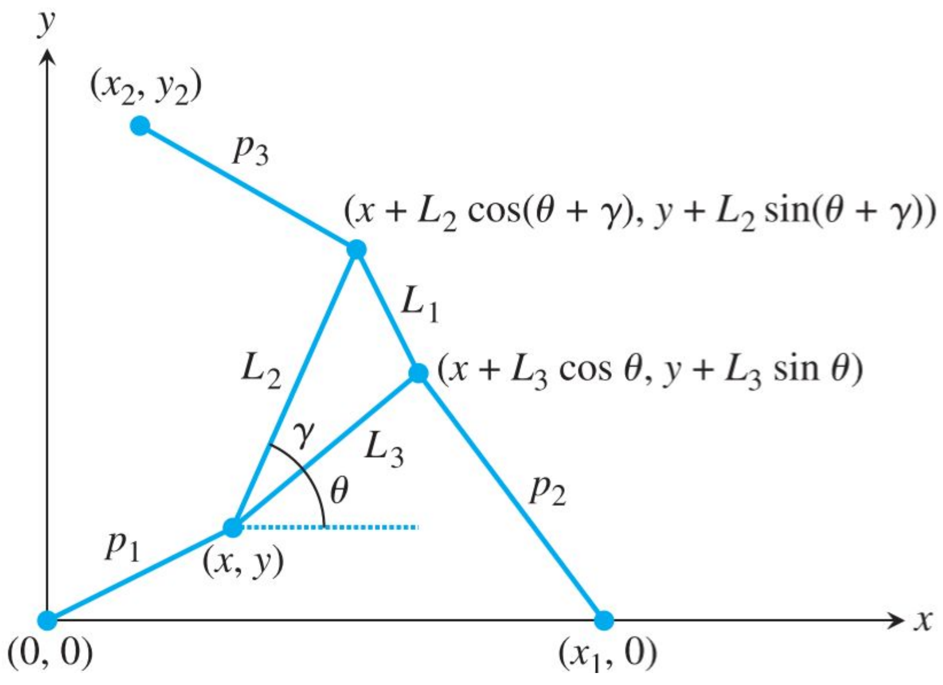
Scientific Computing: Project 1

Brian Livian, Xingyu Liu, Yonah Moise

Introduction

Scientific computing or numerical methods is a powerful discipline of mathematics that can be used to solve complicated problems. Scientific computing can be used to describe the problem of the robotic arm, also known as the Stewart platform. A Stewart platform consists of 6 struts: P1, P2, P3, L1, L2. L3. P1, 2, and 3 are dynamic joints, meaning they change length. L1, 2, and 3 are formed at a fixed length to form a triangle. Thus, the structure has six degrees of freedom. ɤ is the angle opposite to L1. θ is the angle between L3 and the horizontal. A picture of a two dimensional model of the Stewart platform is found below [1]:


y
(x_2, y_2)
p_3
$(x + L_2\cos(\theta+\gamma), y + L_2\sin(\theta+\gamma))$
L_1
L_2
$(x + L_3\cos\theta, y + L_3\sin\theta)$
γ
L_3
θ
p_2
p_1
(x, y)
x
$(0, 0)$
$(x_1, 0)$

The problem we are tasked with is to create a function with inputs of L1, L2, L3, x1, y1, x2, y2 and ɣ, with an output of f(θ). We are then tasked to plot f(θ) on the interval [-π, π] (being that f(θ) is 2π periodic). Lastly, we are tasked to use inputs of L1, L2, L3, (x1, y1), (x2, y2) and ɣ to calculate the position of (x,y).

