ire-3-5-test-site-data-analysis-1

August 28, 2024

[99]:
```
import os
import math
import datetime
import numpy as np
import pandas as pd
from math import pi
import scipy.integrate as spi
from scipy.integrate import quad
from matplotlib import pyplot as plt
from scipy.signal import savgol_filter
from scipy.ndimage import median_filter
from scipy.optimize import minimize_scalar
from scipy.ndimage import gaussian_filter1d
from scipy.interpolate import UnivariateSpline
from sklearn.linear_model import LinearRegression
from matplotlib.ticker import (AutoMinorLocator, MultipleLocator)
```

0.1 1. Define Constants

[100]:
```
# nitrous constants
p_c = 7251                                                              # kPa
T_c = 309.57                                                            # Kelvin
rho_c = 452                                                             # kg/m^3

# 4.1 constants
b1 = -6.71893
b2 = 1.35966
b3 = -1.3779
b4 = -4.051

# 4.2 constants
b1_l = 1.72328
b2_l = -0.83950
b3_l = 0.51060
b4_l = -0.10412

# 4.3 constants
```

```
b1_g = -1.00900
b2_g = -6.28792
b3_g = 7.50332
b4_g = -7.90463
b5_g = 0.629427
```

[101]:
```
# run tank dimensions
inner_diameter = 0.146304                                      # m
cyl_height = 0.912114                                          # m
V_total = 0.01533                                              # m^3
base_area = np.pi * ((inner_diameter/2)**2)                    # m^2
```

[102]:
```
# injector
hole_diameter = 2                                              # mm
num_holes = 16
```

[103]:
```
# nozzle
A = 0.0011
gamma = 1.2661
R = 375.5075                                                   # specific gas␣
 ↪constant of mixture
T_o = 2842                                                     # K
```

[104]:
```
# fuel grain
m_fuel = 1.310                                                 # kg
m_fuel_uncertainty = 0.0005                                    # kg
```

[105]:
```
# input filename
input_data = 'HOTFIRE3.5.txt'
```

[106]:
```
# ambient temp
outdoor_temp = 17                                              # Celcius
```

0.2 2. Define Functions

[107]:
```
# get data in time range required
def get_data_range(data, time, start, end):
        return [d for t, d in zip(time, data) if start <= t <= end]
```

[108]:
```
# calculate total impulse by numerically integrating the thrust values with␣
 ↪respect to time
def calculate_total_impulse(time, thrust):
    def integrand(t):
        return np.interp(t, time, thrust)
    total_impulse = 0
    for i in range(len(time) - 1):
```

```
        total_impulse += spi.quad(integrand, time[i], time[i+1])[0]
    return total_impulse
```

[109]:
```
# determine nitrous vapour pressure, liquid density and gas density given␣
↪temperature
def nitrous_equations(temp_kelvin):
    T_r = temp_kelvin/T_c

    # 4.1 Vapour Pressure
    p = p_c * math.exp((1/T_r) * ((b1 * (1-T_r)) + (b2 * (1-T_r)**(3/2)) + (b3␣
↪* (1-T_r)**(5/2)) + (b4 * (1-T_r)**5)))

    # 4.2 Density of Saturated Liquid
    p_l = rho_c * math.exp((b1_l * (1-T_r)**(1/3)) + (b2_l * (1-T_r)**(2/3)) +␣
↪(b3_l * (1-T_r)) + (b4_l * (1-T_r)**(4/3)))

    # 4.3 Density of Saturated Gas
    p_g = rho_c * math.exp((b1_g * ((1/T_r)-1)**(1/3)) + (b2_g * ((1/
↪T_r)-1)**(2/3)) + (b3_g * ((1/T_r)-1)) + (b4_g * ((1/T_r)-1)**(4/3)) + (b5_g␣
↪* ((1/T_r)-1)**(5/3)))

    return p, p_l, p_g

# turn nitrous vapour pressure, liquid density and gas density into lists
def nitrous_equation_lists(temp_list):
    vap_list = []
    liquid_list = []
    gas_list = []
    for temp_kelvin in temp_list:
        p, p_l, p_g = nitrous_equations(temp_kelvin)
        vap_list.append(p)
        liquid_list.append(p_l)
        gas_list.append(p_g)

    return vap_list, liquid_list, gas_list
```

[110]:
```
# Determine liquid density from temperature. Finds key from closest value in␣
↪dictionary using list comprehension
def get_liquid_density_from_pressure(temp_liquid_density_dict,␣
↪pressure_temp_dict, pressure_key):
    pressure_KPa = pressure_key * 6.89476
    temp_K = pressure_temp_dict.get(pressure_KPa) or␣
↪pressure_temp_dict[min(pressure_temp_dict.keys(), key = lambda key:␣
↪abs(key-pressure_KPa))]
```

```
    liquid_density = temp_liquid_density_dict.get(temp_K) or␣
↪temp_liquid_density_dict[min(temp_liquid_density_dict.keys(), key = lambda␣
↪key: abs(key-temp_K))]

    return liquid_density
```

[162]:
```
# determine discharge coefficient given mass flow rate and combustion chamber␣
↪pressure
def calculate_discharge_coefficient(n_o, hole_diameter, num_holes, p_run_range,␣
↪run_temp_range, delta_p_run_cc, temp_liquid_density_dict):
    injection_area = (np.pi * ((hole_diameter/1000)/2)**2)
    discharge_coefficient = []
    for i in range(len(p_run_range)):
        liquid_density =␣
↪get_liquid_density_from_pressure(temp_liquid_density_dict,␣
↪pressure_temp_dict, p_run_range[i])
        discharge_coeff = n_o / (num_holes * injection_area * np.
↪sqrt(2*liquid_density*delta_p_run_cc[i]*6892.76))
        discharge_coefficient.append(discharge_coeff)

    return discharge_coefficient

def calculate_discharge_coefficient_uncertainty(discharge_coefficient,␣
↪hole_diameter, num_holes, p_run_range, delta_p_run_cc,␣
↪temp_liquid_density_dict, mass_flow_rate_std_error):
    injection_area = (np.pi * ((hole_diameter/1000)/2)**2)
    c_d_std_error = []
    for i in range(len(p_run_range)):
        liquid_density =␣
↪get_liquid_density_from_pressure(temp_liquid_density_dict,␣
↪pressure_temp_dict, p_run_range[i])
        discharge_coefficient_std_error = (1 / (num_holes * injection_area * np.
↪sqrt(2*liquid_density*abs(delta_p_run_cc[i])*6892.76))) *␣
↪mass_flow_rate_std_error
        c_d_std_error.append(discharge_coefficient_std_error)
    return c_d_std_error
```

[163]:
```
def get_nitrous_mass_unknown_ullage(pressure, m_total):

    temp_K = pressure_temp_dict.get(pressure) or␣
↪pressure_temp_dict[min(pressure_temp_dict.keys(), key = lambda key:␣
↪abs(key-pressure))]
    p, p_liq, p_gas = nitrous_equations(temp_K)
    x = (V_total-((m_total/p_gas)))/((m_total/p_liq)-(m_total/p_gas))      # x␣
↪= mass of liquid / total mass
```

```
    m_liq = x * m_total                                                        #␣
↪get mass of liquid in tank
    m_gas = (1-x) * m_total                                                    #␣
↪get mass of gas in tank
    V_liq = m_liq / p_liq                                                      #␣
↪get volume of liquid
    V_gas = m_gas / p_gas                                                      #␣
↪get volume of gas
    V_liq_height = V_liq / base_area                                           #␣
↪get height of liquid in tank
    dip_tube_length = cyl_height - V_liq_height                                #␣
↪get diptube length
    ullage_percentage = (dip_tube_length / cyl_height) * 100

    return m_liq, m_gas, ullage_percentage

# def get_nitrous_mass_known_ullage(outside_temp, ullage_percentage):

#     p, p_liq, p_gas = nitrous_equations(outside_temp + 273.15)                 #␣
↪get pressure and density values given known temperature
#     V_liq_height = cyl_height * (1-(ullage_percentage / 100))                  #␣
↪get height of liquid given fixed 10% ullage
#     V_liq = V_liq_height * base_area                                           #␣
↪get volume of liquid in tank using liquid height and cylinder area of base
#     m_liq = V_liq * p_liq                                                      #␣
↪get mass of liquid given volume and density at given temperature
#     V_gas_height = cyl_height * (ullage_percentage / 100)                      #␣
↪get height of gas (ullage)
#     V_gas = V_gas_height * base_area                                           #␣
↪get volume of gas in tank using ullage and cylinder area of base
#     m_gas = V_gas * p_gas                                                      #␣
↪get mass of gas given volume and density at given temperature
#     m_total = m_liq + m_gas                                                    #␣
↪get total amount of nitrous
#     ullage_percentage = (V_gas_height / cyl_height) * 100                      #␣
↪calculate ullage percentage

#     return m_liq, m_gas, ullage_percentage
```

[164]:

```
def nozzle_mass_flow_rate(time, pressure, A, gamma, R, T_o):
    pt = pressure * 6894.76  # Convert psi to Pascals (Pa)
    mdot = ((A * pt) / np.sqrt(T_o)) * np.sqrt(gamma / R) * (((gamma + 1) /␣
↪2)**(-((gamma + 1) / (2 * (gamma - 1)))))
    return mdot
```

