Reality Check 1, Stewart Platform in 2 Dimensions

Jenaveve White

Reality Check

Jenaveve White

Introduction

A Stewart platform is a versatile parallel manipulator characterized by six degrees of freedom, allowing for precise control of a mobile platform connected to a fixed base via six adjustable legs. Developed by Eric G. Stewart in the 1960s, these platforms can move freely in three-dimensional space, making them ideal for various applications, including robotics, flight simulators, medical devices, and aerospace testing. Their advantages include high precision, versatility in motion profiles, and a compact design, which enable them to effectively accommodate diverse payloads and operational requirements. Overall, Stewart platforms play a crucial role in advancing technology across multiple fields by providing sophisticated motion control solutions.

Question 1

Write a Python function for f(θ). The parameters $L_1, L_2, L_3, \gamma, x_1, x_2, y_2$ are fixed constants, and the strut lengths p1, p2, p3 will be known for a given pose.

To write the function for f(θ) I created a function named f that takes in a parameter θ and calculates f(θ). The parameters that are fixed were set to global variables and later called inside of my f function. From our text, we have relationships for our 2 dimension stewart platform as follows:

$$A_2 = L_3 cos(\theta) - x1$$

$$B_2 = L_3 sin(\theta)$$

$$A_3 = L_2 cos(\theta + \gamma) - x_2$$

$$B_3 = L_2 sin(\theta + \gamma) - y_2$$

$$D = 2(A_2 B_3 - B_2 A_3)$$

$$N_1 = B_3(p_2^2 - p_1^2 - A_2^2 - B_2^2) - B_2(p_3^2 - p_1^2 - A_3^2 - B_3^2)$$

$$N_2 = -A_3(p_2^2 - p_1^2 - A_2^2 - B_2^2) + A_2(p_3^2 - p_1^2 - A_3^2 - B_3^2)$$

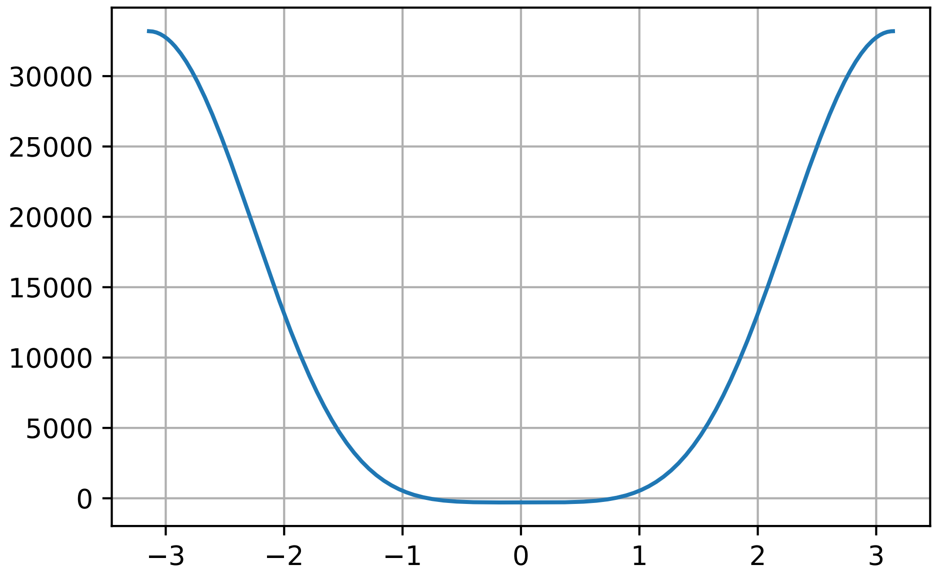
Using these equations, the function calculates $N_1^2 + N_2^2 - p_1^2 D^2$, which should output 0, if working correctly.

After creating this function, to ensure it is finding the roots correctly my code returned a value of .0000000000004, which for our purposes is essentially zero. This tells us that the function is working properly. The miniscule value is likely due to a computational rounding error, which is fairly insignificant.

Question 2

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

L1 = 2
L2 = np.sqrt(2)
L3 = np.sqrt(2)
gamma = np.pi /2
x1 = 4
x2 = 0
y2 = 4
p1 = np.sqrt(5)
p2 = p1
p3 = p1
# Question 1
def f(theta):
    A2 = L3*np.cos(theta)-x1
    B2= L3*np.sin(theta)
    A3 = L2*np.cos(theta + gamma) - x2
    B3 = L2*np.sin(theta+ gamma) -y2
    D = 2 * (A2* B3 - B2*A3)
    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)
    return N1**2+N2**2-p1**2*D**2


theta = np.pi/4
x_array = np.linspace(-np.pi, np.pi,400)
plt.plot(x_array, f(x_array))
plt.grid()
plt.show()


#def triangle(p1, p2, gamma, theta):
#
#    A2 = L3*np.cos(theta)-x1
#    B2= L3*np.sin(theta)
#    A3 = L2*np.cos(theta + gamma) - x2
#    B3 = L2*np.sin(theta+ gamma) -y2
#    D = 2 * (A2* B3 - B2*A3)
#    N1 = B3*(p2**2-p1**2-A2**2-B2)-B2*(p3**2-p1**2-A3**2-B3**2)
#    N2 = -A3*(p2**2-p1**2-A2**2-B2)+A2*(p3**2-p1**2-A3**2-B3**2)
```

