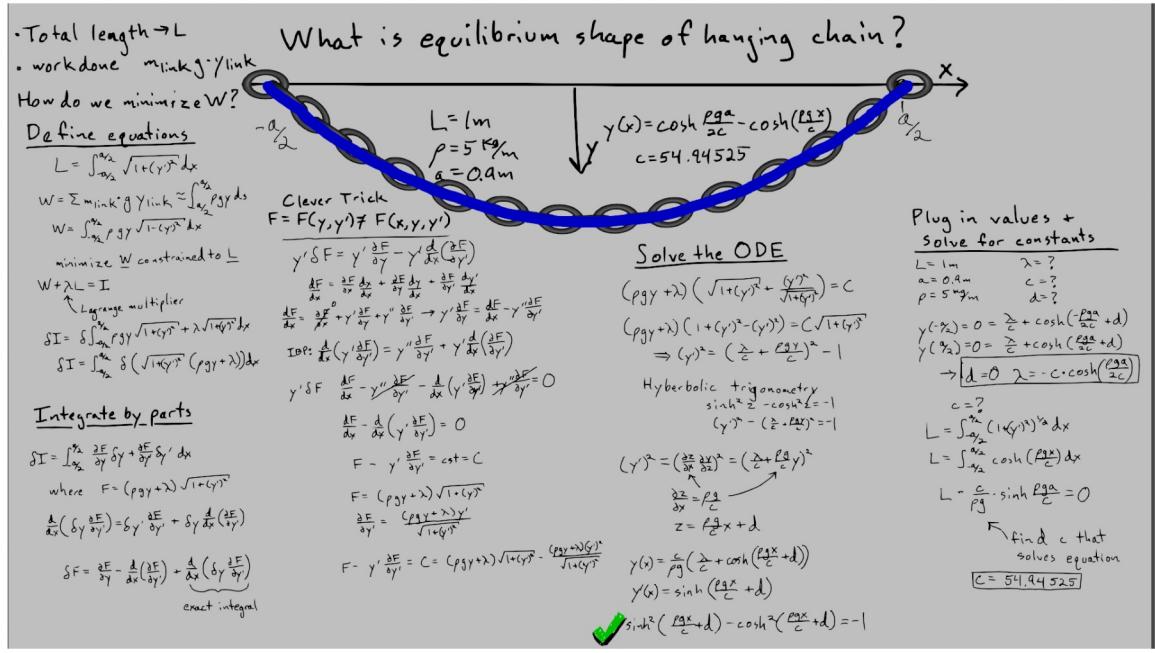


HW_03 - Nonlinear solutions



Creating a solution for the hanging chain, we reached a point where the constants required a nonlinear solution to an algebraic equation,

```
In [ ]: import numpy as np
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
```

$$1. y(x) = \cosh \frac{\rho g a}{2c} - \cosh \frac{\rho g x}{c}$$

$$2. L = \int_{-a/2}^{a/2} \cosh \frac{\rho g x}{c} dx \rightarrow L = \frac{c}{\rho g} \sinh \frac{\rho g a}{c}$$

The second equation does not have an "analytical" solution. Where "analytical" refers to an equation with separable input/output. What you need is a "numerical" solution to equation 2:

what c will satisfy this equation?