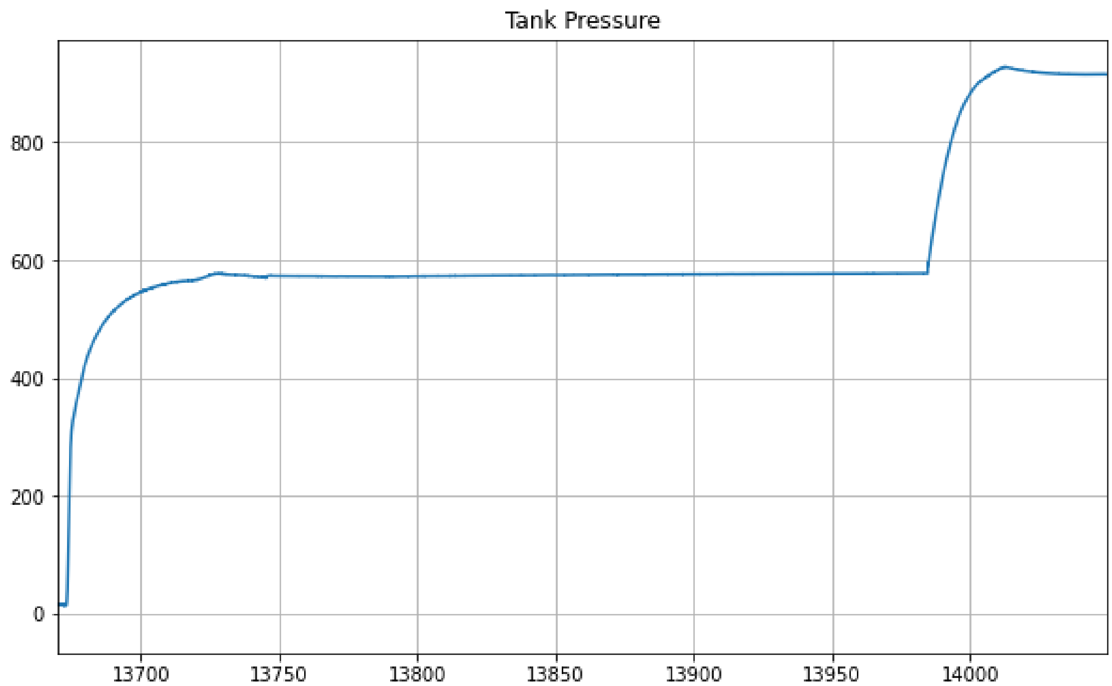
0.3 3. Determine Start and End Times

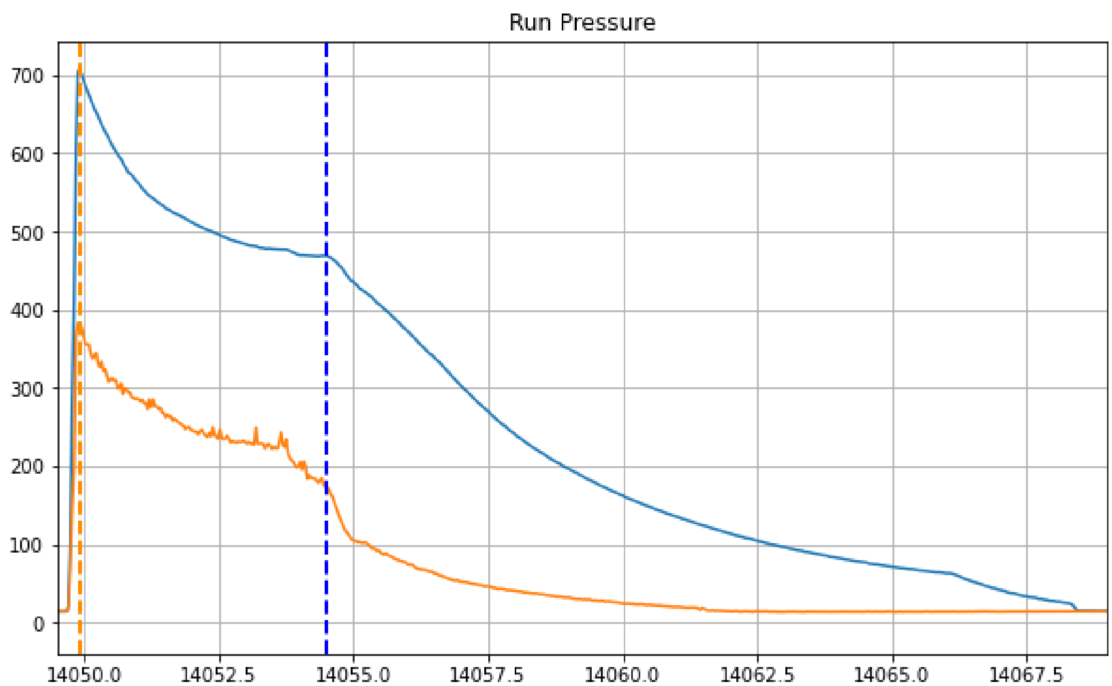
[165]:
```
column_names = ['time_ms', 'run_pressure_V', 'fill_pressure_V',␣
↪'purge_pressure_V', 'tank_pressure_V', 'tank_mass_V', 'thrust_V',␣
↪'cc_pressure_V', 'tank_temp_V', 'run_temp_V', 'vent_temp_V', 'garbage',␣
↪'run_pressure_sw', 'fill_pressure_sw', 'purge_pressure_sw',␣
↪'tank_pressure_sw', 'tank_mass_sw', 'thrust_sw', 'cc_pressure_sw',␣
↪'run_temp_sw', 'tank_temp_sw', 'ven_temp_sw']
df = pd.read_csv(input_data, sep='\t', names=column_names, index_col=False,␣
↪dtype=np.float64, skip_blank_lines=True)
time, p_fill, p_tank, p_run, p_cc, tank_temp, run_temp, tank_mass, thrust =␣
↪(df['time_ms'] / 1000, df['fill_pressure_sw'], df['tank_pressure_sw'],␣
↪df['run_pressure_sw'], df['cc_pressure_sw'], df['tank_temp_sw'],␣
↪df['run_temp_sw'], df['tank_mass_sw'], df['thrust_sw'])
```

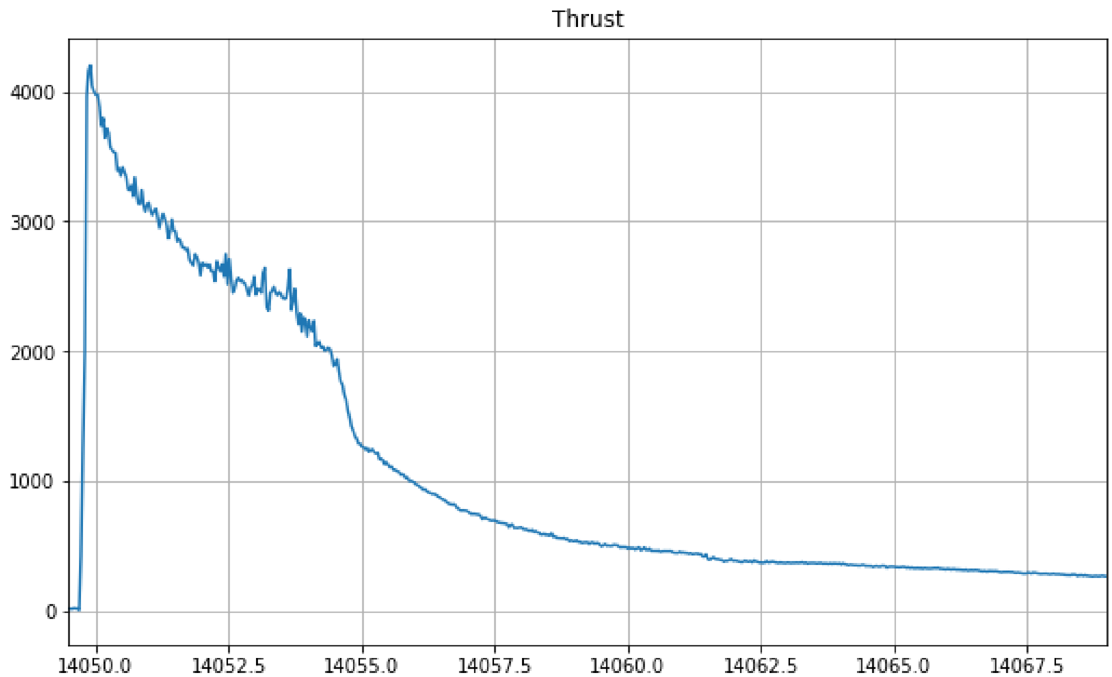
[166]:
```
# play around with these values until you find the correct times
start_time_fill = 13670
start_time = 14049.5
end_time = 14069
end_time_fill = start_time
start_time_liq_pre = 14049.9
end_time_liq_pre = 14054.5
start_time_liq = 0.4
end_time_liq = 5
```

[167]:
```
plt.figure(figsize=(10, 6))
plt.plot(time, p_tank)
plt.title('Tank Pressure')
plt.grid(True)
plt.xlim(start_time_fill, start_time)
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(time, p_run)
plt.plot(time, p_cc)
plt.axvline(x= start_time_liq_pre, color='darkorange', linestyle='--',␣
↪linewidth=2)
plt.axvline(x= end_time_liq_pre, color='blue', linestyle='--', linewidth=2)
plt.title('Run Pressure')
plt.grid(True)
plt.xlim(start_time, end_time)
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(time, thrust)
#plt.axvline(x= liq_start, color='darkorange', linestyle='--', linewidth=2)
#plt.axvline(x= liq_end, color='blue', linestyle='--', linewidth=2)
```

```
plt.title('Thrust')
plt.grid(True)
plt.xlim(start_time, end_time)
plt.show()
```







0.4 4. Extract Data

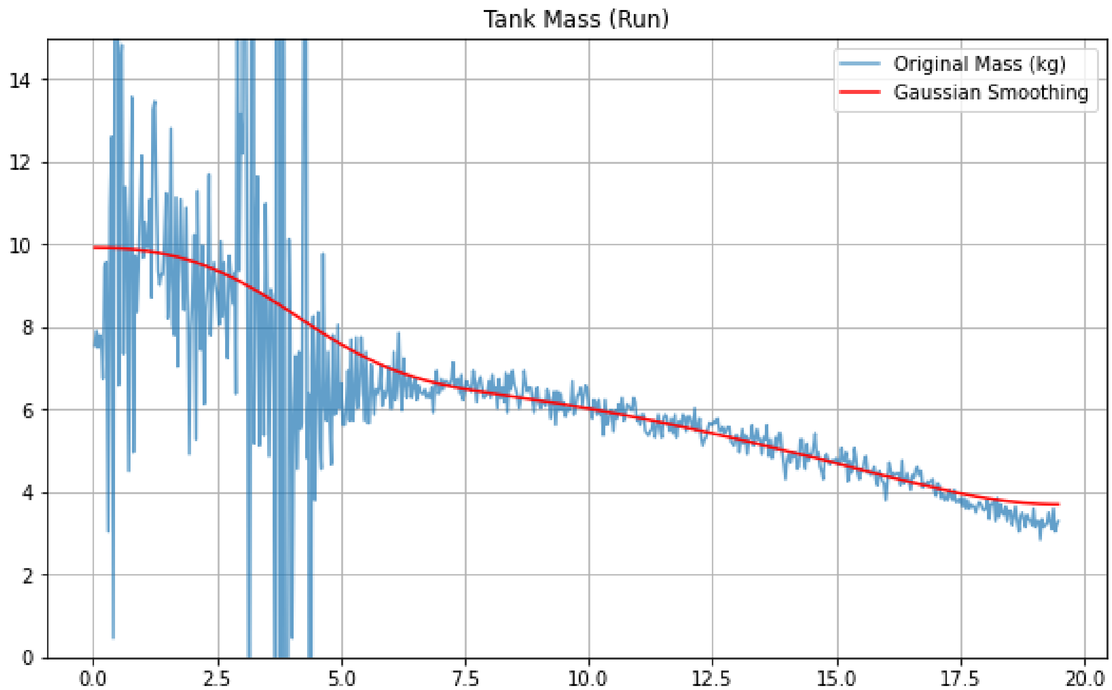
[168]:

```
# create dictionary of carbon_dioxide liquid density and temperature values (as␣
↪solving for liquid density from temp is too mathematically intensive)
temp_list = np.linspace(183, 309, num = 10000)
vap_list, liquid_list, gas_list = nitrous_equation_lists(temp_list)
temp_liquid_density_dict = dict(zip(temp_list, liquid_list))
pressure_temp_dict = dict(zip(vap_list, temp_list))

# get time ranges
time_range = [t - start_time for t in time[(time >= start_time) & (time <=␣
↪end_time)]]
time_range_fill = [t - start_time_fill for t in time if start_time_fill <= t <=␣
↪end_time_fill]
time_range_full = [t - start_time_fill for t in time[(time >= start_time_fill)␣
↪& (time <= end_time)]]

# get data ranges in time frame
p_fill_range = get_data_range(p_fill, time, start_time_fill, end_time_fill)
p_tank_fill_range = get_data_range(p_tank, time, start_time_fill, end_time_fill)
p_tank_range = get_data_range(p_tank, time, start_time, end_time)
p_run_range = get_data_range(p_run, time, start_time, end_time)
```