```
#
#    x = N1 / D
#    y = N2 / D
#
#
#    answ = [x,y, theta]
#    return answ
#
```

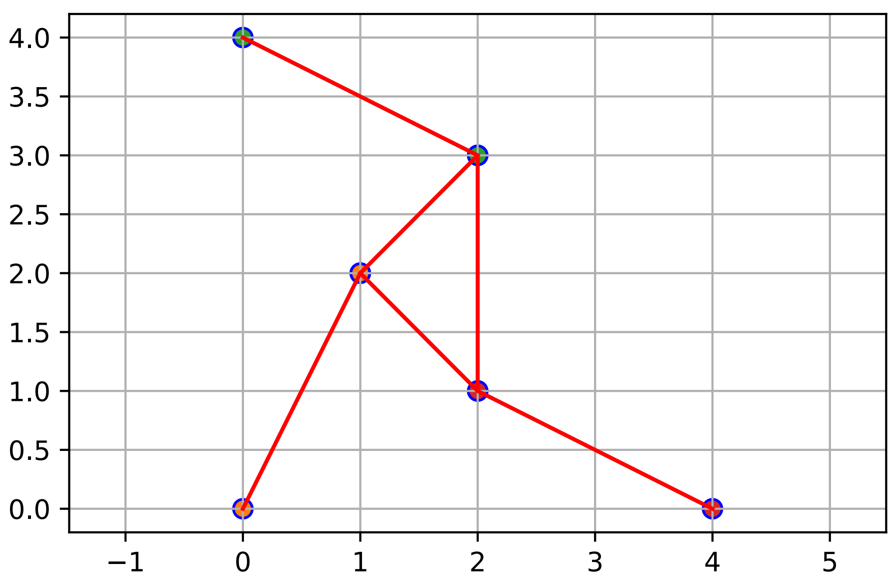


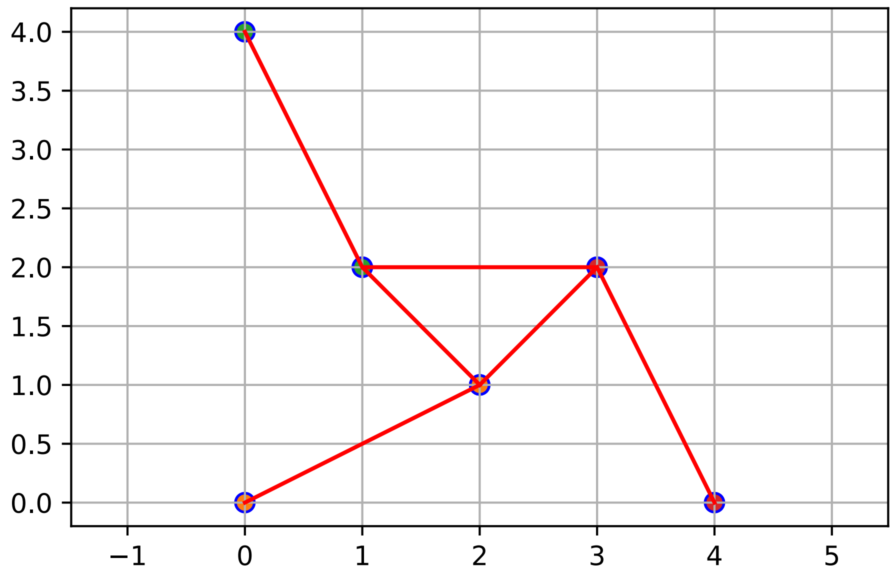
Question 3

Reproduce Figure 1.15.

```
def plot_triangle(point1, point2, point3, x1, x2 , y2):


    # Prepare x and y coordinates
    x = [point1[0], point2[0], point3[0], point1[0]]  # Closing the triangle
    y = [point1[1], point2[1], point3[1], point1[1]]  # Closing the triangle
    strut1x = [0, point1[0]]
    strut1y= [0, point1[1]]
    strut2x = [0, point2[0]]
    strut2y = [x1,point2[1]]
    strut3x = [x2, point3[0]]
    strut3y = [y2, point3[1]]

    # Create the plot
    plt.figure()
```

```python
    plt.plot(x, y, linestyle='-', color='red')
    plt.plot(strut1x, strut1y, linestyle='-', color='red')
    plt.plot(strut2x, strut2y, linestyle='-', color='red')
    plt.plot(strut3x, strut3y, linestyle='-', color='red')


# Scatter with small open circles
    plt.scatter(x, y, marker='o', edgecolor='blue', s=50)
    plt.scatter(strut1x, strut1y, marker='o', edgecolor='blue', s=50 )
    plt.scatter(strut2x, strut2y, marker='o', edgecolor='blue', s=50 )
    plt.scatter(strut3x, strut3y, marker='o', edgecolor='blue', s=50 )


