Appendix A Problem 1 Python Code

```
#!/usr/bin/env python

from cProfile import label
from webbrowser import get
import isentropic as isen
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np

# Problem 1

# Read .csv data file in as pandas dataframe, remove nan rows, convert to
    np array
trajectory = pd.read_csv('../HW_3_problem_1_data.csv',skiprows=[1])
trajectory = trajectory.dropna()
trajectory = trajectory.to_numpy()

traj_dict = {'time' : 0, 'h' : 1, 'u' : 2, 'T' : 3, 'p' : 4, 'rho' : 5}

# Part A
mach_isen = [isen.get_mach_number(u=u,T=T) for u, T in zip (trajectory[:,
    traj_dict['u']],trajectory[:,traj_dict['T']])]
mach_roomtemp = [isen.get_mach_number(u=u,T=298,p=p) for u,p in zip(
    trajectory[:,traj_dict['u']],trajectory[:,traj_dict['p']])]
mach_diff = [abs(Mi-Mr) for Mi,Mr in zip(mach_isen,mach_roomtemp)]
percent_diff = [diff/mi*100 for diff,mi in zip(mach_diff,mach_isen)]

fig, ax = plt.subplots()
plt.plot(trajectory[:,traj_dict['time']],mach_isen, label = 'Isentropic
    Mach')
ax.set_xlabel('Time [s]')
ax.set_ylabel('Mach')
ax.set_title('Mach vs. Time')
ax.legend()
plt.savefig('../images/problem_1/Mach_correct_vs_Time.png')
plt.close()

fig, ax = plt.subplots()
plt.plot(trajectory[:,traj_dict['time']],mach_roomtemp, label = 'Room Temp
     Mach')
ax.set_xlabel('Time [s]')
ax.set_ylabel('Mach')
ax.set_title('Mach vs. Time')
ax.legend()
plt.savefig('../images/problem_1/Mach_298_vs_Time.png')
plt.close()

```

```python
fig, ax = plt.subplots()
plt.plot(trajectory[:,traj_dict['time']],mach_isen, label = 'Isentropic Mach')
plt.plot(trajectory[:,traj_dict['time']],mach_roomtemp, label = 'Room Temp Mach')
plt.plot(trajectory[:,traj_dict['time']],mach_diff, label =  'Absolute Difference')
ax.set_xlabel('Time [s]')
ax.set_ylabel('Mach')
ax.set_title('Mach vs. Time')
ax.legend()
plt.savefig('../images/problem_1/Mach_vs_Time.png')
plt.close()

fig, ax = plt.subplots()
plt.plot(trajectory[:,traj_dict['time']],percent_diff)
ax.set_xlabel('Time [s]')
ax.set_ylabel('Mach Error [%]')
ax.set_title('Mach Error vs. Time')
plt.savefig('../images/problem_1/Mach_Error_vs_Time.png')
plt.close()

# Part B
# Find the Mach number where rho_inf = rho_0 -> I know it's M = 0
mach_rho_equal = isen.get_mach_number(rho_t_ratio=1)

# Calculate time history of rho_inf/rho_t
# isentropic function returns rho_t/rho_inf, take inverse
rho_rho_t = [1/isen.get_density_ratio(M=M) for M in mach_isen]