$$f(c) = L - \frac{c}{\rho g} \sinh \frac{\rho g a}{c} = 0$$

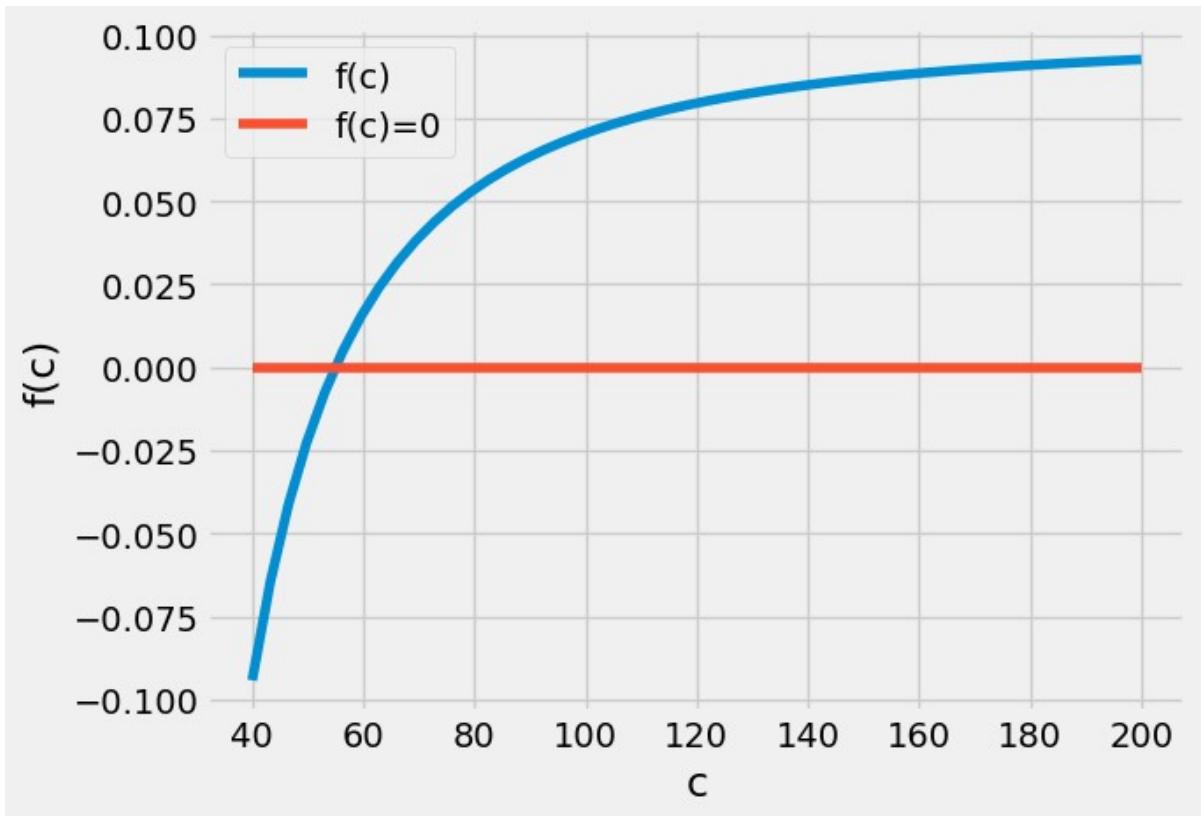
These problems often come up when engineering systems have large displacements or large rotations that cannot be ignored. One way to approach this problem is to *guess* the solution. You could try:

c=	f(c)
c=0	$-\infty$
c=50	-0.022
c=100	0.07
c=200	0.09

If you happen to guess numbers that change the sign of $f(c)$, then you know one interval where $f(c_{solution}) = 0$ must have been true. I find it helps to plot the function to see where the solution may exist

```
In [ ]: g = 9.81
L = 1
a = 0.9
rho = 5
F = lambda c: L-c/rho/g*np.sinh(rho*g*a/c)
c = np.linspace(40,200)
plt.plot(c,F(c), label='f(c)')
plt.plot(c,np.zeros(c.shape), label='f(c)=0')
plt.legend()
plt.xlabel('c')
plt.ylabel('f(c)')

Out[ ]: Text(0, 0.5, 'f(c)')
```



Numerical solution

We can use `fsolve` to automate the guess-and-check method. You need 2 things:

1. a function $f(c)$ that returns the result $f(c) = L - \frac{c}{pg} \sinh \frac{\rho ga}{c} = 0$
2. an initial guess, c_0

Numerical solutions always require an initial guess for the solution and they will iterate until your function $f(c_{\text{sol}}) \approx 0$.

Note: `fsolve` has more advanced features than 'guess-and-check', but at its core it uses algorithms to reduce the number of guesses and checks.

Define $f(c)$ with `lambda`

In Python, you can use the `lambda` function to create functions in one line. The other way to create a function is using `def`.

Note: `def` is a much richer way to create functions in Python. We will use it later when we want more involved functions.