    # Set aspect ratio and limits
    plt.axis('equal')
    plt.grid()


#testing my triangle function

plot_triangle((1,2),(2,3),(2,1), 4,4,0)
plot_triangle((2,1),(1,2),(3,2), 4,4,0)
```



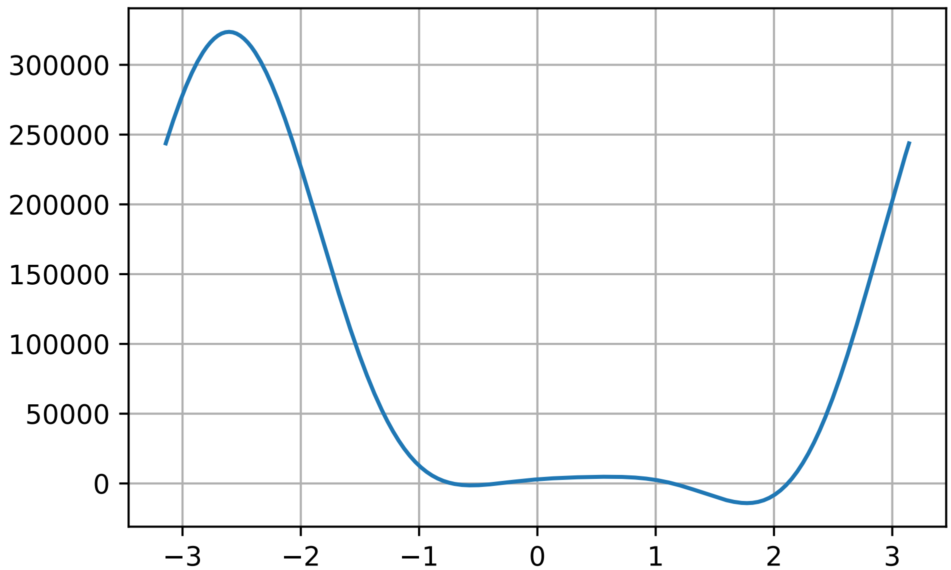


Question 4

Solve the forward kinematics problem for the planar Stewart platform specified by $x1 = 5$,$(x2, y2) = (0, 6)$, $L1 = L3 = 3$, $L2 = 3\sqrt{2}$, $\gamma = \pi/4$, $p1 = p2 = 5$, $p3 = 3$. Begin by plotting $f(\theta)$. Use an equation solver of your choice to find all four poses (roots of $f(\theta)$), and plot them. Check your answers by verifying that p1, p2, p3 are the lengths of the struts in your plot.

```
from scipy.optimize import fsolve
import numpy as np
import matplotlib.pyplot as plt

import plotly_express as px

L1 = 3
L2 = 3 * np.sqrt(2)
L3 = 3
gamma = np.pi /4
x1 = 5
x2 = 0
y2 = 6
p1 = 5
p2 = 5
p3 = 3

# Define the function f(θ) from earlier
def f(theta):
    A2 = L3 * np.cos(theta) - x1
    B2 = L3 * np.sin(theta)
    A3 = L2 * np.cos(theta + gamma) - x2
```

```
    B3 = L2 * np.sin(theta + gamma) - y2
    D = 2 * (A2 * B3 - B2 * A3)
    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)
    return N1**2 + N2**2 - p1**2 * D**2

x_array = np.linspace(-np.pi, np.pi,400)
plt.plot(x_array, f(x_array))
plt.grid()
plt.show()

#def rootfinder(start, stop, num):
#     initial_guesses = np.linspace(start, stop, num)  # Multiple guesses for
better coverage
#    roots = []
#
#    for i in initial_guesses:
#        root = fsolve(f, i)
#        # Add unique roots only
#        if root not in roots :
#            roots.append(root)
#
#    roots = np.array(roots).flatten()
#    print("Roots found:", roots)
#
#    # Convert roots to a more usable format
#    return roots
#
## Print the roots
#rootfinder(-np.pi, np.pi, 5)
```

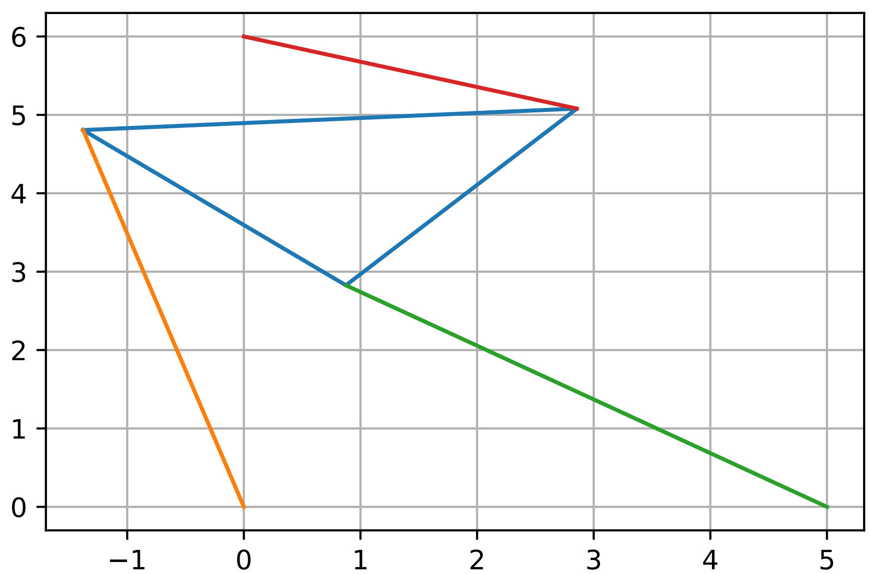


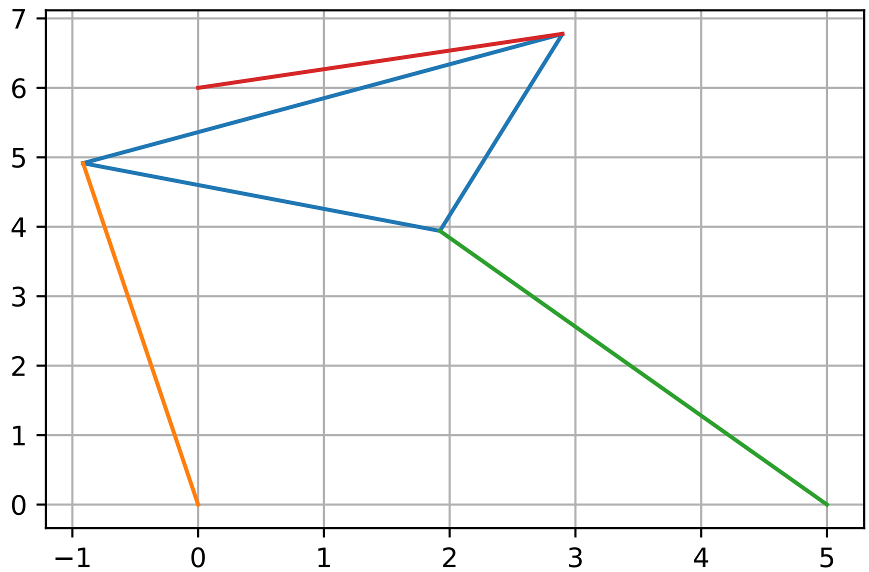