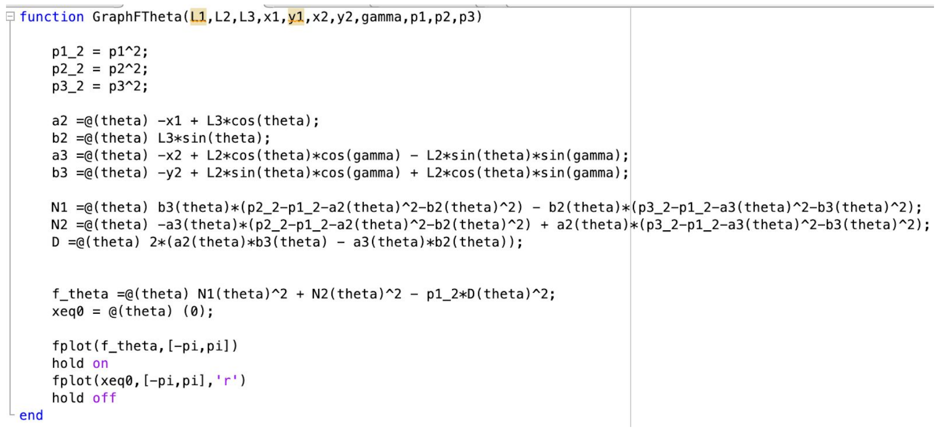
Methodology

We used Matlab and Python to write the programs for this assignment. (See Appendix, where Xingyu Liu and Brian Livian used Python for this assignment.)

For problem 1, we wrote a program that calculates the function, *f(θ)*, given inputs for the variables *L1, L2, L3, P1, P2, P3*, and Ɣ.[1] The function was tested for *f(θ)* given the parameter input values, by hard-programming the two roots *θ = π/4, -π/4*.
For problem 2, we plotted *f(θ)*, for the given inputs of problem 1, on the interval [-π,π].
For problem 3, we used the Bisection Method to calculate the approximate roots for *f(θ)*, given new parameter input values. To do so, we used the code from problem 2 to plot a graph of our *f(θ)*. From this graph, we identified how many roots exist, and the approximate intervals which contain these roots. We subsequently input these intervals into our Bisection Method program, to produce the approximate roots for *f(θ)*. After having identified an approximate root *θ*, we are able to calculate *x* and *y* using their definitions. [1]

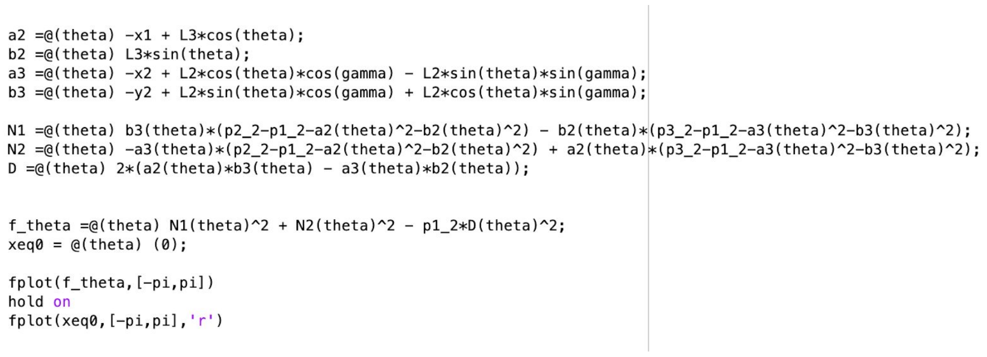
<u>**Computer Simulations and Results**</u> [Yonah Moise also approached this assignment using Matlab.]

First, I wrote a program for the function *f(θ)*, and plugged in the test-values (given in problem #1) to ensure that the program was working correctly. Using values to give us an *f(θ)* with 2 roots, *π/4* and *-π/4*, we verified that *f(π/4) = 0* and *f(-π/4) = 0*.