Here, you define the function $f(c)$ with `lambda`:

```
In [ ]: g = 9.81
L = 1
a = 0.7
rho = 5

f = lambda c: L-c/rho/g*np.sinh(rho*g*a/c)
```

```
In [ ]: f(40)
```

```
Out[ ]: 0.210816148302033
```

Solve $f(c_{\text{sol}})=0$ with fsolve

The numerical solver, `fsolve`, is part of the `scipy.optimize` library. Import the function with the `from ... import`-command.

```
In [ ]: from scipy.optimize import fsolve
```

Now, you can solve for the value of `c_sol` that creates a solution to $f(c_{\text{sol}})=0$. Use the function, `f` and an initial guess, $c0=40$.

```
In [ ]: c0 = 40
c_sol = fsolve(f, c0)

print('c_sol = {} and f(c_sol) = {}'.format(c_sol[0], f(c_sol)))
```

```
c_sol = 22.67152264101508 and f(c_sol) = [1.56541446e-14]
```

Plug into catenary equation

Now, you have a solution for c that describes the hanging chain. Plug it into the original equation

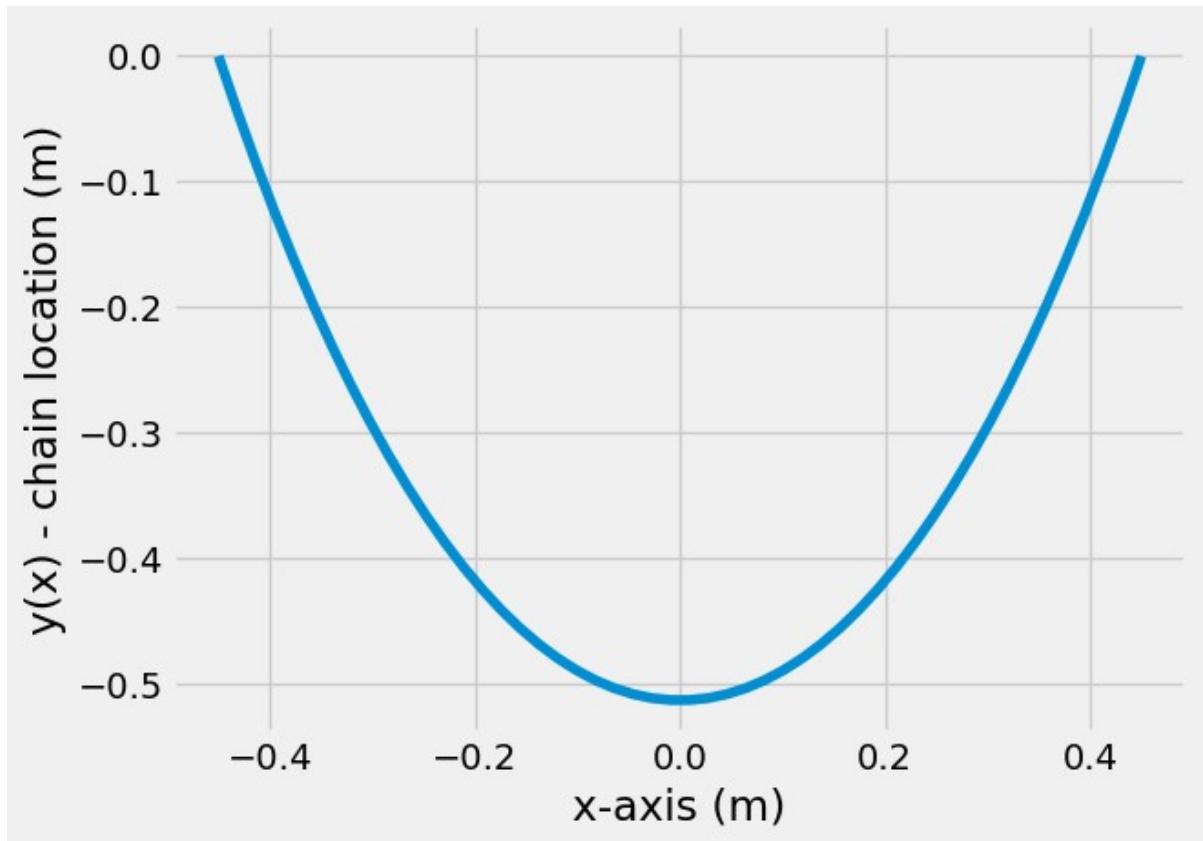
$$1. y(x) = \cosh \frac{\rho g a}{2c} - \cosh \frac{\rho g x}{c}$$

and plot the final shape.

```
In [ ]: x = np.linspace(-0.9/2, 0.9/2)
y = np.cosh(9.81*5*0.9/2/c_sol[0])-np.cosh(9.81*5*x/c_sol[0])
```

```
In [ ]: plt.plot(x,-y)
plt.xlabel('x-axis (m)')
plt.ylabel('y(x) - chain location (m)')
```

```
Out[ ]: Text(0, 0.5, 'y(x) - chain location (m)')
```