```
import numpy as np
import matplotlib.pyplot as plt

L1 = 3
L2 = 3 * np.sqrt(2)
L3 = 3
gamma = np.pi / 4
x1 = 5
x2 = 0
y2 = 6
p1 = 5
p2 = 5
p3 = 3


# Define the function f(θ) from earlier
def f(theta):
    A2 = L3 * np.cos(theta) - x1
    B2 = L3 * np.sin(theta)
    A3 = L2 * np.cos(theta + gamma) - x2
    B3 = L2 * np.sin(theta + gamma) - y2
    D = 2 * (A2 * B3 - B2 * A3)
    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)
    return N1**2 + N2**2 - p1**2 * D**2

def triangleplotting(theta):
```

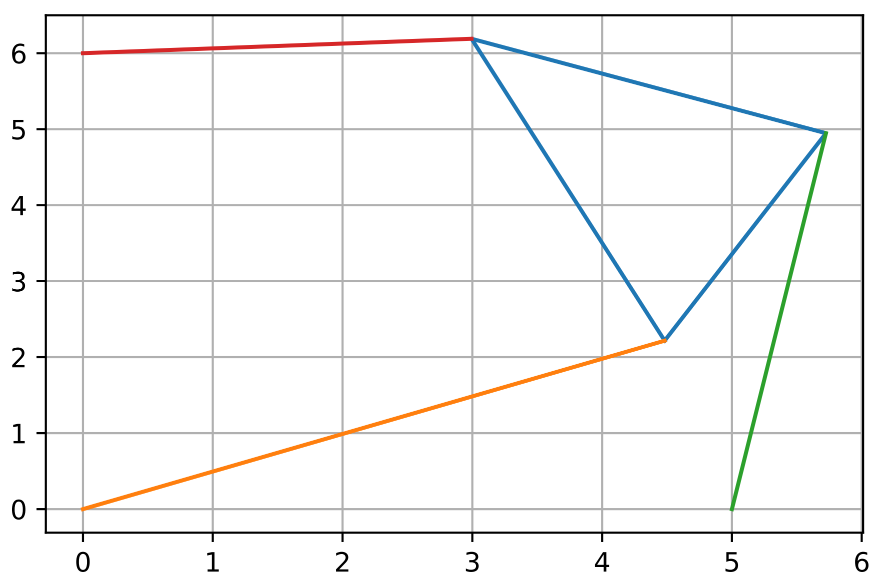
```
    A2 = L3*np.cos(theta)-x1
    B2= L3*np.sin(theta)
    A3 = L2*np.cos(theta + gamma) - x2
    B3 = L2*np.sin(theta+ gamma) -y2
    D = 2 * (A2* B3 - B2*A3)
    N1 = B3*(p2**2-p1**2-A2**2-B2**2)-B2*(p3**2-p1**2-A3**2-B3**2)
    N2 = -A3*(p2**2-p1**2-A2**2-B2**2)+A2*(p3**2-p1**2-A3**2-B3**2)
    x = N1/D
    y = N2/D

    #coordinate points for the triangle
    u1 = N1/D
    u2 = x + L3 * np.cos(theta)
    u3 = x + L2 * np.cos(theta + gamma)
    m1 = N2/D
    m2 = y + L3 * np.sin(theta)
    m3 = y + L2 * np.sin(gamma + theta)

    plt.grid()
    plt.autoscale()
    #plots the inner triangle.
     plt.plot([x,u3, x + L3 * np.cos(theta),x ],[y, y + L2 * np.sin(gamma +
theta), y + L3 * np.sin(theta),y ])

    #plots strut 1
    plt.plot([0, x],[0, y])

    #plots strut 2
    plt.plot([x1,u2 ],[0, m2])

    #plots strut 3
    plt.plot([x2, u3],[y2, m3])


#tested Triangle plot
#triangleplotting( np.pi /4)

def secant(f, x0, x1, k):
    for i in range(1,k):
        x2 = x1 - f(x1)*(x1-x0)/(f(x1)-f(x0))
        x0 =x1
        x1 = x2
    return x2