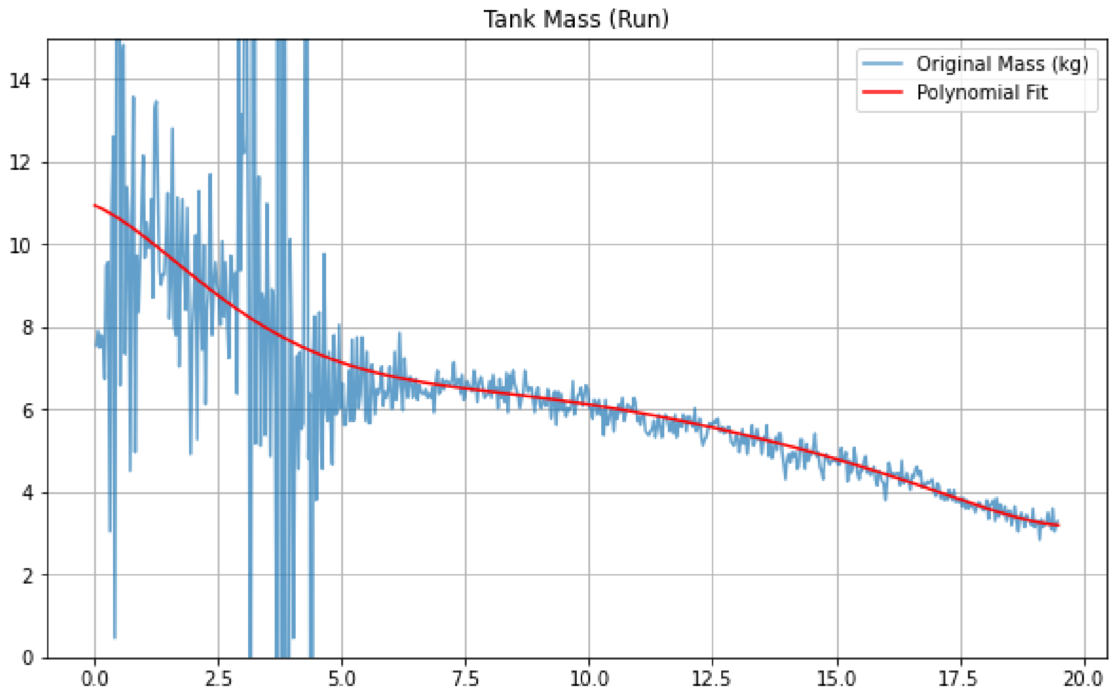
```
p_cc_range = get_data_range(p_cc, time, start_time, end_time)
tank_temp_range = get_data_range(tank_temp, time, start_time, end_time)
run_temp_range = get_data_range(run_temp, time, start_time, end_time)
thrust_range = get_data_range(thrust, time, start_time, end_time)
mass_range = get_data_range(tank_mass, time, start_time_fill, start_time)
mass_run_range = get_data_range(tank_mass, time, start_time, end_time)

# calculate pressure deltas
delta_p_tank_fill = [pt - pf for pt, pf in zip(p_tank_fill_range, p_fill_range)]
delta_p_tank_run = [pt - pr for pt, pr in zip(p_tank_range, p_run_range)]
delta_p_run_cc = [pr - pc for pr, pc in zip(p_run_range, p_cc_range)]
```

0.5 5. Mass Data Fitting and Mass Estimates

[169]:
```
# Method 1: Low Pass Filter and Gaussian Smoothing (ideal)
#smoothed_mass_savgol = savgol_filter(mass_run_range, window_length=100,␣
↪polyorder=1) # Savitzky-Golay filter (Low-Pass)
smoothed_mass_gaussian = gaussian_filter1d(mass_run_range, sigma=60) # Gaussian␣
↪Smoothing

plt.figure(figsize=(10, 6))
plt.plot(time_range, mass_run_range, label='Original Mass (kg)', alpha=0.7)
#plt.plot(time_range, smoothed_mass_savgol, label='Savitzky-Golay (Low-Pass)',␣
↪color='red')
plt.plot(time_range, smoothed_mass_gaussian, label='Gaussian Smoothing',␣
↪color='red')
plt.title('Tank Mass (Run)')
plt.ylim(0,15)
plt.grid(True)
plt.legend()
plt.show()
```

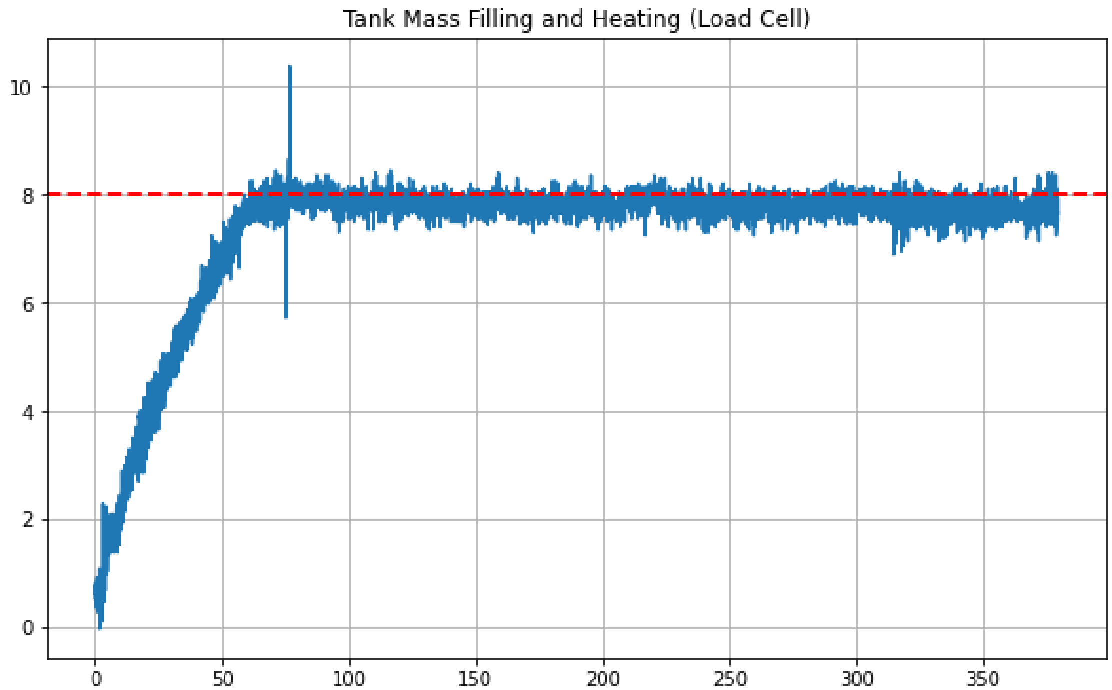


[170]:
```
# Method 2: Manual Polynomial Fit (desperate for a fit if too many vibrations)
# manually define points
coordinates = np.array([[0, 11], [1, 10], [2.5, 9], [5, 6.9], [6, 6.8], [7.5, 6.
 ↪7], [9, 6.3], [10, 6], [12.5, 5.7], [13.5, 5.1], [15, 4.9], [17.5, 3.8], [19.
 ↪5, 3.2]])
x_coordinates = coordinates[:, 0]
y_coordinates = coordinates[:, 1]
degree = 7  # You can adjust the degree based on the complexity
coefficients = np.polyfit(x_coordinates, y_coordinates, degree)
poly_fit = np.poly1d(coefficients)
fit_line = np.linspace(min(x_coordinates), max(x_coordinates), 100)

plt.figure(figsize=(10, 6))
plt.plot(time_range, mass_run_range, label='Original Mass (kg)', alpha=0.7)
plt.plot(fit_line, poly_fit(fit_line), label='Polynomial Fit', color='red')
plt.title('Tank Mass (Run)')
plt.ylim(0,15)
plt.grid(True)
plt.legend()
plt.show()
```



[171]:
```python
# If using Method 2 for mass data fit
def integrand(x):
    return derivative_poly_fit(x)

derivative_poly_fit = poly_fit.deriv()
integrated_mass, _ = quad(integrand, min(x_coordinates), max(x_coordinates))
```

[172]:
```python
# Total Mass (end of fill)
m_total_start = 8

plt.figure(figsize=(10, 6))
plt.plot(time_range_fill, mass_range, label='Mass (kg)')
plt.axhline(y= m_total_start, color='red', linestyle='--', linewidth=2)
plt.title('Tank Mass Filling and Heating (Load Cell)')
#plt.xlim(1750,2000)
#plt.ylim(7, 9)
plt.grid(True)
plt.show()
```



[173]:
```
# Mass from diptube length (ullage) estimate
# m_liq, m_gas, ullage_percentage =␣
↪get_carbon_dioxide_mass_known_ullage(outdoor_temp, 25)
# diptube_mass = m_liq + m_gas
```

[174]:
```
# mass estimates
print('Load Cell Mass (end of fill) =', m_total_start, 'kg')
print('Integrated Mass Flow Rate Fit =', abs(integrated_mass), 'kg')
```

```
Load Cell Mass (end of fill) = 8 kg
Integrated Mass Flow Rate Fit = 7.7469687629837845 kg
```

0.6 6. Calculate Important Parameters

[175]:
```
# fill time
fill_time = start_time - start_time_fill
```

[176]:
```
# peak tank pressure
peak_tank_pressure = max(p_tank_range)

# peak_tank_temperature
peak_tank_temp = max(tank_temp_range)

# peak run pressure
```

```
peak_run_pressure = max(p_run_range)

# peak combustion chamber pressure
peak_cc_pressure = max(p_cc_range)
```

[177]:
```
# ullage factor, liquid and gaseous carbon_dioxide mass
m_liq, m_gas, ullage_percentage =␣
↪get_nitrous_mass_unknown_ullage(peak_tank_pressure, m_total_start)
#m_liq, m_gas, ullage_percentage =␣
↪get_carbon_dioxide_mass_known_ullage(outdoor_temp)
```

[178]:
```
# mass flow rate
average_mass_flow_rate, _ = quad(lambda x: derivative_poly_fit(x),␣
↪start_time_liq, end_time_liq)
average_mass_flow_rate = average_mass_flow_rate / (end_time_liq -␣
↪start_time_liq)
peak_mass_flow_rate = minimize_scalar(lambda x: derivative_poly_fit(x),␣
↪bounds=(start_time_liq, end_time_liq), method='bounded').fun

# discharge coefficient
average_c_d = calculate_discharge_coefficient(-average_mass_flow_rate,␣
↪hole_diameter, num_holes, p_run_range, run_temp_range, delta_p_run_cc,␣
↪temp_liquid_density_dict)
peak_c_d = calculate_discharge_coefficient(-peak_mass_flow_rate, hole_diameter,␣
↪num_holes, p_run_range, run_temp_range, delta_p_run_cc,␣
↪temp_liquid_density_dict)
```

```
/var/folders/yn/w934gfts1gv92p3xhy03fn880000gn/T/ipykernel_43633/2675315785.py:7
: RuntimeWarning: invalid value encountered in sqrt
  discharge_coeff = n_o / (num_holes * injection_area *
np.sqrt(2*liquid_density*delta_p_run_cc[i]*6892.76))
```

[179]:
```
# burn time
burn_time = end_time - start_time

# peak thrust
peak_thrust = max(thrust_range)

# total impulse
total_impulse = calculate_total_impulse(time_range, thrust_range)
```

[180]:
```
# nozzle mass flow rate
nozzle_mass_flow_rate = [nozzle_mass_flow_rate(t, p, A, gamma, R, T_o) for t, p␣
↪in zip(time_range, p_cc_range)]

## specific impulse
```

```
# Method 1: Use Nozzle Mass Flow Rate
isp = []
for thrust, mdot in zip(thrust_range, nozzle_mass_flow_rate):
    isp_value = thrust / (mdot * 9.81)
    isp.append(isp_value)

average_isp_value = np.mean([value for t, value in zip(time_range, isp) if␣
↪start_time_liq <= t <= end_time_liq])

# Method 2: Total Impulse
total_impulse_isp = total_impulse / (((mean_mass + m_fuel) * 9.81))
```

[181]:
```
# print values
print('Ullage Factor =', ullage_percentage, '%')
print('Fill Time =', fill_time, 's')
print('Peak Tank Pressure =', peak_tank_pressure, 'psi')
print('Peak Tank Temp =', peak_tank_temp, 'C')
print('Peak Run Pressure =', peak_run_pressure, 'psi')
print('Peak CC Pressure =', peak_cc_pressure, 'psi')
print('Peak Mass Flow Rate =', - peak_mass_flow_rate, 'kg/s')
print('Average Mass Flow Rate =', - average_mass_flow_rate, 'kg/s')
print('Mass of Liquid =', m_liq, 'kg')
print('Mass of Gas =', m_gas, 'kg')
print('Burn Time =', burn_time, 's')
print('Peak Thrust =', peak_thrust, 'N')
print('Total Impulse =', total_impulse, 'Ns')
print('Specific Impulse (nozzle mass flow rate) =', average_isp_value, 's')
print('Specific Impulse (total impulse) =', total_impulse_isp, 's')
```

```
Ullage Factor = 52.47833571085181 %
Fill Time = 379.5 s
Peak Tank Pressure = 915.308473 psi
Peak Tank Temp = 36.168775 C
Peak Run Pressure = 707.907059 psi
Peak CC Pressure = 382.397356 psi
Peak Mass Flow Rate = 0.9748043909643459 kg/s
Average Mass Flow Rate = 0.7734310332390019 kg/s
Mass of Liquid = 7.808115483833367 kg
Mass of Gas = 0.19188451616663293 kg
Burn Time = 19.5 s
Peak Thrust = 4201.756879 N
Total Impulse = 20932.31468903329 Ns
Specific Impulse (nozzle mass flow rate) = 228.12442423957899 s
Specific Impulse (total impulse) = 232.3489723816815 s
```