# Defining a flow as compressible when density changes > 5%
# Need time in flight when Mach number when rho_inf/rho_t = 0.95
mach_rho_095 = isen.get_mach_number(rho_t_ratio=1/0.95)

closest_mach_095 = min(mach_isen, key=lambda x:abs(x-mach_rho_095))
print(f'Mach at closest time = {closest_mach_095}')
idx_095 = mach_isen.index(closest_mach_095)

closest_time_095 = trajectory[idx_095,traj_dict['time']]
print(f'Closest Time = {closest_time_095}')

closest_rho_ratio_095 = rho_rho_t[idx_095]
print(f'rho_inf/rho_t at closest time = {closest_rho_ratio_095}')


fig,ax = plt.subplots()
plt.plot(trajectory[:,traj_dict['time']],rho_rho_t)
ax.set_xlabel('Time [s]')
ax.set_ylabel(r'$\frac{\rho_\infty}{\rho_0}$', fontsize=18)
ax.set_title(r'$\frac{\rho_\infty}{\rho_0}$ vs Time')
```

```
plt.savefig('../images/problem_1/rho_rho_t_vs_Time.png')
plt.close()

fig,ax = plt.subplots()
plt.plot(mach_isen,rho_rho_t)
ax.set_xlabel('Mach')
ax.set_ylabel(r'$\frac{\rho_\infty}{\rho_0}$', fontsize=18)
ax.set_title(r'$\frac{\rho_\infty}{\rho_0}$ vs Mach')
plt.savefig('../images/problem_1/rho_rho_t_vs_Mach.png')
plt.close()

fig,ax1 = plt.subplots()
ax2 = ax1.twinx()
ax1.plot(trajectory[:,traj_dict['time']],rho_rho_t,label='Density Ratio')
ax2.plot(trajectory[:,traj_dict['time']],mach_isen,'r',label='Freestream
    Mach')
ax1.set_xlabel('Time [s]')
ax1.set_ylabel(r'$\frac{\rho_\infty}{\rho_0}$', fontsize=18)
ax2.set_ylabel('Mach')
ax1.set_title(r'$\frac{\rho_\infty}{\rho_0}$ vs Time')
fig.legend()
plt.savefig('../images/problem_1/rho_rho_t_and_Mach_vs_Time.png')
#plt.close()
ax1.plot(closest_time_095,closest_rho_ratio_095,'kd')
ax2.plot(closest_time_095,closest_mach_095,'bo')
ann1 = ax1.annotate(f'rho_inf/rho_t = {np.round(closest_rho_ratio_095,2)}\
    nt = {np.round(closest_time_095)}',(closest_time_095+5,
    closest_rho_ratio_095))
ann2 = ax2.annotate(f'M = {np.round(closest_mach_095,2)}\nt = {np.round(
    closest_time_095)}',(closest_time_095+5,0))
plt.savefig('../images/problem_1/rho_rho_t_and_Mach_vs_Time_marked.png')
#plt.close()
ann1.remove()
ann2.remove()
ann1 = ax1.annotate(f'rho_inf/rho_t = {np.round(closest_rho_ratio_095,2)}\
    nt = {np.round(closest_time_095)}',(closest_time_095+1,
    closest_rho_ratio_095))
ann2 = ax2.annotate(f'M = {np.round(closest_mach_095,2)}\nt = {np.round(
    closest_time_095)}',(closest_time_095+1,0))
ax1.set_xlim(left=0,right=10)
ax2.set_xlim(left=0,right=10)
plt.savefig('../images/problem_1/rho_rho_t_and_Mach_vs_Time_marked_zoom.