#fig,axes = plt.subplots(2,2, figsize = 10)
plt.figure()
firsttheta = secant(f, -1, -0.6, 10)
```

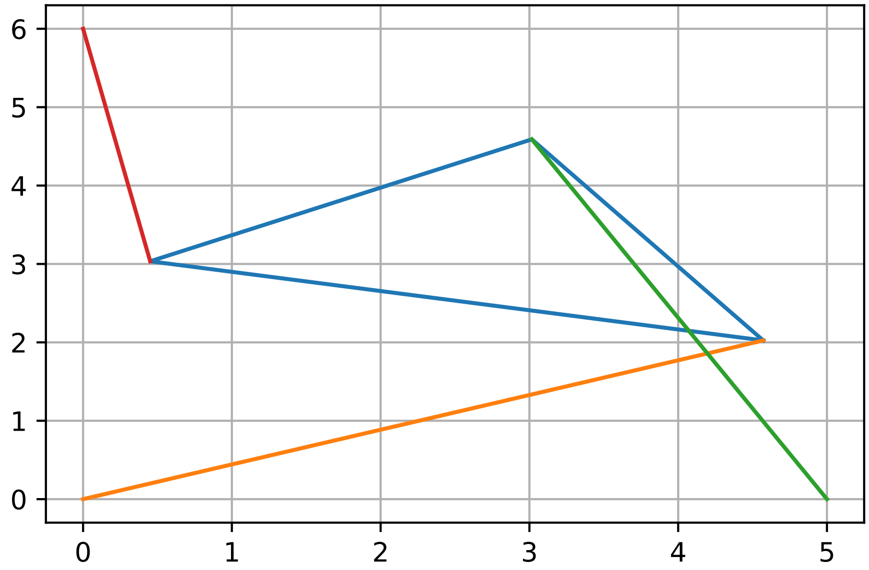
```python
triangleplotting(firsttheta)
print(firsttheta)

plt.figure()
secondtheta = secant(f, -.4, -.3, 5)
#print(secondtheta)
triangleplotting(secondtheta)
#
plt.figure()
thirdtheta = secant(f, 1, 1.2, 5)
triangleplotting(thirdtheta)
#
plt.figure()
fourththeta = secant(f, 2, 2.2, 5)
triangleplotting(fourththeta)
```

