Appendix A Problem 1 Python Code

```python
# Compressible Flow
# AEE 553
# Homework 4 - Problem 2
# Evan Burke

from traceback import print_tb
import numpy as np
from matplotlib import pyplot as plt
from matplotlib.ticker import MaxNLocator
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
import shocks as ns
import oblique as os
import isentropic as isen
from scipy.optimize import fsolve

gamma = 1.4
delta = 1 # weak shock solution

def find_beta(M=None,gamma=1.4,theta=None,delta=1):
    theta = np.deg2rad(theta)
    lamb = ((M**2-1)**2 - 3*(1 + (gamma-1)/2*M**2) * (1 + (gamma+1)/2*M**2) * np.tan(theta)**2)**0.5
    chi = ((M**2-1)**3 - 9 * (1 + (gamma-1)/2 * M**2) * (1 + (gamma-1)/2 * M**2 + (gamma+1)/4*M**4)*np.tan(theta)**2)/lamb**3
    tan_beta = (M**2 - 1 + 2*lamb*np.cos((4*np.pi*delta+np.arccos(chi))/3)) / (3 * (1 + (gamma-1)/2*M**2)*np.tan(theta))
    beta = np.arctan(tan_beta)
    beta = np.rad2deg(beta)
    print(f'Shock angle = {beta}')
    return beta

def find_theta(M=None,beta=None,gamma=1.4):
    beta = np.deg2rad(beta)
    tanth = 2 / np.tan(beta) * (M**2 * np.sin(beta)**2 - 1) / (M**2 * (gamma + np.cos(2*beta)) + 2)
    theta = np.arctan(tanth)
    theta = np.rad2deg(theta)
    print(theta)
    return theta

t = find_theta(M=2,beta= 53.4229405)

betas = np.linspace(0.5,60,num=120,endpoint=True)
machs = np.linspace(1.1,15,num=120,endpoint=True)
print(betas)
print(machs)
```

```

data = []

for M in machs:
    for beta in betas:
        M1n = os.get_m1_normal(M1=M,beta=beta)
        M2n = os.get_m2_normal(M1n=M1n)
        p2_p1 = ns.get_static_pressure_ratio_normal_shock(M1=M1n)
        t = find_theta(M=M,beta=beta)
        data.append((M,beta,p2_p1,t))

print(data)

Xs = [point[0] for point in data]
Ys = [point[1] for point in data]
Zs = [point[2] for point in data]
ts = [point[3] for point in data]

’’’fig = plt.figure()
ax = fig.add_subplot(111, projection=’3d’)
surf = ax.plot_trisurf(Xs, Ys, Zs, cmap=cm.jet, linewidth=0)
fig.colorbar(surf)
fig.tight_layout()
plt.show()’’’

import plotly.graph_objects as go

’’’marker_data = go.Scatter3d(
    x=Xs,
    y=Ys,
    z=Zs,
    marker=go.scatter3d.Marker(size=3),
    opacity=0.8,
    mode=’markers’
)’’’

data=[go.Scatter3d(x=Xs, y=Ys, z=Zs, mode=’markers’, marker_color=Zs, marker_colorscale=’Viridis’)]
#fig = go.Figure(go.Surface(x=Xs,y=Ys,z=Zs,colorscale="Plotly3",cmin=-5, cmax=5))
fig = go.Figure(data)

fig.update_layout(
    title=’something’,
    autosize=False,
    width=1000,
    height=1000,
        scene=dict(
        xaxis_title=’X Axis Title’,
        yaxis_title=’Y Axis Title’,
        zaxis_title='Z Axis Title',
    ),
)


'''fig = go.Figure(data=[go.Scatter3d(x=Xs, y=Ys, z=Zs, mode='markers',
                                   marker_color=ts, marker_colorscale='Viridis')])'''
fig.show()

'''fig = go.Figure(data=[go.Surface(z=Zs, x=Xs, y=Ys)])
fig.show()'''

#fig=go.Figure(data=marker_data)
#fig.show()

'''fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
surf = ax.plot_surface(M,beta,

plt.show()'''
```

Appendix B Problem 3 MATLAB Code

```
%% Compressible Flow - AEE 553
% Homework 5 - Problem 3
% Evan Burke
% 28 October 2022

clear; close; clc;

% Givens
th2 = 20; th3 = -15; % deg
M1 = 3; p1 = 1;
gamma = 1.4;

% Region 2 and 3 oblique shock solution
b2 = beta_solver(gamma,M1,th2);
b3 = beta_solver(gamma,M1,th3);

M1n2 = M1 * sind(b2);
M1n3 = M1 *sind(b3);

M2n = ((M1n2^2+2/(gamma-1)) / (2*gamma/(gamma-1)*M1n2^2-1))^0.5;