0.7 7. Mass Error Propagation

[182]:
```
# Mass Error (standard deviation and standard error)
#mass_data = np.array([m_total_start, abs(integrated_mass), diptube_mass])
mass_data = np.array([m_total_start, abs(integrated_mass)])
mean_mass = np.mean(mass_data)
std_dev = np.std(mass_data, ddof=1)
std_error = std_dev / np.sqrt(len(mass_data))

print("Mean Mass:", mean_mass, "kg")
print("Standard Deviation:", std_dev, "kg")
print("Standard Error:", std_error, "kg")
```

```
Mean Mass: 7.873484381491892 kg
Standard Deviation: 0.17892010354618657 kg
Standard Error: 0.12651561850810777 kg
```

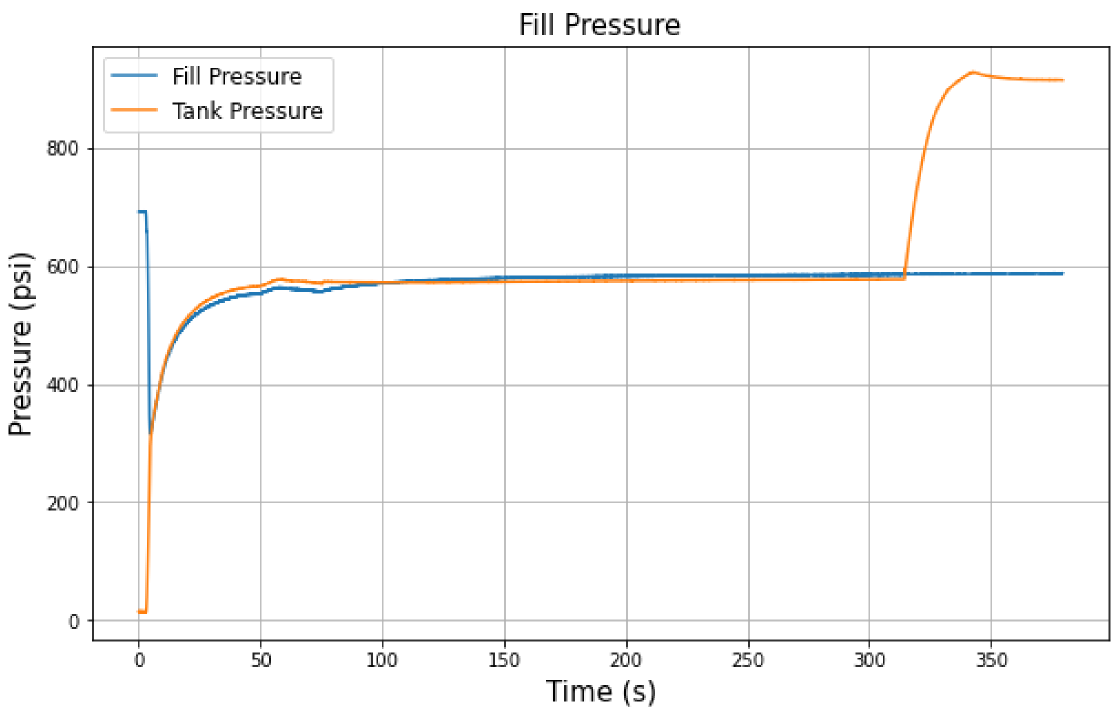
[183]:
```
# Mass Flow Rate Error
mass_flow_rate_std_error = np.abs(average_mass_flow_rate) * std_error
```

[185]:
```
# Discharge Coefficient Error
average_c_d_std_error =␣
 ↪calculate_discharge_coefficient_uncertainty(average_c_d, hole_diameter,␣
 ↪num_holes, p_run_range, delta_p_run_cc, temp_liquid_density_dict,␣
 ↪mass_flow_rate_std_error)
peak_c_d_std_error = calculate_discharge_coefficient_uncertainty(peak_c_d,␣
 ↪hole_diameter, num_holes, p_run_range, delta_p_run_cc,␣
 ↪temp_liquid_density_dict, mass_flow_rate_std_error)
```

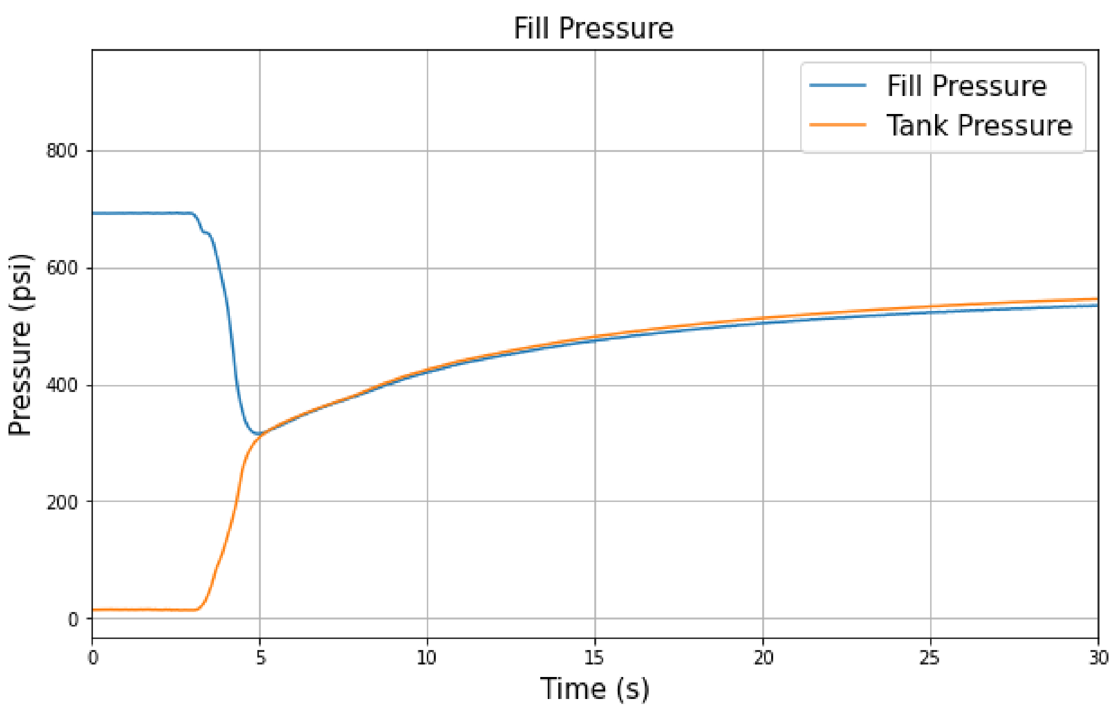
[201]:
```
# Total Impulse Error
upper_mean_mass = mean_mass + std_error
lower_mean_mass = mean_mass - std_error

upper_m_fuel = m_fuel + m_fuel_uncertainty
lower_m_fuel = m_fuel - m_fuel_uncertainty

# Calculate upper and lower bounds for total_impulse_isp
upper_total_impulse_isp = (total_impulse) / (((lower_mean_mass + upper_m_fuel)␣
 ↪* 9.8))
lower_total_impulse_isp = (total_impulse) / (((upper_mean_mass + lower_m_fuel)␣
 ↪* 9.8))
```

0.8 8. Plots

[186]:
```
# Fill Pressure
plt.figure(figsize=(10, 6))
plt.plot(time_range_fill, p_fill_range, label='Fill Pressure')
plt.plot(time_range_fill, p_tank_fill_range, label='Tank Pressure')
plt.xlabel('Time (s)', fontsize = 15)
plt.ylabel('Pressure (psi)', fontsize = 15)
plt.title('Fill Pressure', fontsize = 15)
plt.grid(True)
plt.legend(fontsize = 12)
plt.gca().set_facecolor('white')
plt.savefig('fill_pressure.png', facecolor='white')
plt.show()

# Fill Pressure Zoomed
plt.figure(figsize=(10, 6))
plt.plot(time_range_fill, p_fill_range, label='Fill Pressure')
plt.plot(time_range_fill, p_tank_fill_range, label='Tank Pressure')
plt.xlabel('Time (s)', fontsize = 15)
plt.ylabel('Pressure (psi)', fontsize = 15)
plt.title('Fill Pressure', fontsize = 15)
plt.grid(True)
plt.legend(fontsize = 15)
plt.xlim(0,30)
#plt.ylim(0,500)
plt.gca().set_facecolor('white')
plt.savefig('fill_pressure_zoomed.png', facecolor='white')
plt.show()
```

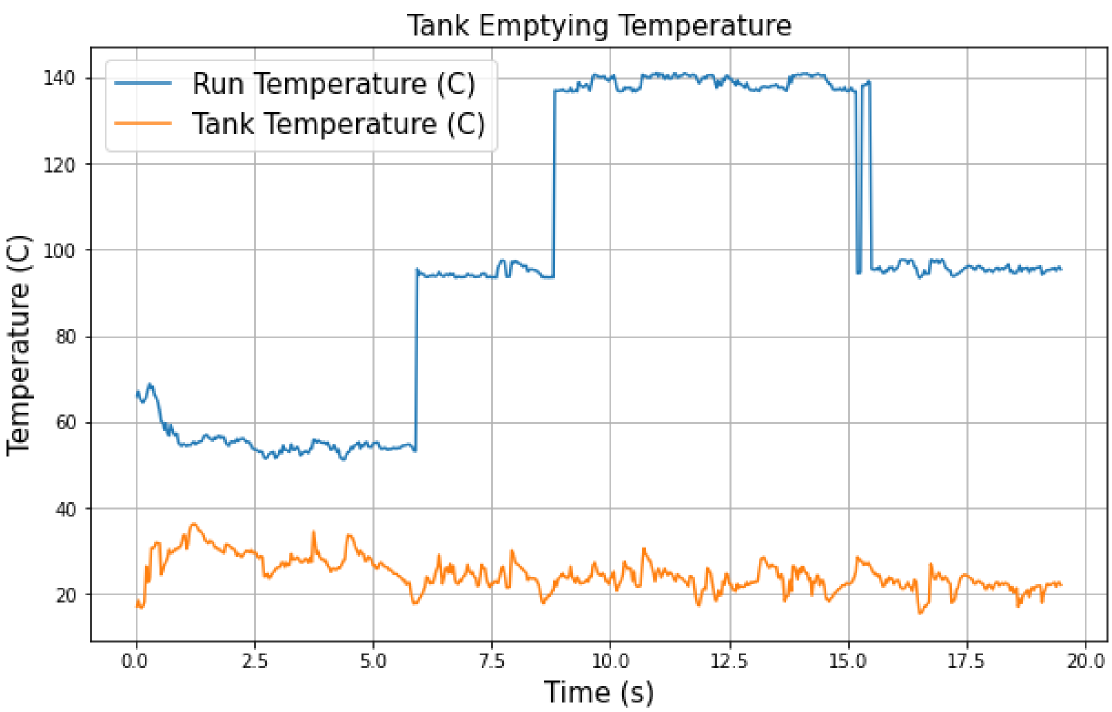


Fill Pressure
Fill Pressure
Tank Pressure
Pressure (psi)
0
200
400
600
800
0
50
100
150
200
250
300
350
Time (s)