```
-0.7208492044603896
```




7
6
5
4
3
2
1
0
−1
0
1
2
3
4
5


6
5
4
3
2
1
0
0
1
2
3
4
5
6

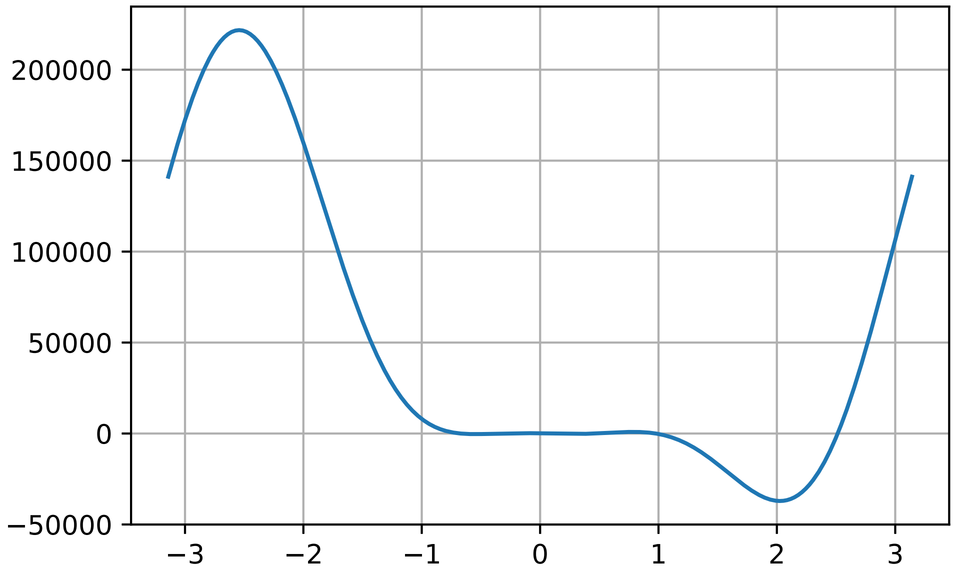


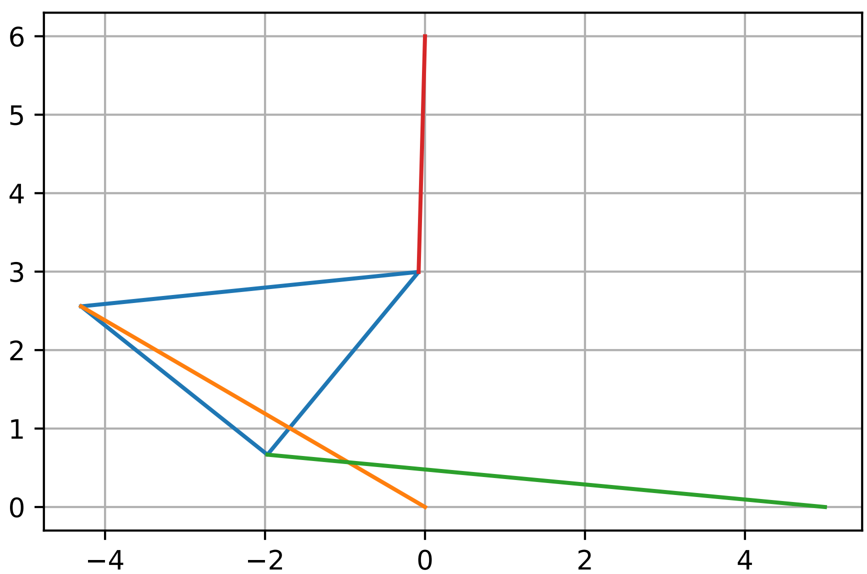
Question 5

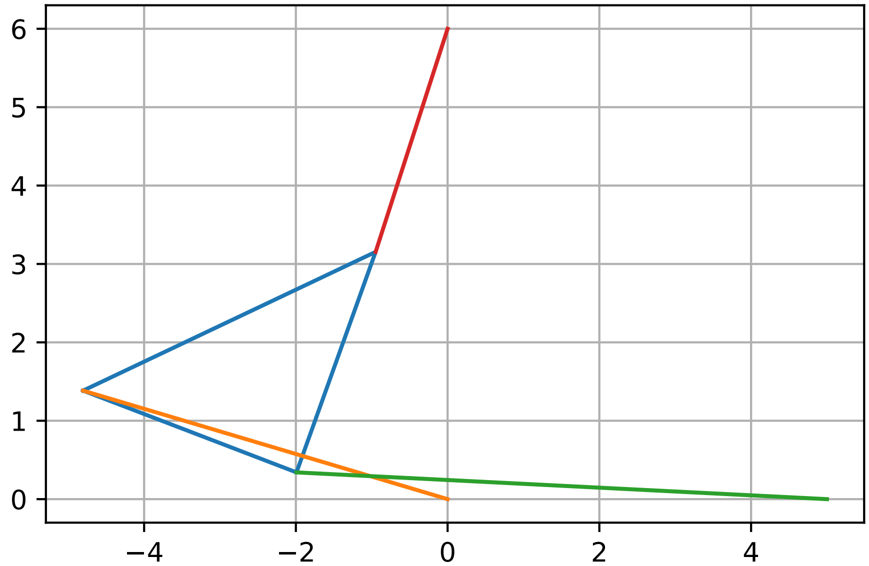
Change strut length to p2 = 7 and re-solve the problem. For these parameters, there are six poses

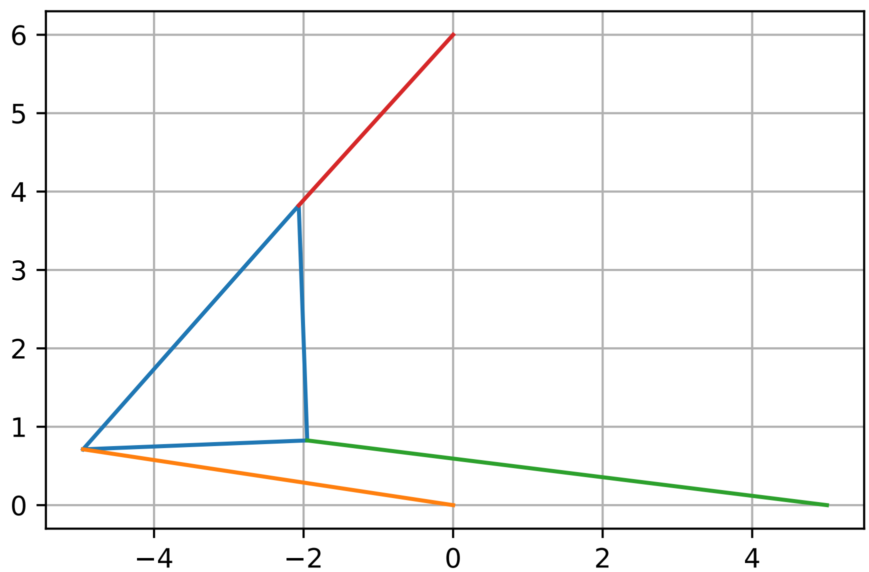
```
p2 = 7

x_array = np.linspace(-np.pi, np.pi,400)
plt.plot(x_array, f(x_array))
plt.grid()
plt.show()

g1 = secant(f,-.7, -.6, 3)
g2 = secant(f,-.4, -.3, 3)
g3 = secant(f,0, .1, 3)
g4 = secant(f,.2, .5, 3)
g5 = secant(f,.9, 1.1, 3)
g6 = secant(f,2.3, 2.6, 3)