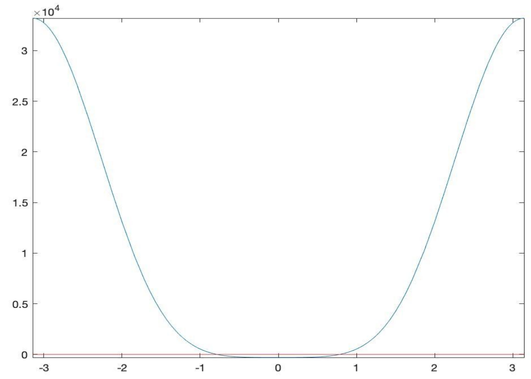


```
function GraphFTheta(L1,L2,L3,x1,y1,x2,y2,gamma,p1,p2,p3)

    p1_2 = p1^2;
    p2_2 = p2^2;
    p3_2 = p3^2;

    a2 =@(theta) -x1 + L3*cos(theta);
    b2 =@(theta) L3*sin(theta);
    a3 =@(theta) -x2 + L2*cos(theta)*cos(gamma) - L2*sin(theta)*sin(gamma);
    b3 =@(theta) -y2 + L2*sin(theta)*cos(gamma) + L2*cos(theta)*sin(gamma);

    N1 =@(theta) b3(theta)*(p2_2-p1_2-a2(theta)^2-b2(theta)^2) - b2(theta)*(p3_2-p1_2-a3(theta)^2-b3(theta)^2);
    N2 =@(theta) -a3(theta)*(p2_2-p1_2-a2(theta)^2-b2(theta)^2) + a2(theta)*(p3_2-p1_2-a3(theta)^2-b3(theta)^2);
    D =@(theta) 2*(a2(theta)*b3(theta) - a3(theta)*b2(theta));


    f_theta =@(theta) N1(theta)^2 + N2(theta)^2 - p1_2*D(theta)^2;
    xeq0 = @(theta) (0);

    fplot(f_theta,[-pi,pi])
    hold on
    fplot(xeq0,[-pi,pi],'r')
    hold off
end
```

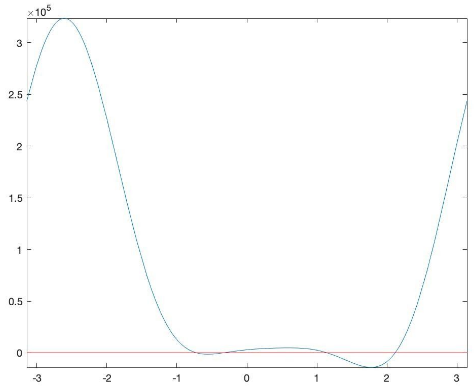
Subsequently, I used a similar code to graph the test-case (provided through the values in problem #1).



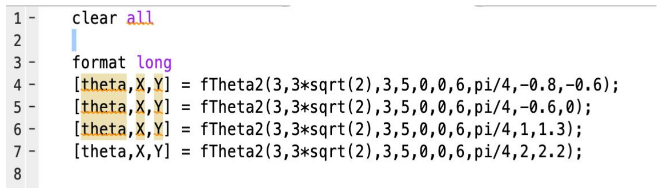
```
a2 =@(theta) -x1 + L3*cos(theta);
b2 =@(theta) L3*sin(theta);
a3 =@(theta) -x2 + L2*cos(theta)*cos(gamma) - L2*sin(theta)*sin(gamma);
b3 =@(theta) -y2 + L2*sin(theta)*cos(gamma) + L2*cos(theta)*sin(gamma);

N1 =@(theta) b3(theta)*(p2_2-p1_2-a2(theta)^2-b2(theta)^2) - b2(theta)*(p3_2-p1_2-a3(theta)^2-b3(theta)^2);
N2 =@(theta) -a3(theta)*(p2_2-p1_2-a2(theta)^2-b2(theta)^2) + a2(theta)*(p3_2-p1_2-a3(theta)^2-b3(theta)^2);
D =@(theta) 2*(a2(theta)*b3(theta) - a3(theta)*b2(theta));


f_theta =@(theta) N1(theta)^2 + N2(theta)^2 - p1_2*D(theta)^2;
xeq0 = @(theta) (0);

fplot(f_theta,[-pi,pi])
hold on
fplot(xeq0,[-pi,pi],'r')
```



Finally, I wrote another function for the Bisection Method, which returns the approximate-roots *θ*, for the case of four roots (as given in problem #3). This function also returns the values of *x* and *y* calculated using the approximate-roots. But first I had to identify the domains of the roots, by graphing *f(θ)* for the given-values.



