AIM 5003 - Numerical Methods

Project #1: Simulating a Robotic Arm

Atreish Ramlakhan, Nosson Weissman, Aishwarya Singh

20/20

Strenghts :

① Required format was used

② Well written report

③ 2, 4, 6 roots were found

Weakness : Plots should have been included in the report.

Introduction:

This project demonstrates the value of a numerical method for identifying a root (i.e. a 0) of a polynomial if it exists. We use the Bisection Method to derive roots of a nested function of several variables we are given through an interval [-π,π]. The function seeks to calculate an angle θ that is most efficient given the generated values (x,y). This (x,y) point is generated from our imput values, so to ensure that this is calculated precicely so that all other imput parameters of the function operate within there constarints, we must generate the θ that does so by finding a root. First, we script the function with known imput parameters p1, p2, p3,L1,L2,L3,γ,x1,x2,y2 and use these to determine the unknown θ. Next, we graph the function in the interval [-π,π]. Next we observe the location of the roots, and graph a second scaled in view of the output. The final part of our experiment is to use a method, Bisection method was chosen, to find a root for the equation in a chosen interval (from the scaled in version of the graph) to the precision of a particular tolerance. This is repeated for the same function for 3 different sets of imput paramaters to determine the roots of the equation. ✓ ✓ ✓

Methodology:

We used the given forumla from the text when states that we are to use variables that are given p1, p2, p3, L1, L2, L3, γ, x1, x2, y2 and from there calculate unknown θ. The following equations are drafted from the parameters of the geometric interpretation:

$$p_1^2 = x^2 + y^2$$

$$p_2^2 = (x + A_2)^2$$
$$+ (y + B_2)^2$$

$$p_3^2 = (x + A_3^2)^2$$
$$+ (y + B_3^2)^2$$

$$A_2 = L_3 cos\theta - x_1$$
$$B_2 = L_3 sin\theta$$
$$A_3 = L_2 cos(\theta + \gamma) - x_2$$
$$= L_2 [cos\theta cos\gamma$$
$$- sin\theta sin\gamma] - x_2$$
$$B_3 = L_2 sin(\theta + \gamma) - y_2$$
$$= L_2 [cos\theta sin\gamma$$
$$+ sin\theta cos\gamma] - y_2$$

$$p_2^2 = x^2 + y^2 + 2A_2 x$$
$$+ 2B_2 y + A_2^2 + B_2^2$$

$$p_3^2 = x^2 + y^2 + 2A_3 x$$
$$+ 2B_3 y + A_3^2 + B_3^2$$

$$f = N_1^2 + N_1^2 - p_2^2 D^2$$
$$= 0$$

The last equation we must use to ensure that the correct θ is found by finiding the root. Since we will be using a continuous function that we must find a root for, we can use the bisection method very easily. There are no discontinuities just by viewing the graph so that we a Bisection section.

Computer Experiments:

We experiemented with the f0 function by firest imputing values x1 = 4, x2 = 1, y2 = 4, y1 = 0 and the remaining values were given. We then solved for the value of θ. This enabled us to graph the function from [-π,π]. This demonstrates where the function graphs and we can visually detect where the roots of the equation are by finding where the graphs equal 0. After we see that the graph crosses 0, we graph it again and use the bisection method code from class to find the roots. This gave use the roots for all 3 functions with different values for their imputs. In this particular experiment, we used the numpy linspace function with 1000 pieces of information, this returns evenly spaced numbers in an interval and then the numpy vectorize function that takes in the 1000 values into a numpy array and graphs it. This works for all graphs. We are able to zoom in and figure out the necessary intervals for the roots of the individual functions and then with the selected tolerance using the bisection method we found all roots.

In [ ]:

```
import math
import numpy as np
import matplotlib.pyplot as plt
```

1 Write function and test for parameter values

In [ ]:

```
def f0(Theta, L1, L2, L3, x1, x2, y1, y2, P1, P2, P3, Alpha):

        A2 = (-x1) + L3*math.cos(Theta)
        B2 = L3*math.sin(Theta)

        A3 = (-x2) + L2*math.cos(Theta)*math.cos(Alpha) - L2*math.sin(Theta)*math.sin(Al
pha)
        B3 = (-y2) + L2*math.sin(Theta)*math.cos(Alpha) + L2*math.cos(Theta)*math.sin(Al
pha)

        N1 = B3*(P2**2 - P1**2 - A2**2 - B2**2) - B2*(P3**2 - P1**2 - A3**2 - B3**2)
        N2 = (-A3)*(P2**2 - P1**2 - A2**2 - B2**2) + A2*(P3**2 - P1**2 - A3**2 - B3**2)
        D = 2*(A2*B3 - A3*B2)

        x = N1/D
        y = N2/D

        fofTheta = N1**2 + N2**2 - P1**2*D**2
        return fofTheta, x, y
```