M3n = ((M1n3^2+2/(gamma-1)) / (2*gamma/(gamma-1)*M1n3^2-1))^0.5;

M2 = M2n/sind(b2-th2);
M3 = M3n/sind(b3-th3);

p2 = p1*(1 + 2*gamma/(gamma+1)*(M1n2^2-1));
p3 = p1*(1 + 2*gamma/(gamma+1)*(M1n3^2-1));

% Check for sign change in function
for i=1:20 % know that p4 is greater than 1, 20 seems high enough to find one sign change
    [b4,b4p,th4,th4p,diff] = shock_interaction(i,gamma,M2,M3,p2,p3,th2*pi/180,th3*pi/180);
    x0(i) = i;
    b4s(i) = b4;
    b4ps(i) = b4p;
    th4s(i) = th4;
    th4ps(i) = th4p;
    diffs(i) = diff;

    if diffs(i) < 0
        polarity(i) = -1;
    else
        polarity(i) = 1;
    end

    if i>1
```

```
        if polarity(i-1) ~= polarity(i)
            break
        end
    end
end

figure
plot(x0,diffs,[1,20],[0,0],'r')
grid
xlabel('P4 guess')
ylabel('Objective Function Value')
legend('Objective Function','Zero Value')

% Solver Initial Conditions
i = 2;
x(1) = x0(end-1); % last value before sign change
x(2) = x0(end); % final value of polarity check, opposite sign as x(end-1)
y(1) = diffs(end-1); % value associated with x(end-1)
y(2) = diffs(end); % value associated with x(2)
m(1) = 1; % initial slope needed for solution, arbitrary

% Bisection Method
while abs(diff) > 1e-5
    [b4,b4p,th4,th4p,diff] = shock_interaction(x(i),gamma,M2,M3,p2,p3,th2*pi/180,th3*pi/180);
    y(i) = diff;
    m = (y(i)-y(i-1)) / (x(i)-x(i-1));
    x(i+1) = -y(i)/m + x(i);
    i = i + 1;
end

fprintf('p4 = %f\n',x(i))
fprintf('b4 = %f\n',b4*180/pi)
fprintf('b4p = %f\n',b4p*180/pi)
fprintf('th4 = %f\n',th4*180/pi)
fprintf('th4p = %f\n',th4p*180/pi)

function [b4,b4p,th4,th4p,diff] = shock_interaction(p0,gamma,M2,M3,p2,p3,th2,th3)
    syms b4 b4p th4 th4p p4 % declare symbolic vars
    % currently accepts and outputs radians instead of degrees
    eq_1 = p4/p3 == 1 + 2*gamma/(gamma+1) * ((M3*sin(b4))^2-1); % Oblique shock eqn  p2/p1 across OS
    eq_2 = p4/p2 == 1 + 2*gamma/(gamma+1) * ((M2*sin(b4p))^2-1); % Oblique shock eqn, p2/p1 across OS
    eq_3 = tan(th4) == 2*cot(b4) * (M3^2*sin(b4)^2-1) / (M3^2 *(gamma + cos(2*b4)) + 2); % theta-beta-Mach relation
    eq_4 = tan(th4p) == 2*cot(b4p) * (M2^2*sin(b4p)^2-1) / (M2^2 *(gamma + cos(2*b4p)) + 2); % theta-beta-Mach relation
    eq_5 = solve(eq_1,b4); % solve for b4
    eq_6 = solve(eq_2,b4p); % solve for b4'
    eq_7 = solve(eq_3,th4); % solve for th4
    eq_8 = solve(eq_4,th4p); % solve for th4'

    b4i = subs(eq_5,p4,p0); % Placeholder value for beta4, extracting from syms
    b4 = double(abs(b4i(1))); % Converting beta4 val to double, taking positive, forming array
    b4pi = subs(eq_6,p4,p0); % Placeholder value for beta4'
    b4p = double(-abs(b4pi(1))); % Converting beta4' val to double, taking negative, forming array
    th4i = subs(eq_7,b4); % Placeholder value, theta4
    th4 = double(th4i); % Val to double, into array
    th4pi = subs(eq_8,b4p); % Placeholder value, theta4'
    th4p = double(th4pi); % Val to double, into array
    diff = th4 - th4p + th3 - th2; % 'Objective function', want 0 per constraints
end

function [beta] = beta_solver(gamma,M,theta)
    % accepts and returns degrees
    delta=1;
    theta=theta*pi/180;
    lamb = ((M^2-1).^2 - 3*(1 + (gamma-1)/2*M^2) * (1 + (gamma+1)/2*M.^2) * tan(theta)^2)^0.5;
    chi = ((M^2-1)^3 - 9 * (1 + (gamma-1)/2 * M^2) * (1 + (gamma-1)/2 * M^2 + (gamma+1)/4*M^4).*tan(theta)^2)/lamb^3;
    tan_beta = (M^2 - 1 + 2*lamb*cos((4*pi*delta+acos(chi))/3)) / (3 * (1 + (gamma-1)/2*M^2).*tan(theta));
    beta = atan(tan_beta)*180/pi;
end
```