plt.figure()
triangleplotting(g1)
plt.figure()
triangleplotting(g2)
plt.figure()
triangleplotting(g3)
plt.figure()
triangleplotting(g4)
plt.figure()
triangleplotting(g5)
```

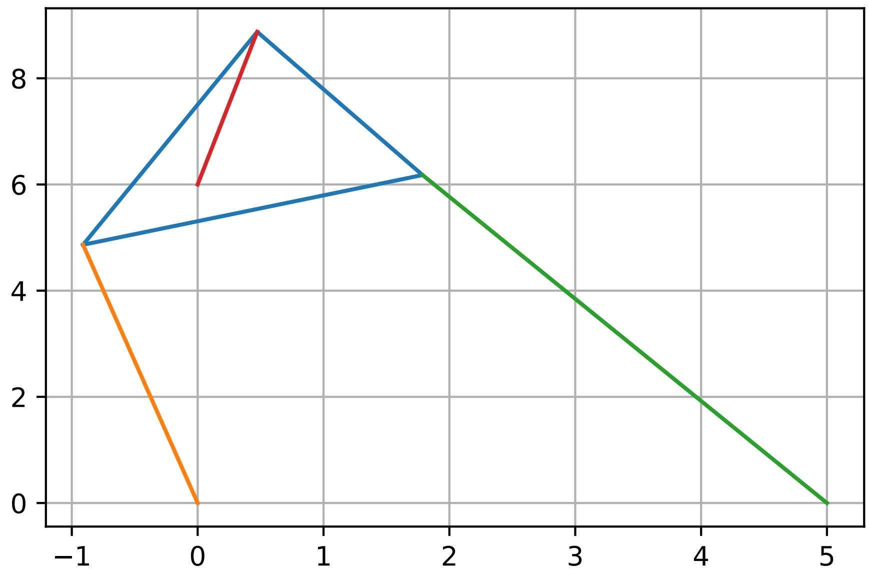
```
plt.figure()
triangleplotting(g6)
```

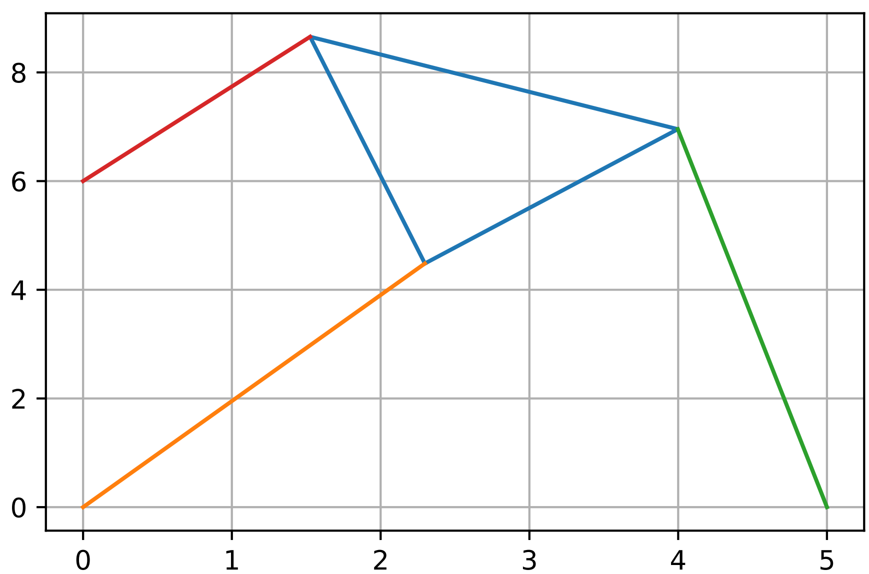


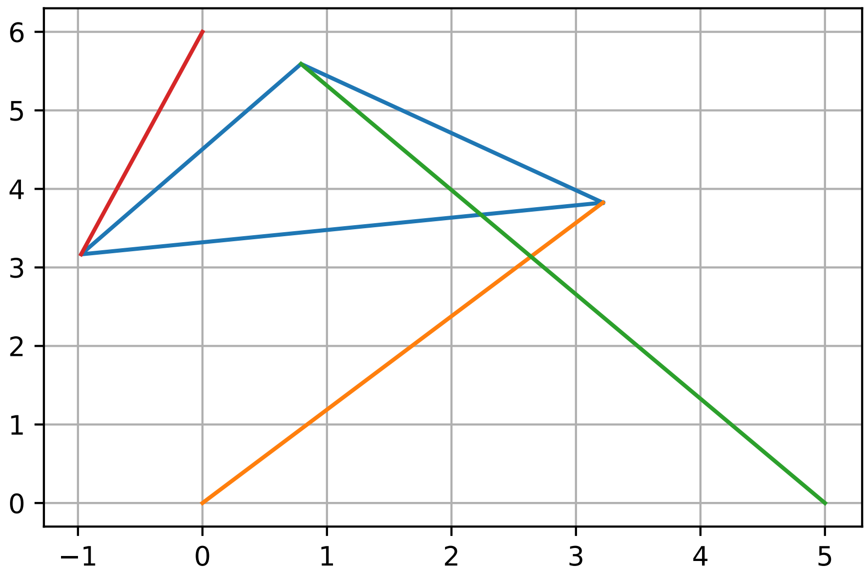



6
5
4
3
2
1
0
−4
−2
0
2
4


6
5
4
3
2
1
0
−4
−2
0
2
4


8
6
4
2
0
−1
0
1
2
3
4
5

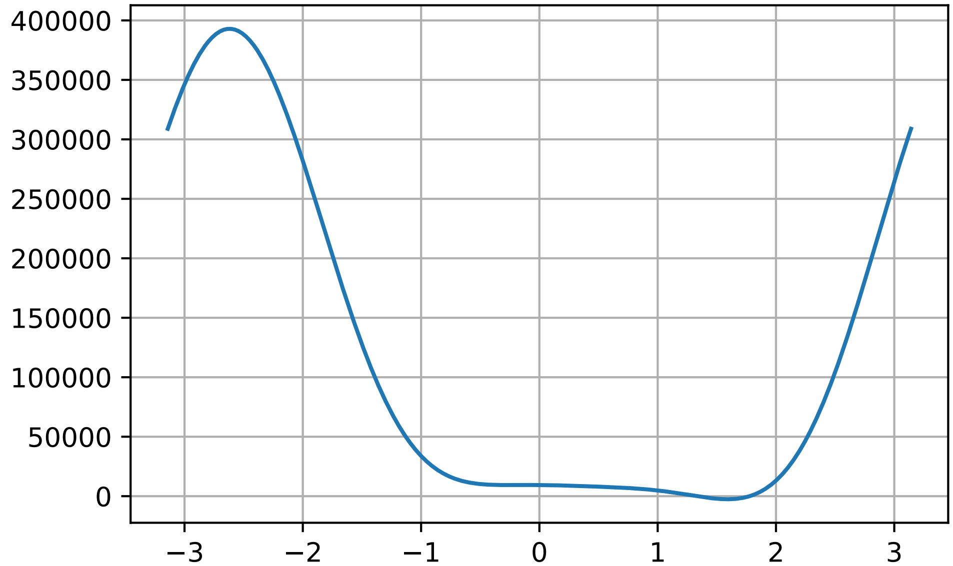

8
6
4
2
0
0
1
2
3
4
5



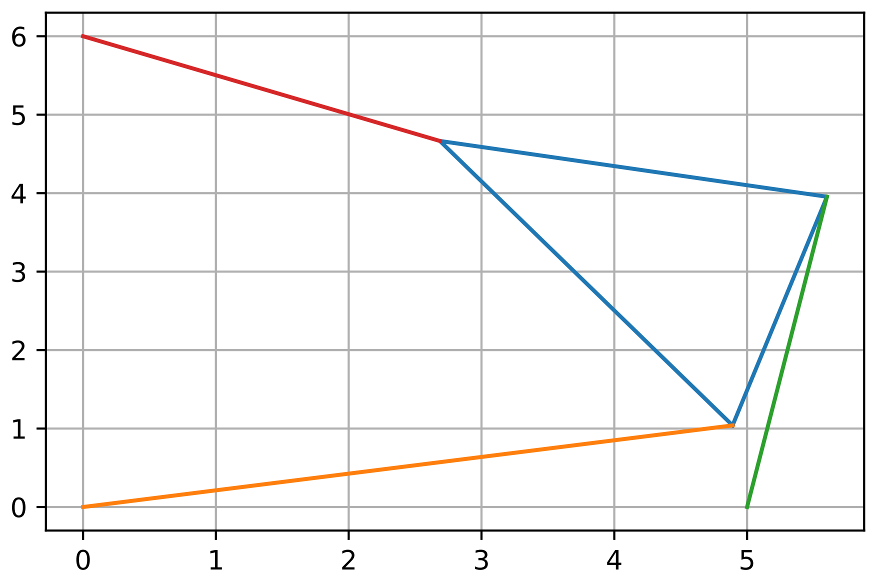
Question 6

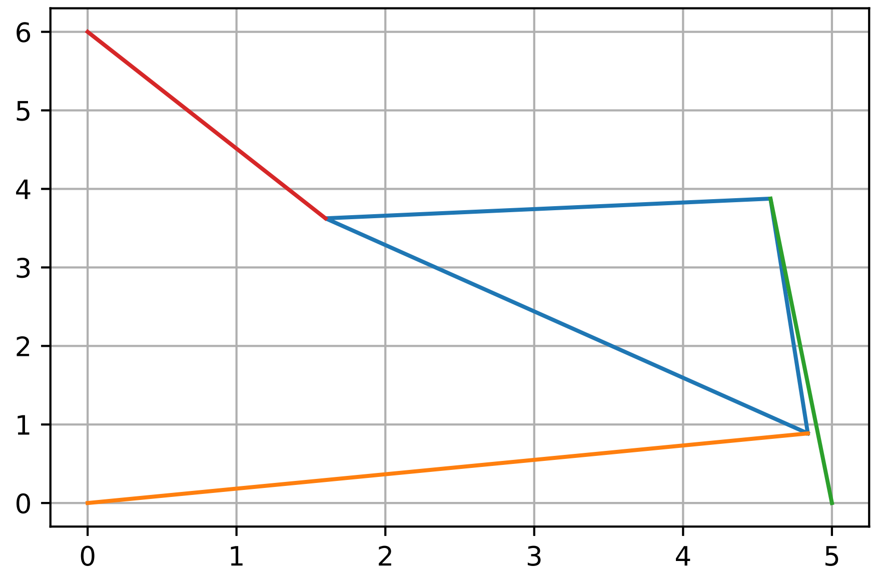
Find a strut length p2, with the rest of the parameters as in Step 4, for which there are only two poses.

```
p2 = 4

x_array = np.linspace(-np.pi, np.pi,400)
plt.plot(x_array, f(x_array))
plt.grid()
plt.show()

g1 = secant(f,1,1.5, 5)
print(g1)
print(g2)
g2 = secant(f, 1.5, 2, 5)

plt.figure()
triangleplotting(g1)
plt.figure()
triangleplotting(g2)
```




```
1.3316422307587437
-0.3550918365143094
```





Question 7

Calculate the intervals in p2, with the rest of the parameters as in Step 4, for which there are 0, 2, 4, and 6 poses, respectively

```
p2 = 0
p2_f = 10
d_p2 =  0.01

# Creating an interval to define where there are exactly 4 roots in our f
function.

Theta = np.linspace(-np.pi, np.pi, 1000)
zeros_prev= 0
while p2< p2_f:
    zeros = 0
    function = f(Theta)
    value1= 1
    for value in function:
       if value1*value < 0:
           zeros += 1
       value1 = value
    if zeros != zeros_prev:
       print(p2)

    zeros_prev = zeros
    p2 += d_p2
```

```

```

```
3.7199999999999647
4.869999999999941
6.969999999999896
7.0299999999998946
7.849999999999877
9.269999999999847
```