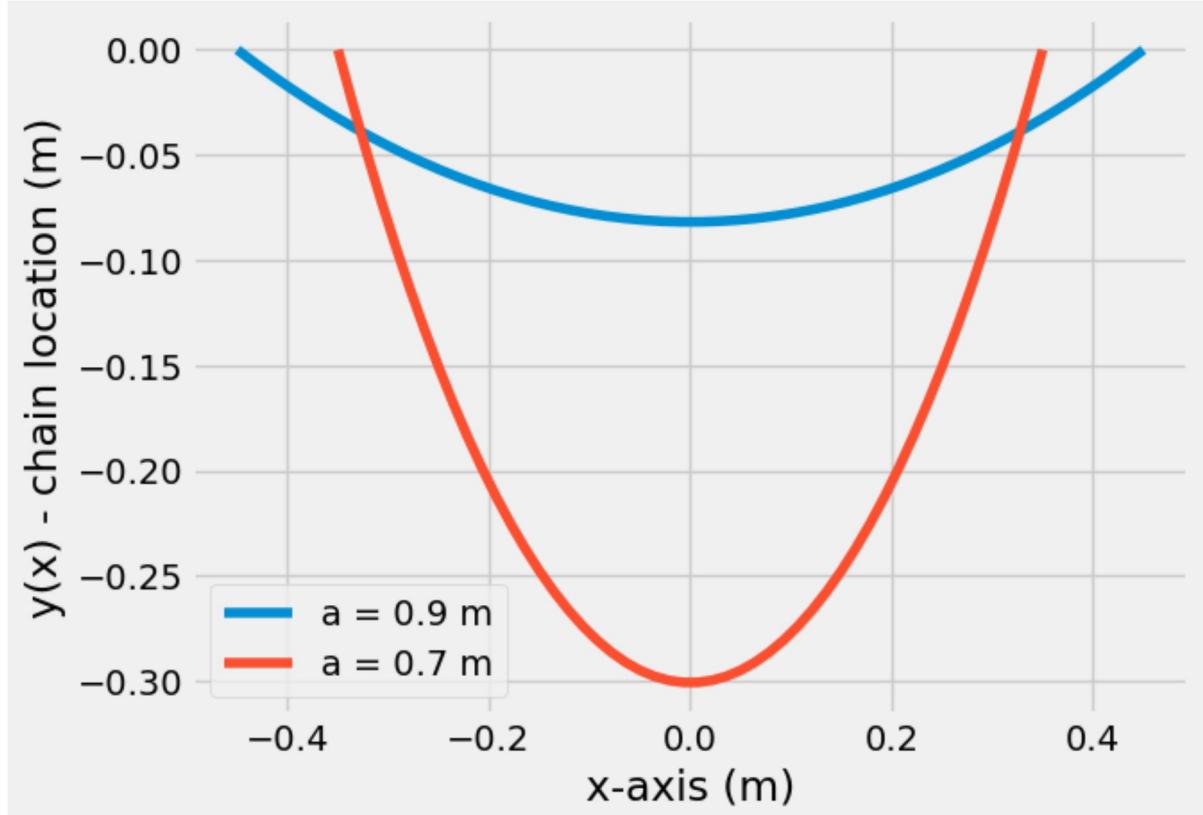
Problem 1

Plot the solution for two hanging chains, the same as we did above:

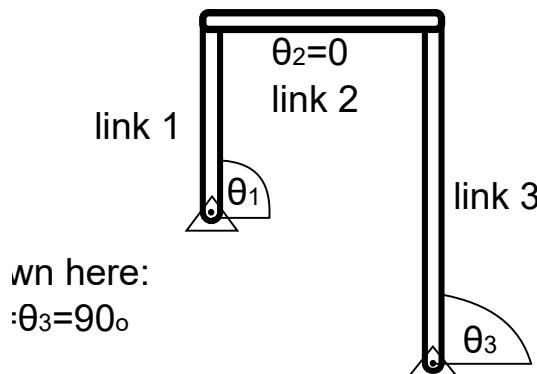
$$g = 9.81 \text{ m/s/s} \quad L = 1 \text{ m} \quad \rho = 5 \text{ kg/m}$$