Having graphed *f(θ)* for the values given for problem #3, and thereby identifying the domains containing the four roots, I input them into the program (together with the other given-values for *f(θ)*).



```
clear all

format long
[theta,X,Y] = fTheta2(3,3*sqrt(2),3,5,0,0,6,pi/4,-0.8,-0.6);
[theta,X,Y] = fTheta2(3,3*sqrt(2),3,5,0,0,6,pi/4,-0.6,0);
[theta,X,Y] = fTheta2(3,3*sqrt(2),3,5,0,0,6,pi/4,1,1.3);
[theta,X,Y] = fTheta2(3,3*sqrt(2),3,5,0,0,6,pi/4,2,2.2);
```

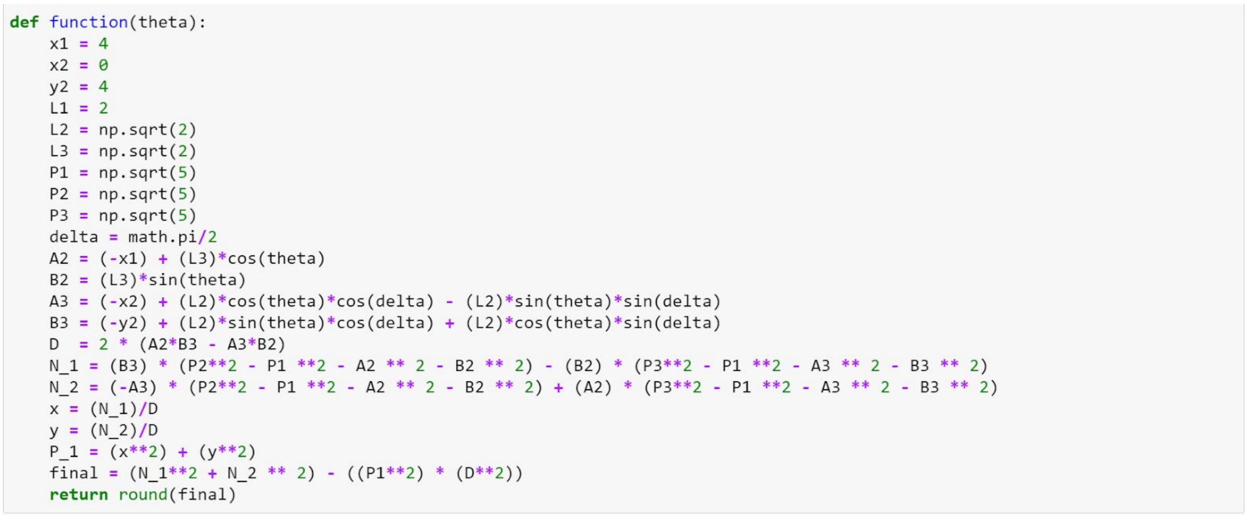
Having run the function for the four aforementioned domains, it yielded the results contained in the table below.

Root #	θ	*x*	*y*
1	-0.7208	-1.378	4.8063

2	-0.3310	-0.9147	4.9156
3	1.1437	4.4817	2.2167
4	2.1159	4.5718	2.0244

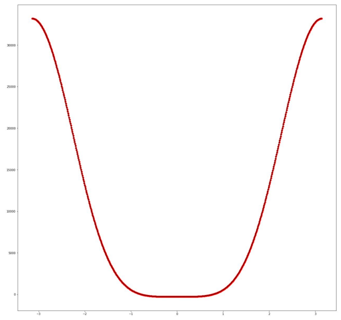
Appendix

Here we have our *f(θ)* function which we wrote in Python, with parameters for x1, x2, y2, L1, L2, L3, P1, P2, P3 and ɣ:

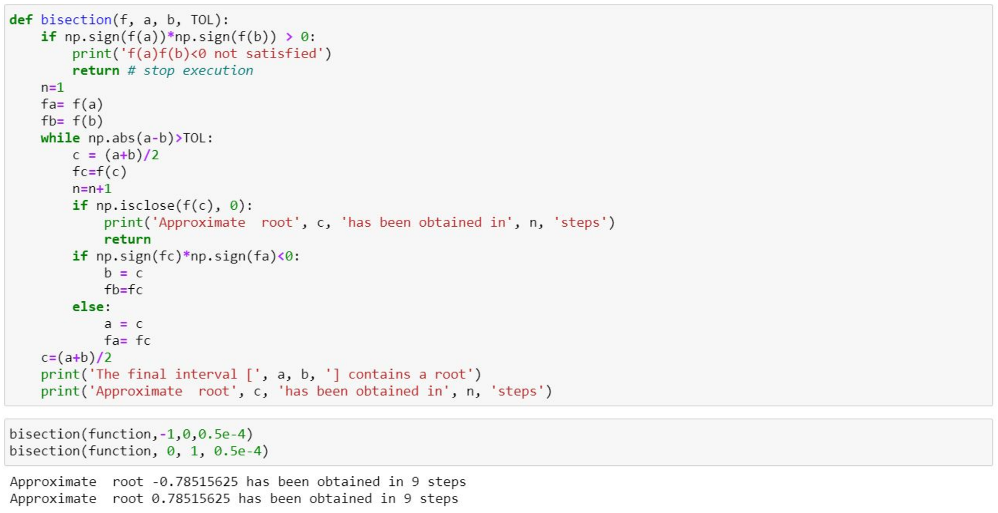


```python
def function(theta):
    x1 = 4
    x2 = 0
    y2 = 4
    L1 = 2
    L2 = np.sqrt(2)
    L3 = np.sqrt(2)
    P1 = np.sqrt(5)
    P2 = np.sqrt(5)
    P3 = np.sqrt(5)
    delta = math.pi/2
    A2 = (-x1) + (L3)*cos(theta)
    B2 = (L3)*sin(theta)
    A3 = (-x2) + (L2)*cos(theta)*cos(delta) - (L2)*sin(theta)*sin(delta)
    B3 = (-y2) + (L2)*sin(theta)*cos(delta) + (L2)*cos(theta)*sin(delta)
    D  = 2 * (A2*B3 - A3*B2)
    N_1 = (B3) * (P2**2 - P1 **2 - A2 ** 2 - B2 ** 2) - (B2) * (P3**2 - P1 **2 - A3 ** 2 - B3 ** 2)
    N_2 = (-A3) * (P2**2 - P1 **2 - A2 ** 2 - B2 ** 2) + (A2) * (P3**2 - P1 **2 - A3 ** 2 - B3 ** 2)
    x = (N_1)/D
    y = (N_2)/D
    P_1 = (x**2) + (y**2)
    final = (N_1**2 + N_2 ** 2) - ((P1**2) * (D**2))
    return round(final)
```

Below is a graph of *f(θ)* (note that *f(θ)* has 2 zeros):



Below is the code used for the "bisection method" [2]. The bisection function calculates the roots of a given function within a given interval and tolerance. The function repeatedly divides the interval by 2 while the difference of a-b > tolerance, thus obtaining a zero. The intervals below were found graphically, thus obtaining roots at ± 0.78515625. Each root represents a position for the robotic arm:

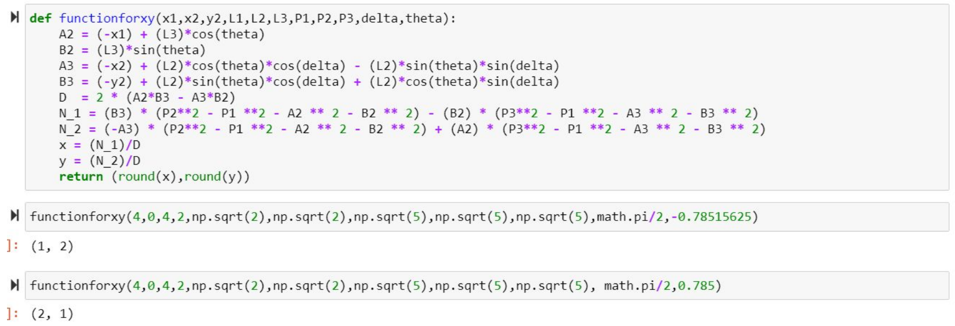


```
def bisection(f, a, b, TOL):
    if np.sign(f(a))*np.sign(f(b)) > 0:
        print('f(a)f(b)<0 not satisfied')
        return # stop execution
    n=1
    fa= f(a)
    fb= f(b)
    while np.abs(a-b)>TOL:
        c = (a+b)/2
        fc=f(c)
        n=n+1
        if np.isclose(f(c), 0):
            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')
```

```
bisection(function,-1,0,0.5e-4)
bisection(function, 0, 1, 0.5e-4)
```

```
Approximate  root -0.78515625 has been obtained in 9 steps
Approximate  root 0.78515625 has been obtained in 9 steps
```

Below is a function- functionforxy. The function takes inputs for *x1, x2, y2, L1, L2, L3, P1, P2, P3, θ* and *Ɣ*. The function returns an output for (*x,y*). Note that since *f(θ)* has 2 roots, there are 2 corresponding (*x,y*) values, or positions for the robotic arm:



```
def functionforxy(x1,x2,y2,L1,L2,L3,P1,P2,P3,delta,theta):
    A2 = (-x1) + (L3)*cos(theta)
    B2 = (L3)*sin(theta)
    A3 = (-x2) + (L2)*cos(theta)*cos(delta) - (L2)*sin(theta)*sin(delta)
    B3 = (-y2) + (L2)*sin(theta)*cos(delta) + (L2)*cos(theta)*sin(delta)
    D  = 2 * (A2*B3 - A3*B2)
    N_1 = (B3) * (P2**2 - P1 **2 - A2 ** 2 - B2 ** 2) - (B2) * (P3**2 - P1 **2 - A3 ** 2 - B3 ** 2)
    N_2 = (-A3) * (P2**2 - P1 **2 - A2 ** 2 - B2 ** 2) + (A2) * (P3**2 - P1 **2 - A3 ** 2 - B3 ** 2)
    x = (N_1)/D
    y = (N_2)/D
    return (round(x),round(y))
```

```
functionforxy(4,0,4,2,np.sqrt(2),np.sqrt(2),np.sqrt(5),np.sqrt(5),np.sqrt(5),math.pi/2,-0.78515625)
```

```
]: (1, 2)
```

```
functionforxy(4,0,4,2,np.sqrt(2),np.sqrt(2),np.sqrt(5),np.sqrt(5),np.sqrt(5), math.pi/2,0.785)
```

```
]: (2, 1)
```

Conclusion

The positions for the two dimensional Stewart platforms in problem #1 can be calculated by use of scientific computing.We confirm that *f(θ) = 0* when setting *θ = π/4* and *-π/4*. However, this problem can extend to more than two positions. The robotic arm can have a number of positions, corresponding to the number of roots the given function *f(θ)* has. Functions with different parameters for *x1, x2, y2, L1, L2, L3, P1, P2, P3* and *ɣ* will have different numbers of roots, and thus different amount of positions for the robotic arm (with a maximum of 6). Furthermore, the same function used to calculate *θ*, includes the calculation of the point (*x,y*).

Citations:

[1] Sauer, Timothy (2017). Numerical Analysis, 3rd. edition. Pearson. ISBN-13: 978-0134696454. pp. 70-72.

[2] From Professor Marian Gidea, in his class slides for class #2 of Numerical Methods/Scientific Computing.