In [ ]:

```
def f1(Theta,
        L1 = 2, L2 = math.sqrt(2), L3 = math.sqrt(2), #given values in the problem
        x1 = 4, x2 = 1, y2 = 4, y1 = 0, #random values chosen
        P1 = math.sqrt(5), P2 = math.sqrt(5), P3 = math.sqrt(5), #given values
        Alpha = math.pi/2):

        return f0(Theta, L1, L2, L3, x1, x2, y1, y2, P1, P2, P3, Alpha)
```

In [ ]:

```
Theta = math.pi/(2) #Testing Values #2 on pg 72
f1(Theta)[0]
```

Out[ ]:

```
4967.5272708720795
```

In [ ]:

```
Theta = math.pi/(-2)
f1(Theta)[0]
```

Out[ ]:

```
2116.4727291279196
```

✓

2 Plot f(theta) for theta in [-pi,pi] Approximately localize roots

In [ ]:

```
plt.style.use('seaborn-whitegrid')
plt.figure(figsize=(15,8))
ax = plt.axes()
x = np.linspace(-math.pi,math.pi,1000)
f2 = np.vectorize(f1)
y = f2(x)[0]
plt.plot(x,y, 'red')
plt.show()
```

In [ ]:

```
plt.style.use('seaborn-whitegrid')
plt.figure(figsize=(15,8))
plt.xlim([-1.5, 1.5])
plt.ylim([-25, 25])
plt.plot(x,y, 'red')
plt.show()

#the range of the roots of the equation are shown in the graph. They are between [-1.5,1]
on the X-axis
```

3 Solve for Theta using an equation solver Method from Ch1. There can be at most 6 roots for Theta, then for x and y

In [ ]:

```
def bisection(f, a, b, TOL):
    if np.sign(f(a)[0])*np.sign(f(b)[0]) > 0: # sign is determin if symbol is -1, 0, or
1
        print('f(a)f(b)<0 not satisfied')
        return # stop execution
    n=1
    fa= f(a)[0]
    fb= f(b)[0]
    while np.abs(a-b)>TOL:
        c = (a+b)/2
        fc=f(c)[0]
        n=n+1
        if np.isclose(f(c)[0], 0):
            print('Approximate  root', c, 'has been obtained in', n, 'steps')
            return
        if np.sign(fc)*np.sign(fa)<0:
            b = c
            fb=fc
        else:
            a = c
            fa= fc
    c=(a+b)/2
    print('The final interval [', a, b, '] contains a root')
    print('Approximate  root', c, 'has been obtained in', n, 'steps')
    return c
```

2 roots ✓

In [ ]:

```
print( bisection(f1,-1.5,-0.5,0.5e-5))
print( bisection(f1,0.5,1.5,0.5e-5))
```

```
The final interval [ -1.161712646484375 -1.1617088317871094 ] contains a root
Approximate  root -1.1617107391357422 has been obtained in 19 steps
-1.1617107391357422
The final interval [ 0.890045166015625 0.8900489807128906 ] contains a root
Approximate  root 0.8900470733642578 has been obtained in 19 steps
0.8900470733642578
```

4 roots

In [ ]:

```
def f4(Theta,
        L1 =3, L2=3*math.sqrt(2),L3=3,
        x1=5, x2=0, y2=6, y1=0,
        P1=5, P2=5,P3=3,
        Alpha=math.pi/4):

        return f0(Theta, L1, L2, L3, x1, x2, y1, y2, P1, P2, P3, Alpha)
```

In [ ]:

```
plt.style.use('seaborn-whitegrid')
plt.figure(figsize=(15,8))
ax = plt.axes()
x = np.linspace(-math.pi,math.pi,1000)
f5 = np.vectorize(f4)
y = f5(x)[0]
plt.plot(x,y, 'red')
plt.show()
```

In [ ]:

```
plt.style.use('seaborn-whitegrid')
plt.figure(figsize=(15,8))
plt.xlim([-2, 2.5])
plt.ylim([-25, 25])
plt.plot(x,y, 'red')
plt.show()
```