1. $a = 0.9 \text{ m}$
2. $a = 0.7 \text{ m}$

```
In [ ]: a_1 = .9 #m  
a_2 = .7 #m  
g = 9.81  
L = 1  
rho = 5  
  
f_1 = lambda c: L-c/rho/g*np.sinh(rho*g*a_1/c)  
f_2 = lambda c: L-c/rho/g*np.sinh(rho*g*a_2/c)  
  
c0 = 40  
c_sol1 = fsolve(f_1, c0)  
c_sol2 = fsolve(f_2, c0)  
  
x_1 = np.linspace(-a_1/2,a_1/2)  
y_1 = np.cosh(9.81*5*a_1/2/c_sol1[0])-np.cosh(9.81*5*x_1/c_sol1[0])  
  
x_2 = np.linspace(-a_2/2,a_2/2)  
y_2 = np.cosh(9.81*5*a_2/2/c_sol2[0])-np.cosh(9.81*5*x_2/c_sol2[0])  
  
plt.plot(x_1,-y_1, label='a = 0.9 m')  
plt.plot(x_2,-y_2, label='a = 0.7 m')  
plt.xlabel('x-axis (m)')  
plt.ylabel('y(x) - chain location (m)')  
plt.legend();
```



Problem 2



wn here:

$$\theta_3 = 90^\circ$$

In the four-bar linkage show above there are 3 bodies moving in 2D (9 DOF) and 4 pins (8 constraints). The linkage configuration is constrained by the two nonlinear equations

1. $l_1 \sin \theta_1 + l_2 \sin \theta_2 - l_3 \sin \theta_3 - d_y = 0$
2. $l_1 \cos \theta_1 + l_2 \cos \theta_2 - l_3 \cos \theta_3 - d_x = 0$

If you have one of the angles, θ_1 , you can use equations 1 and 2 to solve for the other two angles, θ_2 and θ_3 using `fsolve` only now the input is a vector with two values and the output is a vector with two values.

$$\bar{f}(\bar{x}) = \begin{bmatrix} f_1(\theta_2, \theta_3) \\ f_2(\theta_2, \theta_3) \end{bmatrix} = \begin{bmatrix} l_1 \sin \theta_1 + l_2 \sin \theta_2 - l_3 \sin \theta_3 - d_y \\ l_1 \cos \theta_1 + l_2 \cos \theta_2 - l_3 \cos \theta_3 - d_x \end{bmatrix}$$

The linkage system has the following properties:

- link 1: $l_1 = 0.5 \text{ m}$
- link 2: $l_2 = 1 \text{ m}$
- link 3: $l_3 = 1 \text{ m}$

when $\theta_1 = 90^\circ$, $\theta_2 = 0^\circ$, and $\theta_3 = 90^\circ$. So the two grounded pins have a fixed relative position, $r_{3/1} = d_x \hat{i} + d_y \hat{j} = 1 \hat{i} - 0.5 \hat{j}$.

Below, the definition of `Fbar` is defined for $\bar{f}(\bar{x})$ and the function is satisfied for $\theta_1 = \theta_3 = 90^\circ$ and $\theta_2 = 0^\circ$. Then, the links are plotted with `rx` and `ry`, where

$$\bullet rx = \begin{bmatrix} 0 \\ l_1 \cos(\theta_1) \\ l_1 \cos(\theta_1) + l_2 \cos(\theta_2) \\ l_1 \cos(\theta_1) + l_2 \cos(\theta_2) - l_3 \cos(\theta_3) \end{bmatrix}$$

$$\bullet ry = \begin{bmatrix} 0 \\ l_1 \sin(\theta_1) \\ l_1 \sin(\theta_1) + l_2 \sin(\theta_2) \\ l_1 \sin(\theta_1) + l_2 \sin(\theta_2) - l_3 \sin(\theta_3) \end{bmatrix}$$

```
In [ ]: l1 = 0.5
        l2 = 1
        l3 = 1
        a1 = np.pi/2
        dy = -0.5
        dx = 1
        Fbar = lambda x: np.array([l1*np.sin(a1)+l2*np.sin(x[0])-l3*np.sin(x[1])-dy,
                                    l1*np.cos(a1)+l2*np.cos(x[0])-l3*np.cos(x[1])-dx])
```

