w-test-site-data-analysis-template

August 28, 2024

[5]:
```python
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.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

Define the constants specific to this coldflow test

[ ]:
```python
# 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
```

[ ]:
```python
# injector
hole_diameter = 2                                                  # mm
num_holes = ???
```

[ ]:
```python
# input filename
input_data = '???.txt'
```

[ ]:
```python
# ambient temp
outdoor_temp = ???                                                  # Celcius
```

0.2 2. Define Functions

These are the functions required to run the code. Add any additional functions in this section.

[1]:
```
# 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]
```

[3]:
```
# determine co2 vapour pressure, liquid density and gas density given␣
↪temperature
def carbon_dioxide_equations(temp_kelvin):
    # https://mychemengmusings.wordpress.com/2022/06/23/
↪five-short-equations-describing-the-saturated-density-and-enthalpy-of-carbon-dioxide-co2-va
↪
    ZsatT = 1 + (0.001613*(temp_kelvin*(304.128-temp_kelvin))**(0.6))-0.67508

    # Density of Saturated Liquid
    p_l =  ((-3.53267*(temp_kelvin-216.592))/(ZsatT**(0.646)))+1180.409

    # Density of Saturated Gas
    p_g = 467.6 * math.exp(-75.135 * (304.128-temp_kelvin)**0.68 /␣
↪temp_kelvin**1.15 / ZsatT**0.33  - 0.1855 )

    # Vapour Pressure
    p = ((p_g * ZsatT * temp_kelvin)/529.304)*100

    return p, p_l, p_g

# turn CO2 vapour pressure, liquid density and gas density into lists
def carbon_dioxide_equation_lists(temp_list):
    vap_list = []
    liquid_list = []
    gas_list = []
    for temp_kelvin in temp_list:
        p, p_l, p_g = carbon_dioxide_equations(temp_kelvin)
        vap_list.append(p)
        liquid_list.append(p_l)
        gas_list.append(p_g)

    return vap_list, liquid_list, gas_list
```

[4]:
```
# 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
```

[ ]:
```
# 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*delta_p_run_cc[i]*6892.76))) *␣
↪mass_flow_rate_std_error[i]
        c_d_std_error.append(discharge_coefficient_std_error)
    return c_d_std_error
```

[ ]:
```
def get_carbon_dioxide_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 = carbon_dioxide_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_carbon_dioxide_mass_known_ullage(outside_temp, ullage_percentage):

#     p, p_liq, p_gas = carbon_dioxide_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 carbon dioxide
#     ullage_percentage = (V_gas_height / cyl_height) * 100                 #␣
 ↪calculate ullage percentage

#     return m_liq, m_gas, ullage_percentage
```

0.3 3. Determine Start and End Times

This section is used to determine the correct start and end times for filling, burn and liquid phase. I had scripts that would do this automatically, but it was too unreliable with the noisy data. While it might take a bit of time to tweak, hardcoding it like this is much more accurate. It can be very quick after you've done it for the first time.

```
[ ]: 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'])
```

```
[ ]: # play around with these values until you find the correct times. Keep printing
     ↪out the plots below until you get what you want.
     start_time_fill = ???
     start_time = ???
     end_time = ???
     end_time_fill = start_time
     start_time_liq_pre = ???
     end_time_liq_pre = ???
     start_time_liq = ???
     end_time_liq = ???
```

```
[ ]: 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

This section actually extracts the data from the txt file and gets the correct data ranges and pressure deltas.

[ ]:
```
# 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 = carbon_dioxide_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

This section fits the mass data and gives mass estimates

[ ]:
```python
# Low Pass Filter and Gaussian Smoothing (ideal)
smoothed_mass_gaussian = gaussian_filter1d(mass_run_range, sigma=4) # 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_gaussian, label='Gaussian Smoothing', color='red')
plt.title('Tank Mass (Run)')
plt.grid(True)
plt.legend()
plt.show()
```

[ ]:
```python
# Mass Loss Estimate
#Note: This is hardcoded for now.
print(time_range[???])
print(time_range[???])

mass_loss_estimate = smoothed_mass_gaussian[???] - smoothed_mass_gaussian[???]
```

[ ]:
```python
# Total Mass (end of fill)
m_total_start = 8.5

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(8, 9)
plt.grid(True)
plt.show()
```

[ ]:
```python
# 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
```

[ ]:
```python
# mass estimates
print('Load Cell Mass (end of fill) =', m_total_start, 'kg')
print('Mass Loss Estimate =', mass_loss_estimate, 'kg')
# print('Filled to diptube =', diptube_mass, 'kg')
```

0.6 6. Calculate Important Parameters

This is where the important parameters are calculated. Copy over what is printed here to the LaTex reports.

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

```
[ ]: # 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)
```

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

```
[ ]: # mass flow rate
time_derivative = np.gradient(smoothed_mass_gaussian, time_range)
peak_mass_flow_rate = min(time_derivative)
average_mass_flow_rate = np.mean([value for t, value in zip(time_range,␣
 ↪time_derivative) if start_time_liq <= t <= end_time_liq])


# 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)
```

```
[ ]: # 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)
```

[ ]:
```
# 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')
```

0.7 7. Mass Error Propogation

This section propogates the error in the mass data to the mass flow rate and discharge coefficient

[ ]:
```
# 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, mass_loss_estimate])
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")
```

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

[ ]:
```
# 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)
```

0.8 8. Plots

Plot the results

[ ]:
```
# 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()
```

[ ]:
```
# 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()
```

[ ]:
```
# 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')
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()
```

[ ]:
```
# 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()
```

[ ]:
```
# 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(time_range, smoothed_mass_gaussian, label='Gaussian Smoothing',␣
↪color='red')
```

```
upper_bound = smoothed_mass_gaussian + std_error
lower_bound = smoothed_mass_gaussian - std_error
plt.fill_between(time_range, lower_bound, upper_bound, color='red', alpha=0.3)
plt.title('Tank Mass (Run)', fontsize = 15)
plt.xlabel('Time (s)', fontsize = 15)
plt.ylabel('Mass (kg)', fontsize = 15)
plt.grid(True)
plt.gca().set_facecolor('white')
plt.legend(fontsize = 12)
plt.savefig('mass_tank_emptying.png', facecolor='white')
plt.show()

# Mass Flow Rate
plt.figure(figsize=(10, 6))
plt.plot(time_range, - time_derivative, 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) = 0.96 kg/s", linewidth=2)
plt.axhline(y= - peak_mass_flow_rate, color='darkorange', linestyle='--',␣
↪label="Peak ṁ = 1.9 kg/s", linewidth=2)
upper_bound = time_derivative + mass_flow_rate_std_error
lower_bound = time_derivative - mass_flow_rate_std_error
plt.fill_between(time_range, -upper_bound, -lower_bound, color='teal', alpha=0.
↪5, 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(-3,5)
plt.gca().set_facecolor('white')
plt.savefig('injector_mass_flow_rate.png', facecolor='white')
plt.show()

# Discharge Coefficient Estimation
plt.figure(figsize=(10, 6))
plt.plot(time_range, average_c_d, label='Average ṁ = 0.96 kg/s (CO2)', color =␣
↪'blue')
plt.plot(time_range, peak_c_d, label='Peak ṁ = 1.9 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
#plt.fill_between(time_range, -upper_bound_avg, -lower_bound_avg, color='teal',␣
↪alpha=0.5, 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(0,0.8)
plt.gca().set_facecolor('white')
plt.savefig('discharge_coefficient.png', facecolor='white')
plt.show()
```