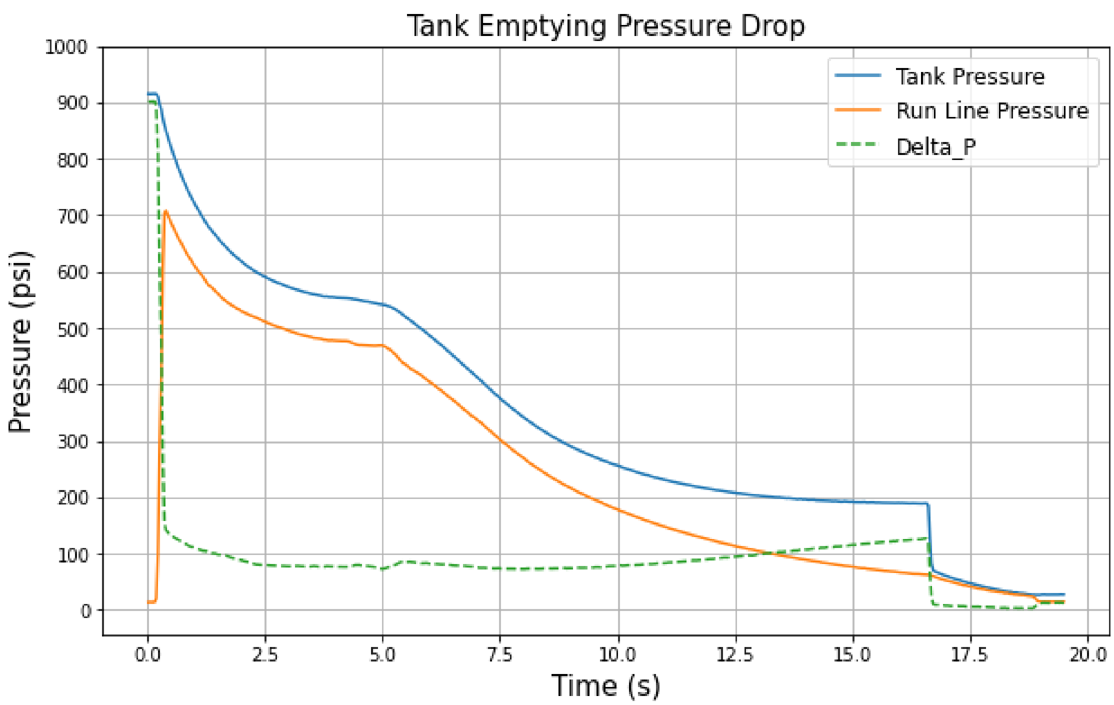
Fill Pressure
Fill Pressure
Tank Pressure
Pressure (psi)
0
200
400
600
800
0
5
10
15
20
25
30
Time (s)

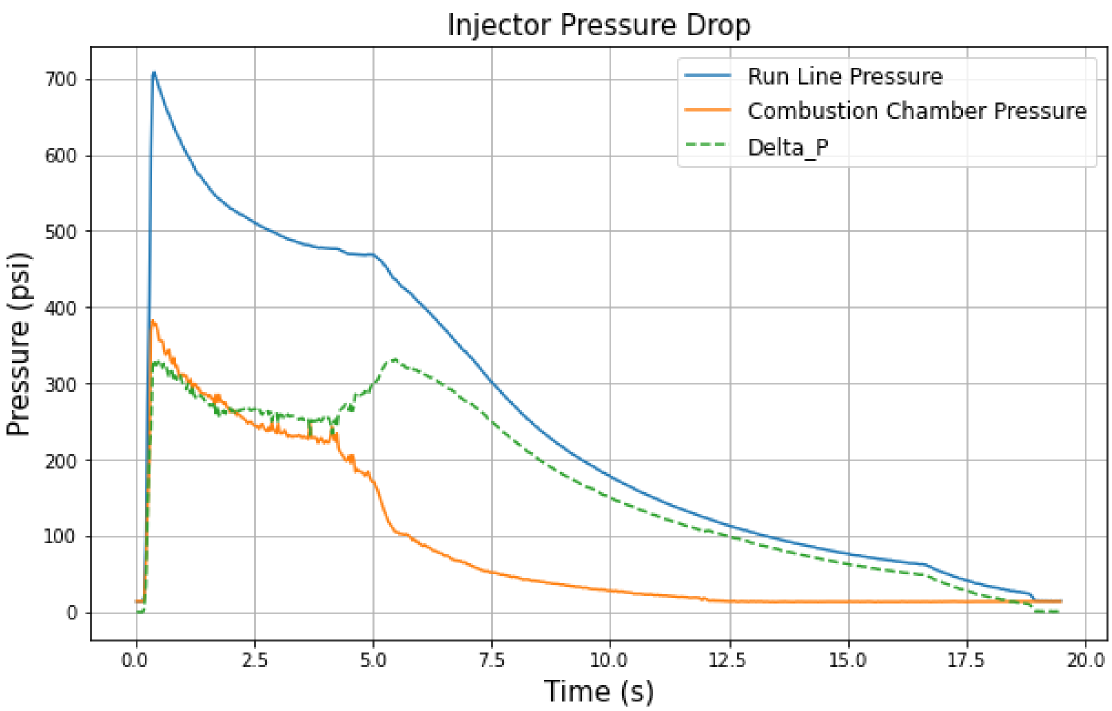
[187]:
```python
# Temperature
plt.figure(figsize=(10, 6))
plt.plot(time_range, run_temp_range, label='Run Temperature (C)')
plt.plot(time_range, tank_temp_range, label='Tank Temperature (C)')
plt.xlabel('Time (s)', fontsize = 15)
plt.ylabel('Temperature (C)', fontsize = 15)
plt.title('Tank Emptying Temperature', fontsize = 15)
plt.grid(True)
plt.legend(fontsize = 15)
plt.gca().set_facecolor('white')
plt.savefig('tank_emptying_temp.png', facecolor='white')
plt.show()
```



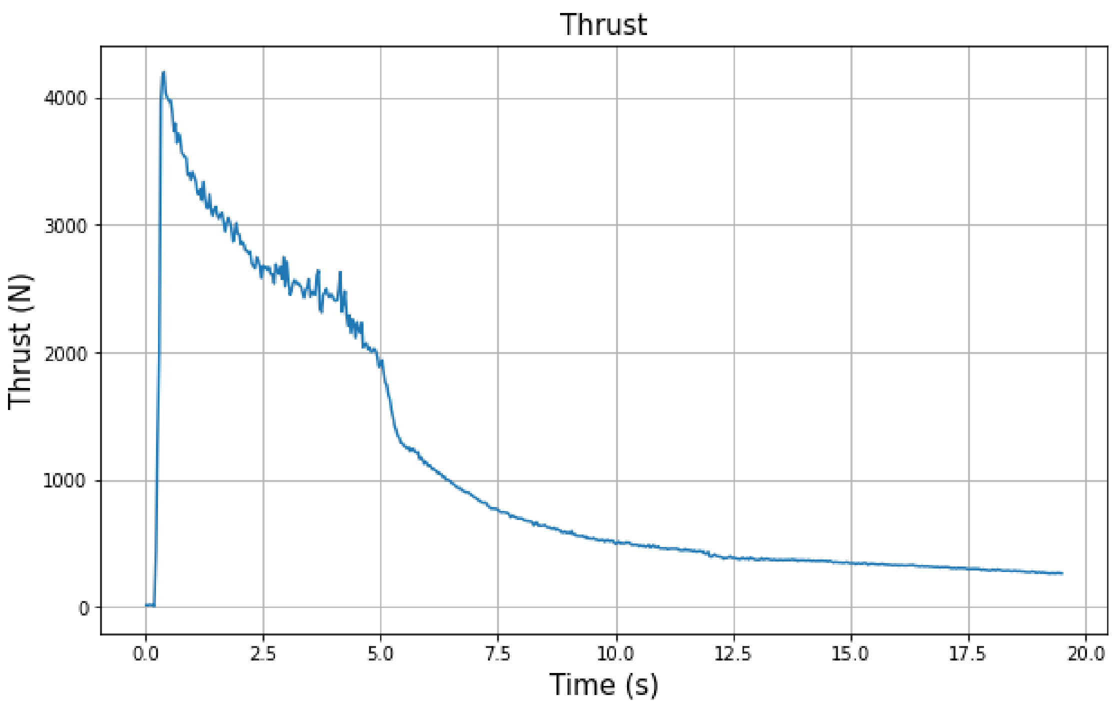
[188]:
```python
# Tank/Run Pressure
plt.figure(figsize=(10, 6))
plt.plot(time_range, p_tank_range, label='Tank Pressure')
plt.plot(time_range, p_run_range, label='Run Line Pressure')
plt.plot(time_range, delta_p_tank_run, linestyle='--', label='Delta_P')
plt.xlabel('Time (s)', fontsize = 15)
plt.ylabel('Pressure (psi)', fontsize = 15)
plt.title('Tank Emptying Pressure Drop', fontsize = 15)
plt.grid(True)
plt.legend(fontsize = 12)
plt.gca().set_facecolor('white')
```

```python
plt.yticks(np.arange(0, max(p_tank_range) + 100, 100))
plt.savefig('tank_emptying_pressure.png', facecolor='white')
plt.show()

# Injector Pressure Drop
plt.figure(figsize=(10, 6))
plt.plot(time_range, p_run_range, label='Run Line Pressure')
plt.plot(time_range, p_cc_range, label='Combustion Chamber Pressure')
plt.plot(time_range, delta_p_run_cc, linestyle='--', label='Delta_P')
plt.xlabel('Time (s)', fontsize = 15)
plt.ylabel('Pressure (psi)', fontsize = 15)
plt.title('Injector Pressure Drop', fontsize = 15)
plt.grid(True)
plt.legend(fontsize = 12)
plt.gca().set_facecolor('white')
plt.savefig('injector_pressure_drop.png', facecolor='white')
plt.show()
```





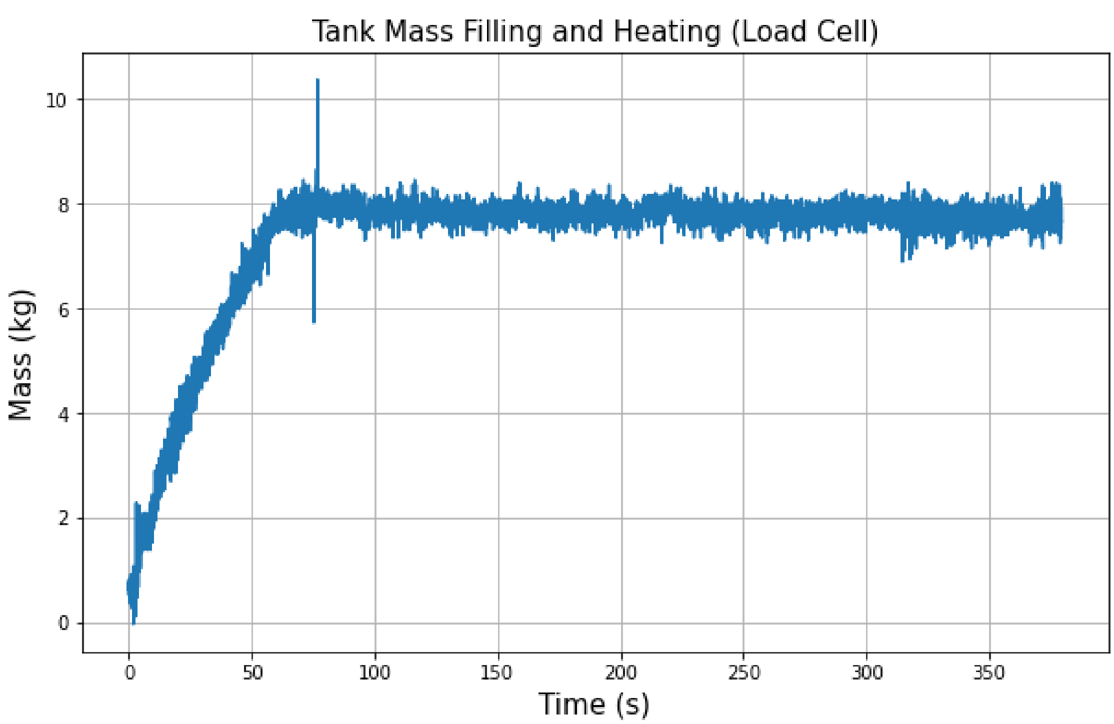
[189]:
```
# Thrust
plt.figure(figsize=(10, 6))
plt.plot(time_range, thrust_range, label='Thrust (N)')
plt.xlabel('Time (s)', fontsize = 15)
plt.ylabel('Thrust (N)', fontsize = 15)
plt.title('Thrust', fontsize = 15)
plt.grid(True)
plt.gca().set_facecolor('white')
plt.savefig('thrust.png', facecolor='white')
plt.show()
```

