t_i) - inlet_open, 4 * np.pi) <
            np.mod(inlet_close - inlet_open, 4 * np.pi)):
        inlet_valve.set_valve_coeff(inlet_valve_coeff)
        test[n1] = 1
    else:
        inlet_valve.set_valve_coeff(0)
    if (np.mod(crank_angle(t_i) - outlet_open, 4 * np.pi) <
            np.mod(outlet_close - outlet_open, 4 * np.pi)):
        outlet_valve.set_valve_coeff(outlet_valve_coeff)
    else:
        outlet_valve.set_valve_coeff(0)
    if (np.mod(crank_angle(t_i) - injector_open, 4 * np.pi) <
            np.mod(injector_close - injector_open, 4 * np.pi)):
        injector_mfc.set_mass_flow_rate(injector_mass / injector_t_open)
    else:
        injector_mfc.set_mass_flow_rate(0)

    # perform time integration, refine time step if necessary
    for n2 in range(4):
        if n2 is 4:
            raise 'Error: Refinement limit reached'
        try:
            sim.advance(t_i)
        except Exception:
            sim.set_max_time_step(1e-6 * 10. ** -n2)
            n_last_refinement = n1
    # coarsen time step if too long ago
    if n1 - n_last_refinement is n_wait_coarsening:
        sim.set_max_time_step(1e-5)

    # write output data
    states.append(r.thermo.state)
    V[n1] = r.volume
    m[n1] = r.mass
    mdot_in[n1] = inlet_valve.mdot(0)
    mdot_out[n1] = outlet_valve.mdot(0)
    d_W_v_d_t[n1] = - (r.thermo.P - ambient_air.thermo.P) * A_piston * \
```

```
        piston_speed(t_i)
    heat_release_rate[n1] = - r.volume * ct.gas_constant * r.T * \
        np.sum(gas.standard_enthalpies_RT * r.thermo.net_production_rates, 0)


########################################################################
# Plot Results in matplotlib
########################################################################

import matplotlib.pyplot as plt

# pressure and temperature
plt.clf()
plt.subplot(211)
plt.plot(t, states.P / 1.e5)
plt.ylabel('$p$ [bar]')
plt.xlabel('$\phi$ [deg]')
plt.xticks(plt.xticks()[0], [])
plt.subplot(212)
plt.plot(t, states.T)
plt.ylabel('$T$ [K]')
plt.xlabel('$\phi$ [deg]')
plt.xticks(plt.xticks()[0], crank_angle(plt.xticks()[0]) * 180 / np.pi,
           rotation=17)
plt.show()
plt.savefig('ic_engine_t_p_T.png')

# p-V diagram
plt.clf()
plt.plot(V[t > 0.04] * 1000, states.P[t > 0.04] / 1.e5)
plt.xlabel('$V$ [l]')
plt.ylabel('$p$ [bar]')
plt.show()
plt.savefig('ic_engine_p_V.png')

# T-S diagram
plt.clf()
plt.plot(m[t > 0.04] * states.s[t > 0.04], states.T[t > 0.04])
plt.xlabel('$S$ [J/K]')
plt.ylabel('$T$ [K]')
plt.show()
plt.savefig('ic_engine_T_S.png')


# gas composition
plt.clf()
#plt.plot(t, states('O2').X, label='O2')
#plt.plot(t, states('N2O').X, label='N2O')
#plt.plot(t, states('N2').X, label='N2')
plt.plot(t, states('CO2').X, label='CO2')
#plt.plot(t, states('CO').X, label='CO')
plt.plot(t, states('C8H18').X * 10, label='C8H18 x10')
plt.legend(loc=0)
plt.ylabel('$X_i$ [-]')
plt.xlabel('$\phi$ [deg]')
plt.xticks(plt.xticks()[0], crank_angle(plt.xticks()[0]) * 180 / np.pi,
           rotation=17)
plt.show()
plt.savefig('ic_engine_t_X.png')
```

```
#########################################################################
# Integral Results
#########################################################################

from scipy.integrate import trapz
Q = trapz(heat_release_rate, t)
W = trapz(d_W_v_d_t, t)
eta = W / Q
MW = states.mean_molecular_weight
CO_emission = trapz(MW * mdot_out * states('CO').X[:,0], t) / trapz(MW * mdot_out, t)
print('Heat release rate per cylinder (estimate):\t' +
      format(Q / t_sim / 1000., ' 2.1f') + ' kW')
print('Expansion power per cylinder (estimate):\t' +
      format(W / t_sim / 1000., ' 2.1f') + ' kW')
print('Efficiency (estimate):\t\t\t' + format(eta * 100., ' 2.1f') + ' %')
print('CO emission (estimate):\t\t' + format(CO_emission * 1.e6, ' 2.1f') +
      ' ppm')
print('p_max:'+max(states.P/10e4)+'bar')
print('T_max:'+max(states.T)+'K')'
```