    png')
plt.close()

p_t_isen = [isen.get_total_pressure(M=Mi,p=trajectory[i,traj_dict['p']])
    /1000 for i,Mi in enumerate(mach_isen)]
p_t_bernoulli = [(p + 0.5*rho*u**2)/1000 for p,rho,u in zip(trajectory[:,
    traj_dict['p']],trajectory[:,traj_dict['rho']],trajectory[:,traj_dict['
    u']])]
```

```

fig,ax = plt.subplots()
plt.plot(trajectory[:,traj_dict['time']],p_t_isen,label='Isentropic Total
    Pressure')
plt.plot(trajectory[:,traj_dict['time']],p_t_bernoulli,label='Bernoulli
    Total Pressure')
ax.set_xlabel('Time [s]')
ax.set_ylabel(r'$p_0$ [kPa]')
ax.set_title('Total Pressure vs. Time')
ax.legend()
plt.savefig('../images/problem_1/Pt_vs_Time_Isen_Bernoulli.png')
plt.close()

#  Part C
mach_target = 6

close_machs = [Mi for Mi in mach_isen if Mi > 0.99*mach_target and Mi <
    1.01*mach_target]
print(f'Number of points within +- 1% of target mach = {len(close_machs)}'
    )
print(f'Machs within +- 1% of target mach = {close_machs}')

closest_mach_M6 = min(mach_isen, key=lambda x:abs(x-mach_target))
print(f'Closest Mach to Mach 6 = {closest_mach_M6}')
idx_M6 = mach_isen.index(closest_mach_M6)
closest_time_M6 = trajectory[idx_M6,traj_dict['time']]
print(f'Closest time to Mach 6 = {closest_time_M6}')
closest_u_M6 = trajectory[idx_M6,traj_dict['u']]
print(f'Closest freestream velocity to Mach 6 = {closest_u_M6}')
closest_T_M6 = trajectory[idx_M6,traj_dict['T']]
print(f'Closest freestream temp to Mach 6 = {closest_T_M6}')
T_t_M6 = isen.get_total_temperature(M=closest_mach_M6,T=closest_T_M6)
T_t_T_M6 = isen.get_temperature_ratio(M=closest_mach_M6)

# If this were achievable, the static temperature in the nozzle throat:
T_sonic_blanket = isen.get_static_temperature(M=1,T_t=T_t_M6)
print(f'Sonic throat temp = {T_sonic_blanket}')
```

Appendix B Problem 2 Python Code

```python
#!/usr/bin/env python

import isentropic as isen
import shocks
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np
from scipy.optimize import fsolve

# Problem 2

# Read .csv data file in as pandas dataframe, remove nan rows, convert to
    np array
trajectory = pd.read_csv('../HW_3_problem_2_data.csv')
trajectory = trajectory.dropna()
trajectory = trajectory.to_numpy()

# Break into specific columns
time = trajectory[:,0]
p_t_probe = trajectory[:,1]
p1 = trajectory[:,2]

# Part a
# p1 =  static pressure upstream of shock
# p_t_probe = total pressure at stagnation point = p1_t for subsonic =
    p2_t for supersonic

# For subsonic cases:
# p1_t/p1 = (1 + (gamma-1)/2*M1**2)**((gamma-1)/gamma)
# This is valid until the pressure ratio reaches the sonic ratio limit at
    M = 1

# p1_t/p* = (1 + (gamma-1)/2)**((gamma-1)/gamma) = ~1.89

# Any ratio of p_t_probe/p1 < 1.89 = subsonic
# Any ratio of p_t_probe/p1 = 1.89 = exactly sonic
# Any ratio of p_t_probe/p1 > 1.89 = subsonic --> Mach # calculated here
    NOT valid

# For supersonic cases:

# p2t/p1 =p2t/p2*p2/p1

# p2t/p2 = (1 + (gamma-1) / 2 * M2**2)**(gamma/(gamma-1))
# p2_p1 = (1 + 2*gamma/(gamma+1)*(M1**2-1))
# M2**2 = ((1 + (gamma-1)/2 * M1**2) / (gamma * M1**2 - (gamma-1)/2))
# p2t/p1 = (1 + (gamma-1) / 2 * ((1 + (gamma-1)/2 * M1**2) / (gamma * M1
```

```