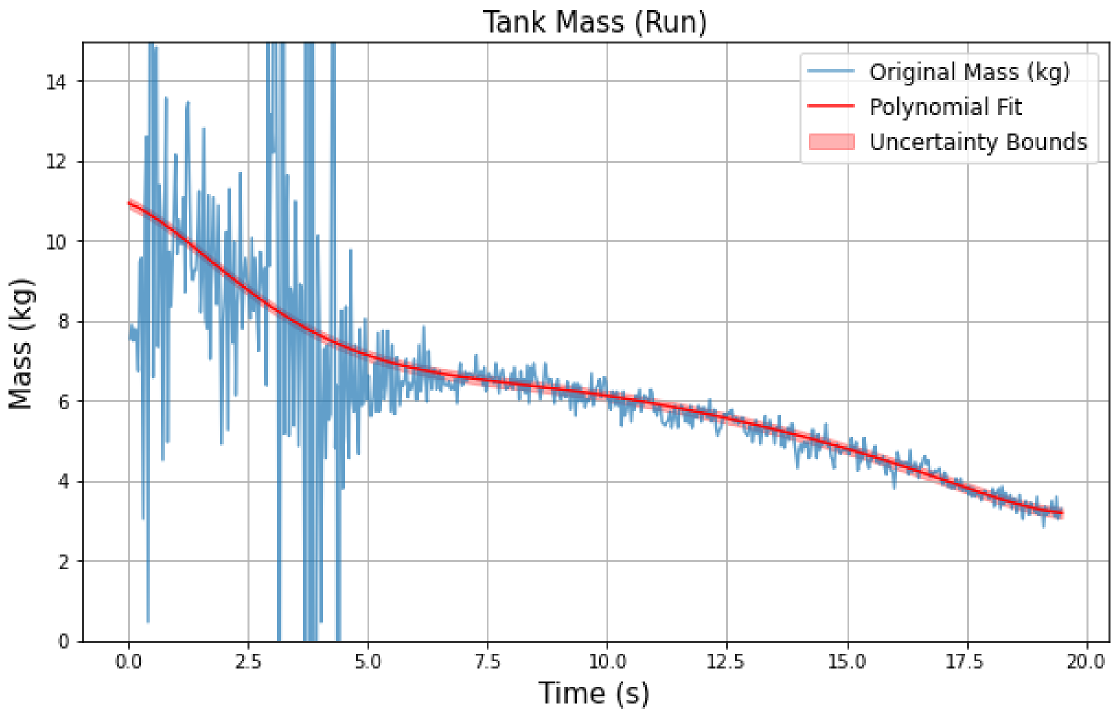


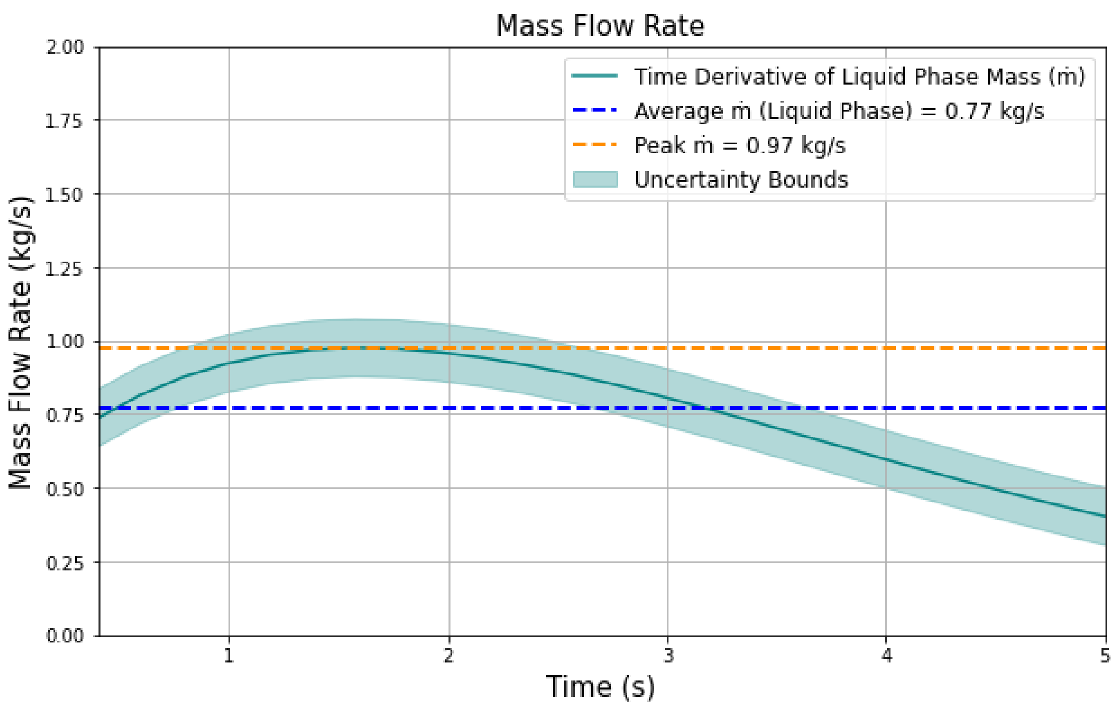
[200]:
```
# Mass
plt.figure(figsize=(10, 6))
plt.plot(time_range_fill, mass_range, label='Mass (kg)')
plt.xlabel('Time (s)', fontsize = 15)
plt.ylabel('Mass (kg)', fontsize = 15)
plt.title('Tank Mass Filling and Heating (Load Cell)', fontsize = 15)
plt.grid(True)
plt.gca().set_facecolor('white')
plt.savefig('mass_fill.png', facecolor='white')
plt.show()


# 2.9 Mass (Run)
plt.figure(figsize=(10, 6))
plt.plot(time_range, mass_run_range, label='Original Mass (kg)', alpha=0.7)
plt.plot(fit_line, poly_fit(fit_line), label='Polynomial Fit', color='red')
upper_bound = poly_fit(fit_line) + std_error
lower_bound = poly_fit(fit_line) - std_error
plt.fill_between(fit_line, upper_bound, lower_bound, color='red', alpha=0.3,␣
  ↪label='Uncertainty Bounds')
plt.title('Tank Mass (Run)', fontsize = 15)
plt.xlabel('Time (s)', fontsize = 15)
plt.ylabel('Mass (kg)', fontsize = 15)
plt.grid(True)
```

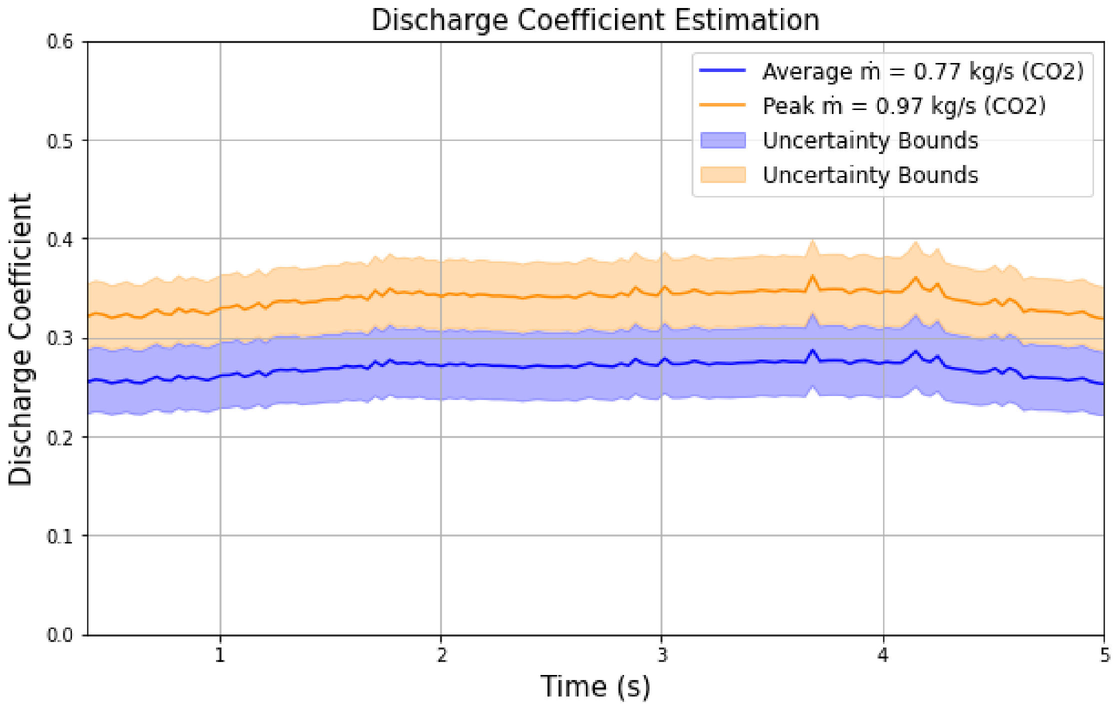
```python
plt.ylim(0,15)
plt.gca().set_facecolor('white')
plt.legend(fontsize = 12)
plt.savefig('mass_tank_emptying.png', facecolor='white')
plt.show()

# Mass Flow Rate
derivative_values = -derivative_poly_fit(fit_line)

# Calculate the upper and lower bounds using the standard error
upper_bound = derivative_values + mass_flow_rate_std_error
lower_bound = derivative_values - mass_flow_rate_std_error

# Create the plot
plt.figure(figsize=(10, 6))
plt.plot(fit_line, derivative_values, label='Time Derivative of Liquid Phase␣
↪Mass (ṁ)', color='teal')
plt.axhline(y= - average_mass_flow_rate, color='blue', linestyle='--',␣
↪label="Average ṁ (Liquid Phase) = 0.77 kg/s", linewidth=2)
plt.axhline(y= - peak_mass_flow_rate, color='darkorange', linestyle='--',␣
↪label="Peak ṁ = 0.97 kg/s", linewidth=2)
plt.fill_between(fit_line, upper_bound, lower_bound, color='teal', alpha=0.3,␣
↪label='Uncertainty Bounds')
plt.title('Mass Flow Rate', fontsize=15)
plt.xlabel('Time (s)', fontsize=15)
plt.ylabel('Mass Flow Rate (kg/s)', fontsize=15)
plt.grid(True)
plt.legend(fontsize=12)
plt.xlim(start_time_liq, end_time_liq)
plt.ylim(0, 2)
plt.gca().set_facecolor('white')
plt.savefig('injector_mass_flow_rate.png', facecolor='white')
plt.show()

average_c_d = np.array(average_c_d)
average_c_d_std_error = np.array(average_c_d_std_error)
peak_c_d = np.array(peak_c_d)
peak_c_d_std_error = np.array(peak_c_d_std_error)

# Discharge Coefficient Estimation
plt.figure(figsize=(10, 6))
plt.plot(time_range, average_c_d, label='Average ṁ = 0.77 kg/s (CO2)', color =␣
↪'blue')
plt.plot(time_range, peak_c_d, label='Peak ṁ = 0.97 kg/s (CO2)', color =␣
↪'darkorange')
upper_bound_avg = average_c_d + average_c_d_std_error
```

```
lower_bound_avg = average_c_d - average_c_d_std_error
upper_bound_peak = peak_c_d + peak_c_d_std_error
lower_bound_peak = peak_c_d - peak_c_d_std_error
plt.fill_between(time_range, upper_bound_avg, lower_bound_avg, color='blue',␣
 ↪alpha=0.3, label='Uncertainty Bounds')
plt.fill_between(time_range, upper_bound_peak, lower_bound_peak,␣
 ↪color='darkorange', alpha=0.3, label='Uncertainty Bounds')
plt.title('Discharge Coefficient Estimation', fontsize = 15)
plt.xlabel('Time (s)', fontsize = 15)
plt.ylabel('Discharge Coefficient', fontsize = 15)
plt.grid(True)
plt.legend(fontsize = 12)
plt.xlim(start_time_liq, end_time_liq)
plt.ylim(0,0.6)
plt.gca().set_facecolor('white')
plt.savefig('discharge_coefficient.png', facecolor='white')
plt.show()
```