In [ ]:

```
print(bisection(f4,-1,-0.5,0.5e-5))
print(bisection(f4,-0.5,0,0.5e-5))
print(bisection(f4,1,1.5,0.5e-5))
print(bisection(f4,2,2.5,0.5e-5))
```

```
The final interval [ -0.7208518981933594 -0.7208480834960938 ] contains a root
Approximate  root -0.7208499908447266 has been obtained in 18 steps        ✓
-0.7208499908447266
The final interval [ -0.3310089111328125 -0.3310050964355469 ] contains a root
Approximate  root -0.3310070037841797 has been obtained in 18 steps        ✓
-0.3310070037841797
The final interval [ 1.1436843872070312 1.1436882019042969 ] contains a root ✓
Approximate  root 1.143686294555664 has been obtained in 18 steps
1.143686294555664
The final interval [ 2.11590576171875 2.1159095764160156 ] contains a root
Approximate  root 2.115907669067383 has been obtained in 18 steps          ✓
2.115907669067383
```

6 roots

In [ ]:

```
def f6(Theta,
        L1 =3, L2=3*math.sqrt(2),L3=3,
        x1=5, x2=0, y2=6, y1=0,
        P1=5, P2=7,P3=3,
        Alpha=math.pi/4):
```

```
        return f0(Theta, L1, L2, L3, x1, x2, y1, y2, P1, P2, P3, Alpha)
```

In [ ]:

```
plt.style.use('seaborn-whitegrid')
plt.figure(figsize=(15,8))
ax = plt.axes()
x = np.linspace(-math.pi,math.pi,1000)
f7 = np.vectorize(f6)
y = f7(x)[0]
plt.plot(x,y, 'red')
plt.show()
```

In [ ]:

```
plt.style.use('seaborn-whitegrid')
plt.figure(figsize=(15,8))
plt.xlim([-2, 3])
plt.ylim([-25, 25])
plt.plot(x,y, 'red')
plt.show()
```

In [ ]:

```
print(bisection(f6,-1,-0.5,0.5e-5))
print(bisection(f6,-0.5,0,0.5e-5))
print(bisection(f6,0,0.25,0.5e-5))
print(bisection(f6,0.25,0.5,0.5e-5))
print(bisection(f6,0.5,1,0.5e-5))
print(bisection(f6,2,3,0.5e-5))
```

```
The final interval [ -0.6731605529785156 -0.67315673828125 ] contains a root
Approximate  root -0.6731586456298828 has been obtained in 18 steps      ✓
-0.6731586456298828
The final interval [ -0.35474395751953125 -0.3547401428222656 ] contains a root
Approximate  root -0.35474205017089844 has been obtained in 18 steps     ✓
-0.35474205017089844
The final interval [ 0.0377655029296875 0.037769317626953125 ] contains a root
Approximate  root 0.03776741027832031 has been obtained in 17 steps      ✓
0.03776741027832031
The final interval [ 0.4588775634765625 0.4588813781738281 ] contains a root
Approximate  root 0.4588794708251953 has been obtained in 17 steps       ✓
0.4588794708251953
The final interval [ 0.9776725769042969 0.9776763916015625 ] contains a root
Approximate  root 0.9776744842529297 has been obtained in 18 steps       ✓
0.9776744842529297
The final interval [ 2.5138511657714844 2.51385498046875 ] contains a root
Approximate  root 2.513853073120117 has been obtained in 19 steps        ✓
2.513853073120117
```

Conclusion:

The function f0 effectively found a suitable angle θ for the parameter values indicated p1, p2, p3,L1,L2,L3,γ,x1,x2,y2. Since these values were known, it was within the function itself to find a point (x,y). However, to be certain that this point actually works for the given parameters, we must ensure that the equation f = N_1^2 + N_2^2 – p_1^2*D^2 = 0. This is a classic problem of finding a root. So we graphed the 3 different sets of imput paramaters and found that indeed we have 2, 4 and 6 roots. The first set of imput parameters were guessed and the final 2 were taken from the text pg. 72. After we graph the function on the interval [-π,π] we observe the location of the roots, i.e. where the function hits 0 and then we localize in with another graph. This 2nd graph allowed up to determine intervals, and how many roots we have so that we apply the Bisection method for finding their approximate values with a certain tolerance. This procedure worked well for all equations.

References Cited:

Sauer, T. (2017). Numerical Analysis (3rd ed.). Upper Saddle River, NJ: Pearson

Sauer, T. (2017). Numerical Analysis (3rd ed.). Upper Saddle River, NJ: Pearson

https://numpy.org/doc/stable/reference/generated/numpy.linspace.html

https://numpy.org/doc/stable/reference/generated/numpy.vectorize.html