**2 - (gamma-1)/2)))**(gamma/(gamma-1)) * (1 + 2*gamma/(gamma+1)*(M1**2-1))
# Must be iteratively solved for M1

# p2t/p1 (M1=1) = 1.89

ratios = [p_t/p for p_t,p in zip(p_t_probe,p1)]
sonic_ratios = isen.get_sonic_ratios()
p_t_p_star = 1/sonic_ratios[0]

def func(M1,p2_t_p1,gamma=1.4,):
    eq = (1 + (gamma-1) / 2 * ((1 + (gamma-1)/2 * M1**2) / (gamma * M1**2 - (gamma-1)/2)))**(gamma/(gamma-1)) * (1 + 2*gamma/(gamma+1)*(M1**2-1)) - p2_t_p1
    return eq

machs = [isen.get_mach_number(p_t_ratio=r) if r < p_t_p_star else float(fsolve(func,4, args=(r))) for r in ratios]
machs_simple = [isen.get_mach_number(p_t_ratio=r) for r in ratios] # If the total pressure values were valid for in front of the shock the entire time

fig,ax = plt.subplots()
plt.plot(time,machs,label='piecewise')
ax.set_xlabel('Time [s]')
ax.set_ylabel('Mach')
ax.set_title('F-16 Mach vs. Time')
plt.savefig('../images/problem_2/Mach_vs_Time_F16.png')
plt.plot(time,machs_simple,'r',label='shock-free')
ax.legend()
plt.savefig('../images/problem_2/Mach_vs_Time_F16_Piecewise_Simple.png')
plt.close()

fig,ax = plt.subplots()
plt.plot(time,ratios,label='Probe Total/Static Pressure')
plt.plot([time[0],time[-1]],[p_t_p_star,p_t_p_star],'r',label='Supersonic Line')
ax.set_xlabel('Time [s]')
ax.set_ylabel(r'$\frac{p_0}{p}$', fontsize=18)
ax.set_title('Probe Total/Static Pressure vs. Time')
ax.legend()
plt.savefig('../images/problem_2/Probe_P_ratio_vs_Time_F16.png')
plt.close()

# Part
# Mach # experienced by probe = subsonic entire time
# Piecewise built up from aircraft Mach # when subsonic and post-normal shock Mach # when aircraft is supersonic

mach_near_probe = [M1 if M1 < 1 else shocks.get_mach_normal_shock(M1) for
```

```
    M1 in machs] # this is actually the mach number immediately behind the
    shock but not the probe because probe is stagnant

# Probe will experience same static temp as temp behind shock

fig,ax = plt.subplots()
plt.plot(time,mach_near_probe,label='In Front of Probe')
ax.set_xlabel('Time [s]')
ax.set_ylabel('Mach')
ax.set_title('Region 2 Mach vs. Time')
plt.savefig('../images/problem_2/Mach_vs_Time_F16_Probe.png')
plt.plot(time,machs,'r',label='Aircraft')
ax.set_title('Region 1/2 Mach vs. Time')
ax.legend()
plt.savefig('../images/problem_2/Mach_vs_Time_F16_Probe_Aircraft.png')
plt.close()

# T = 298 at all altitudes, static temp
T_ts = [isen.get_total_temperature(T=298,M=Mi) for Mi in machs] # Total
    temperature in front of shock
T_ts_probe = T_ts # Total temperature does not change across a shock

T2_shock = [shocks.get_static_temperature_normal_shock(M1=Mi,T1=298) if Mi
     > 1 else 298 for Mi in machs] # Also the probe static temp based off
    of temp after normal shock

fig,ax = plt.subplots()
plt.plot(time,T2_shock)
ax.set_xlabel('Time [s]')
ax.set_ylabel('Static Temperature [K]')
ax.set_title('Region 2 Static Temperature vs. Time')
plt.annotate('Note: Static temp is\nconstant (T=298 K)\nuntil shock forms'
    ,(0,425))
plt.savefig('../images/problem_2/T2_vs_Time_F16.png')
plt.close()

fig,ax = plt.subplots()
plt.plot(time,T_ts_probe)
ax.set_xlabel('Time [s]')
ax.set_ylabel('Probe Stagnation Temperature [K]')
ax.set_title('Probe Stagnation Temperature vs. Time')
plt.savefig('../images/problem_2/Probe_T_t_vs_Time_F16.png')
plt.close()
```