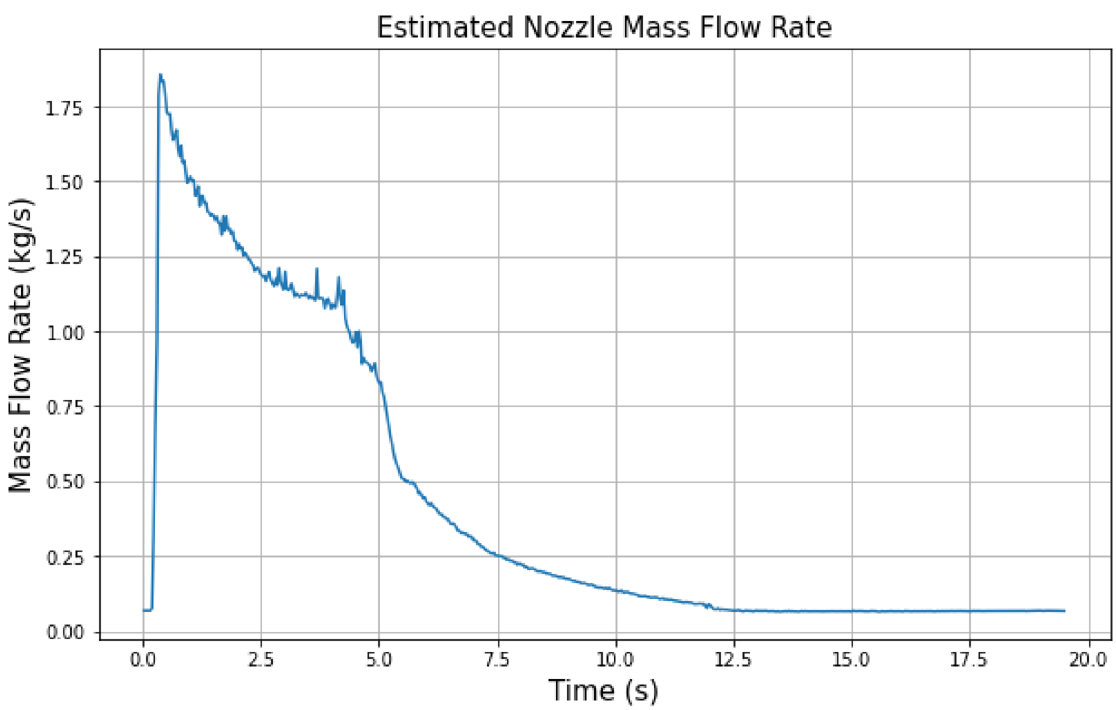

Tank Mass (Run)
Original Mass (kg)
Polynomial Fit
Uncertainty Bounds
Mass (kg)
Time (s)
0
2
4
6
8
10
12
14
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

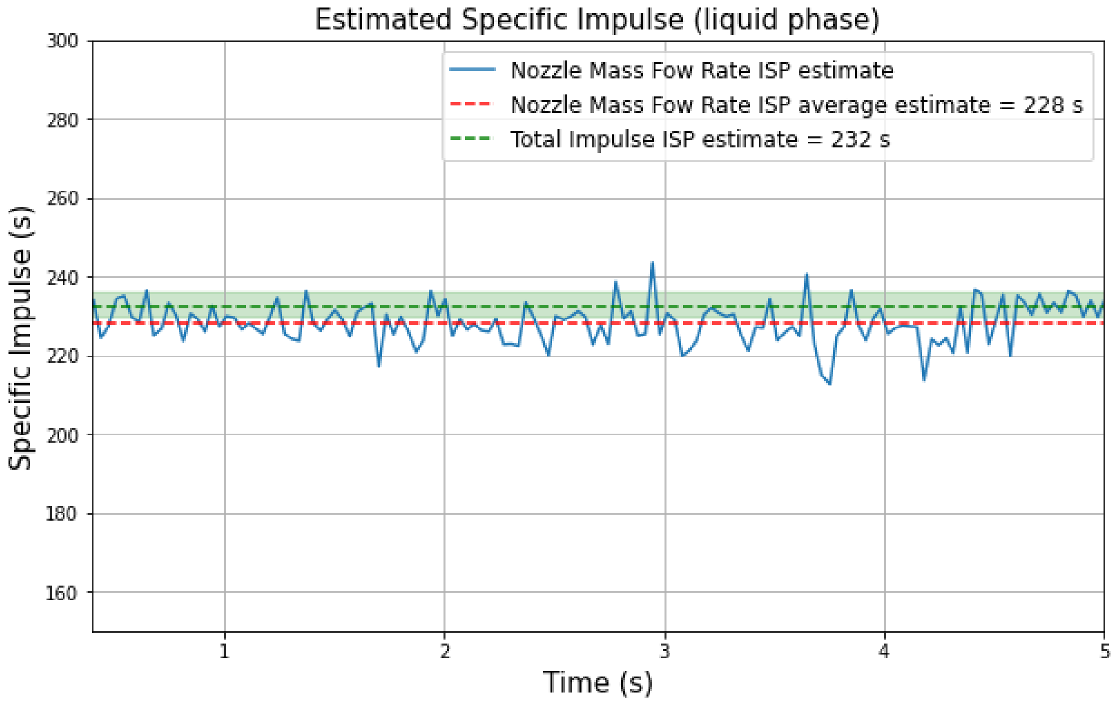

Mass Flow Rate
Time Derivative of Liquid Phase Mass (ṁ)
Average ṁ (Liquid Phase) = 0.77 kg/s
Peak ṁ = 0.97 kg/s
Uncertainty Bounds
Mass Flow Rate (kg/s)
Time (s)
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1
2
3
4
5



[202]:
```
# Nozzle Mass Flow Rate
plt.figure(figsize=(10, 6))
plt.plot(time_range, nozzle_mass_flow_rate)
plt.xlabel("Time (s)", fontsize = 15)
plt.ylabel("Mass Flow Rate (kg/s)", fontsize = 15)
plt.title("Estimated Nozzle Mass Flow Rate", fontsize = 15)
plt.grid(True)
plt.gca().set_facecolor('white')
plt.savefig('nozzle_mass_flow_rate.png', facecolor='white')
plt.show()

#upper_mean_mass = mean_mass + std_error
#lower_mean_mass = mean_mass - std_error

#upper_m_fuel = m_fuel + m_fuel_uncertainty
#lower_m_fuel = m_fuel - m_fuel_uncertainty

# Calculate upper and lower bounds for total_impulse_isp
#upper_total_impulse_isp = (total_impulse) / (((lower_mean_mass + upper_m_fuel)␣
↪* 9.8))
#lower_total_impulse_isp = (total_impulse) / (((upper_mean_mass + lower_m_fuel)␣
↪* 9.8))
#print(upper_total_impulse_isp)
#print(lower_total_impulse_isp)
```

```python
# Specific Impulse (Estimated)
plt.figure(figsize=(10, 6))
plt.plot(time_range, isp, label="Nozzle Mass Fow Rate ISP estimate ")
plt.axhline(y=average_isp_value, color='r', linestyle='--', label="Nozzle Mass␣
 ↪Fow Rate ISP average estimate = 228 s")
plt.axhline(y=total_impulse_isp, color='g', linestyle='--', label="Total␣
 ↪Impulse ISP estimate = 232 s")
plt.fill_between(time_range, upper_total_impulse_isp, lower_total_impulse_isp,␣
 ↪color='g', alpha=0.2)
plt.xlabel("Time (s)", fontsize = 15)
plt.ylabel("Specific Impulse (s)", fontsize = 15)
plt.title("Estimated Specific Impulse (liquid phase)", fontsize = 15)
plt.grid(True)
plt.legend(fontsize = 12)
plt.xlim(start_time_liq, end_time_liq)
plt.ylim(150,300)
plt.gca().set_facecolor('white')
plt.savefig('isp.png', facecolor='white')
plt.show()
```





1 Additional Analysis

[69]:
```
# Determine liquid density from temperature. Finds key from closest value in␣
↪dictionary using list comprehension
def get_density_from_pressure(temp_liquid_density_dict, temp_gas_density_dict,␣
↪pressure_temp_dict, pressure_key):
    pressure_KPa = pressure_key * 6.89476
    temp_K = pressure_temp_dict.get(pressure_KPa) or␣
↪pressure_temp_dict[min(pressure_temp_dict.keys(), key = lambda key:␣
↪abs(key-pressure_KPa))]
    liquid_density = temp_liquid_density_dict.get(temp_K) or␣
↪temp_liquid_density_dict[min(temp_liquid_density_dict.keys(), key = lambda␣
↪key: abs(key-temp_K))]
    gas_density = temp_gas_density_dict.get(temp_K) or␣
↪temp_gas_density_dict[min(temp_gas_density_dict.keys(), key = lambda key:␣
↪abs(key-temp_K))]

    return liquid_density, gas_density
```

[70]:
```
temp_list = np.linspace(183, 309, num = 10000)
vap_list, liquid_list, gas_list = nitrous_equation_lists(temp_list)
temp_liquid_density_dict = dict(zip(temp_list, liquid_list))
temp_gas_density_dict = dict(zip(temp_list, gas_list))
```

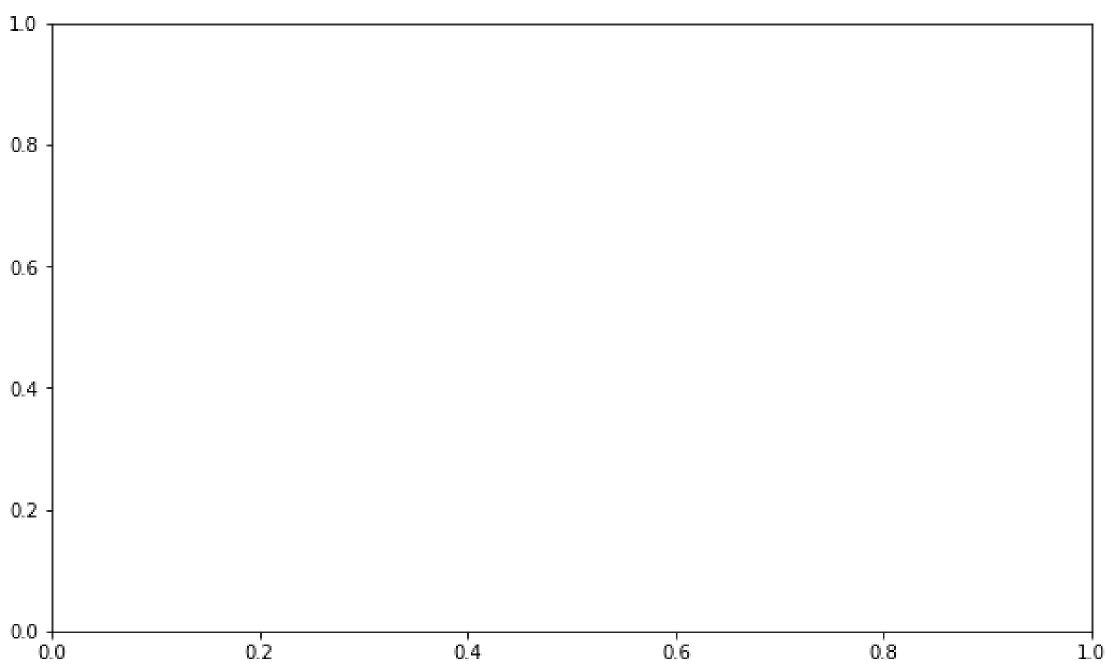
```
pressure_temp_dict = dict(zip(vap_list, temp_list))
```

[71]:
```
liquid_density_list = []
gas_density_list = []
for i in range(len(p_run_range)):
        liquid_density, gas_density = ␣
 ↪get_density_from_pressure(temp_liquid_density_dict, temp_gas_density_dict,␣
 ↪pressure_temp_dict, p_run_range[i])
        liquid_density_list.append(liquid_density)
        gas_density_list.append(gas_density)
```

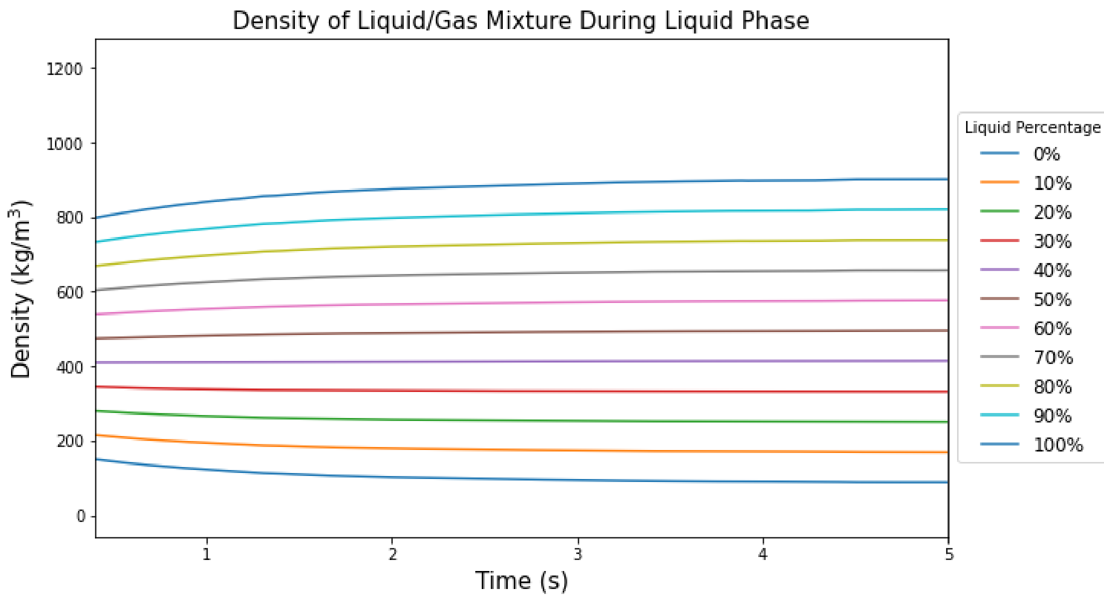
[76]:
```
# 4.2 Density of Saturated Liquid
plt.figure(figsize=(10, 6))
plt.plot(time_range, liquid_density_list)
plt.plot(time_range, gas_density_list)
plt.xlabel('Time (s)')
plt.ylabel('Density (kg/m^3)')
plt.title('Density of Saturated Liquid and Saturated Gas During Liquid Phase')
plt.xlim(start_time_liq, end_time_liq)
plt.savefig('density_1.png', facecolor='white')
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [76], in <cell line: 3>()
      1 # 4.2 Density of Saturated Liquid
      2 plt.figure(figsize=(10, 6))
----> 3 plt.plot(time_range, liquid_density_list)
      4 plt.plot(time_range, gas_density_list)
      5 plt.xlabel('Time (s)')

File /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/
 ↪site-packages/matplotlib/pyplot.py:2769, in plot(scalex, scaley, data, *args,␣
 ↪**kwargs)
   2767 @_copy_docstring_and_deprecators(Axes.plot)
   2768 def plot(*args, scalex=True, scaley=True, data=None, **kwargs):
-> 2769     return gca().plot(
   2770         *args, scalex=scalex, scaley=scaley,
   2771         **({"data": data} if data is not None else {}), **kwargs)

File /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/
 ↪site-packages/matplotlib/axes/_axes.py:1632, in Axes.plot(self, scalex,␣
 ↪scaley, data, *args, **kwargs)
   1390 """
   1391 Plot y versus x as lines and/or markers.
   1392
   (…)
```

```
   1629 (``'green'``) or hex strings (``'#008000'``).
   1630 """
   1631 kwargs = cbook.normalize_kwargs(kwargs, mlines.Line2D)
-> 1632 lines = [*self._get_lines(*args, data=data, **kwargs)]
   1633 for line in lines:
   1634     self.add_line(line)

File /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/
 ↪site-packages/matplotlib/axes/_base.py:312, in _process_plot_var_args.
 ↪__call__(self, data, *args, **kwargs)
    310     this += args[0],
    311     args = args[1:]
--> 312 yield from self._plot_args(this, kwargs)

File /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/
 ↪site-packages/matplotlib/axes/_base.py:498, in _process_plot_var_args.
 ↪_plot_args(self, tup, kwargs, return_kwargs)
    495     self.axes.yaxis.update_units(y)
    497 if x.shape[0] != y.shape[0]:
--> 498     raise ValueError(f"x and y must have same first dimension, but "
    499                      f"have shapes {x.shape} and {y.shape}")
    500 if x.ndim > 2 or y.ndim > 2:
    501     raise ValueError(f"x and y can be no greater than 2D, but have "
    502                      f"shapes {x.shape} and {y.shape}")

ValueError: x and y must have same first dimension, but have shapes (583,) and␣
 ↪(6413,)
```



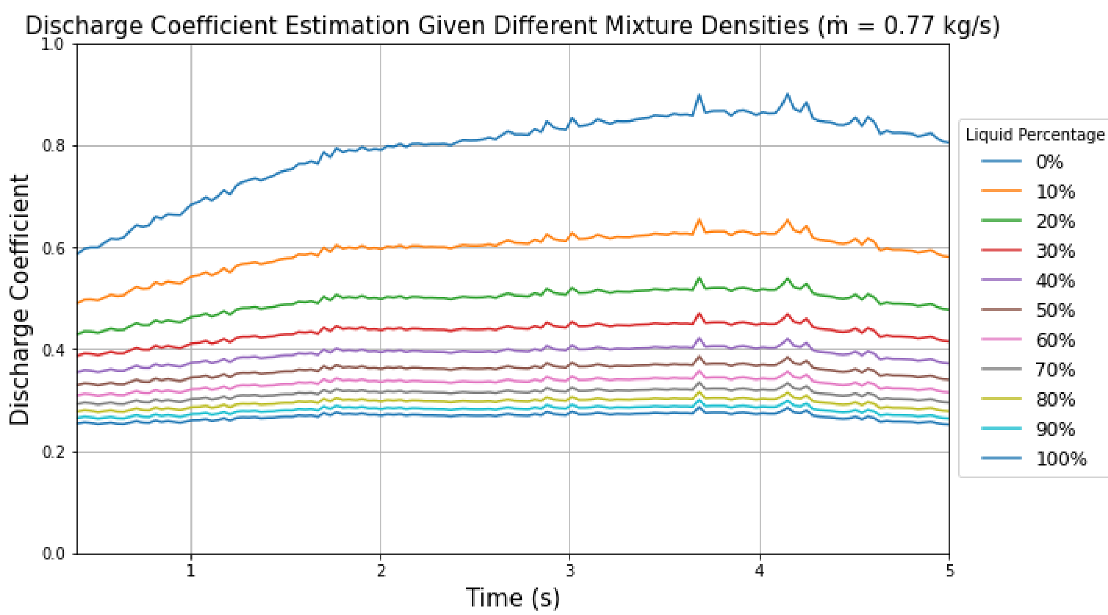
[77]:
```
liquid_density_list = []
gas_density_list = []

liquid_percentages = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]

plt.figure(figsize=(10, 6))

for liquid_percent in liquid_percentages:
    weighted_density_list = []  # List to store weighted average densities for␣
 ↪current liquid percentage

    for i in range(len(p_run_range)):
        liquid_density, gas_density =␣
 ↪get_density_from_pressure(temp_liquid_density_dict, temp_gas_density_dict,␣
 ↪pressure_temp_dict, p_run_range[i])
        liquid_density_list.append(liquid_density)
        gas_density_list.append(gas_density)

        # Calculate weighted average density for the current mixture ratio
        weighted_density = (liquid_density * liquid_percent + gas_density * (1␣
 ↪- liquid_percent))
        weighted_density_list.append(weighted_density)

    # Plot weighted density for current liquid percentage
    plt.plot(time_range, weighted_density_list,␣
 ↪label=f'{int(liquid_percent*100)}%')

plt.xlabel('Time (s)', fontsize=15)
plt.ylabel('Density (kg/m$^3$)', fontsize=15)
plt.title('Density of Liquid/Gas Mixture During Liquid Phase', fontsize=15)
plt.xlim(start_time_liq, end_time_liq)
plt.legend(title="Liquid Percentage", fontsize=12, loc='center left',␣
 ↪bbox_to_anchor=(1, 0.5))
plt.savefig('density_2.png', facecolor='white')
plt.show()
```



[78]:
```
def calculate_discharge_coefficient(liquid_percent, n_o, hole_diameter,␣
↪num_holes, p_run_range, delta_p_run_cc, temp_liquid_density_dict,␣
↪temp_gas_density_dict, pressure_temp_dict):
    injection_area = (np.pi * ((hole_diameter/1000)/2)**2)
    discharge_coefficient = []

    for i in range(len(p_run_range)):
        liquid_density, gas_density =␣
↪get_density_from_pressure(temp_liquid_density_dict, temp_gas_density_dict,␣
↪pressure_temp_dict, p_run_range[i])
        weighted_density = (liquid_density * liquid_percent + gas_density * (1␣
↪- liquid_percent))
        discharge_coeff = n_o / (num_holes * injection_area * np.sqrt(2 *␣
↪weighted_density * delta_p_run_cc[i] * 6892.76))
        discharge_coefficient.append(discharge_coeff)

    return discharge_coefficient

plt.figure(figsize=(10, 6))

for liquid_percent in liquid_percentages:
    discharge_coefficients = calculate_discharge_coefficient(liquid_percent, 0.
↪77, hole_diameter, num_holes, p_run_range, delta_p_run_cc,␣
↪temp_liquid_density_dict, temp_gas_density_dict, pressure_temp_dict)
    plt.plot(time_range, discharge_coefficients,␣
↪label=f'{int(liquid_percent*100)}%')
```

```python
plt.title('Discharge Coefficient Estimation Given Different Mixture Densities␣
  ↪(ṁ = 0.77 kg/s)', fontsize=15)
plt.xlabel('Time (s)', fontsize=15)
plt.ylabel('Discharge Coefficient', fontsize=15)
plt.grid(True)
plt.legend(title = "Liquid Percentage", fontsize=12, loc='center left',␣
  ↪bbox_to_anchor=(1, 0.5))
plt.ylim(0, 1)
plt.xlim(start_time_liq, end_time_liq)
plt.gca().set_facecolor('white')
plt.savefig('density_3.png', facecolor='white')
plt.show()
```

```
/var/folders/yn/w934gfts1gv92p3xhy03fn880000gn/T/ipykernel_43633/828682819.py:8:
RuntimeWarning: invalid value encountered in sqrt
  discharge_coeff = n_o / (num_holes * injection_area * np.sqrt(2 *
weighted_density * delta_p_run_cc[i] * 6892.76))
```




```
[ ]:
```