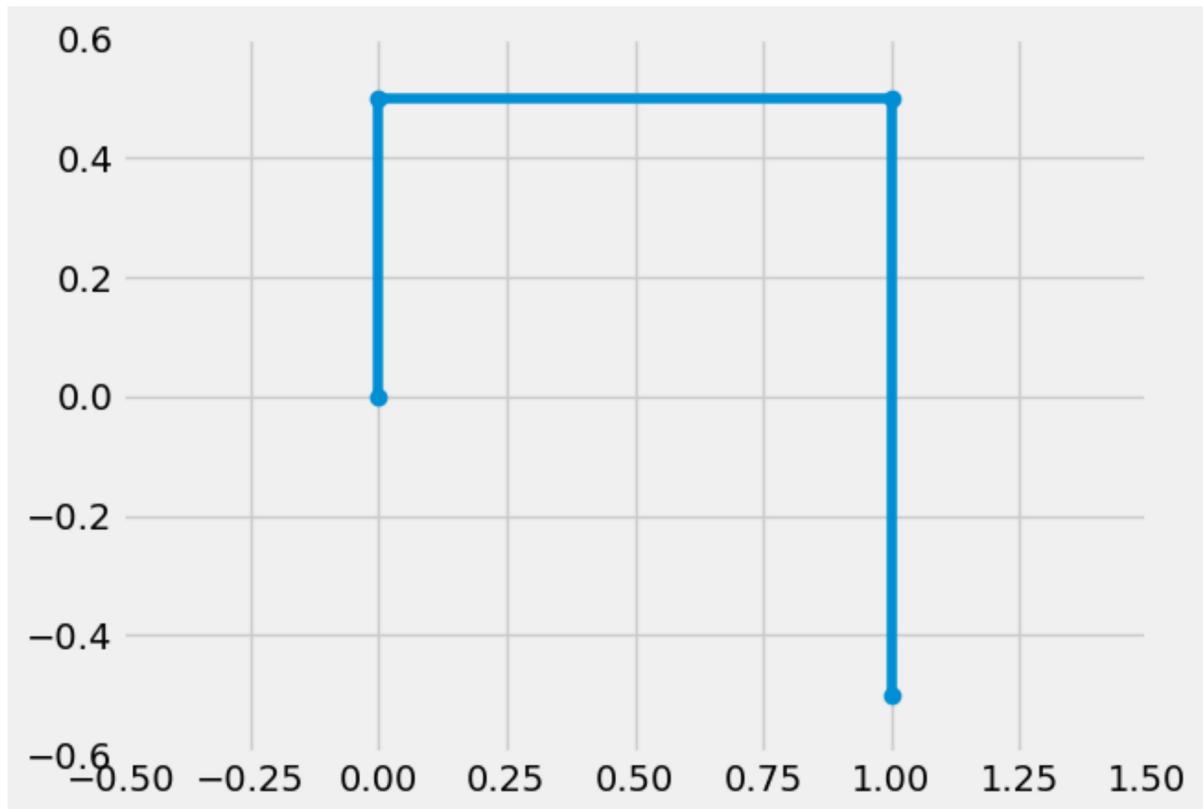
```
In [ ]: x90 = np.array([0,np.pi/2])
        Fbar(x90)
```

```
Out[ ]: array([ 0.00000000e+00, -1.11022302e-16])
```

```
In [ ]: rx = np.array([0,
                      l1*np.cos(a1),
                      l1*np.cos(a1)+l2*np.cos(x90[0]),
                      l1*np.cos(a1)+l2*np.cos(x90[0])-l3*np.cos(x90[1]))]
ry = np.array([0,
              l1*np.sin(a1),
              l1*np.sin(a1)+l2*np.sin(x90[0]),
              l1*np.sin(a1)+l2*np.sin(x90[0])-l3*np.sin(x90[1]))]

plt.plot(rx,ry, 'o-')
plt.axis([-0.5, 1.5, -0.6, 0.6])
```

Out[]: (-0.5, 1.5, -0.6, 0.6)



Your goal:

Change the angle to $\theta_1 = 45^\circ$, 135° , and 180° . Plot the three configurations like above. Use `fsolve` to find θ_2 and θ_3 .

```
In [ ]: x45 = np.array([0,np.pi/4])
x135 = np.array([0,135*np.pi/180])
x180 = np.array([0,np.pi])

a_45 = np.pi/4
Fbar_45 = lambda x: np.array([l1*np.sin(a_45)+l2*np.sin(x[0])-l3*np.sin(x[1])-dy,
                               l1*np.cos(a_45)+l2*np.cos(x[0])-l3*np.cos(x[1])-dx])

a_135 = 135*np.pi/180
Fbar_135 = lambda x: np.array([l1*np.sin(a_135)+l2*np.sin(x[0])-l3*np.sin(x[1])-dy,
                                l1*np.cos(a_135)+l2*np.cos(x[0])-l3*np.cos(x[1])-dx])

a_180 = np.pi
Fbar_180 = lambda x: np.array([l1*np.sin(a_180)+l2*np.sin(x[0])-l3*np.sin(x[1])-dy,
                                l1*np.cos(a_180)+l2*np.cos(x[0])-l3*np.cos(x[1])-dx])

theta_45 = fsolve(Fbar_45,x45)
theta_135 = fsolve(Fbar_135,x135)
theta_180 = fsolve(Fbar_180,x180)
```

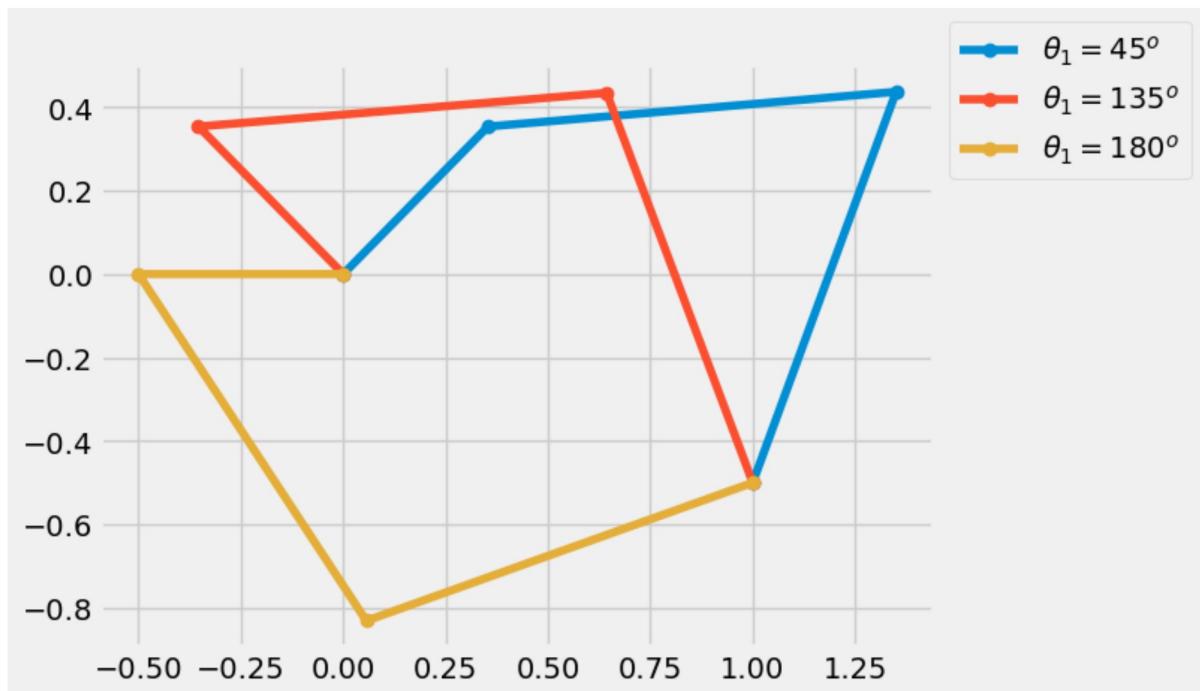
```
In [ ]: rx_45 = np.array([0,
                      11*np.cos(a_45),
                      11*np.cos(a_45)+12*np.cos(theta_45[0]),
                      11*np.cos(a_45)+12*np.cos(theta_45[0])-13*np.cos(theta_45[1]))]
ry_45 = np.array([0,
                  11*np.sin(a_45),
                  11*np.sin(a_45)+12*np.sin(theta_45[0]),
                  11*np.sin(a_45)+12*np.sin(theta_45[0])-13*np.sin(theta_45[1]))]

rx_135 = np.array([0,
                   11*np.cos(a_135),
                   11*np.cos(a_135)+12*np.cos(theta_135[0]),
                   11*np.cos(a_135)+12*np.cos(theta_135[0])-13*np.cos(theta_135[1]))]
ry_135 = np.array([0,
                   11*np.sin(a_135),
                   11*np.sin(a_135)+12*np.sin(theta_135[0]),
                   11*np.sin(a_135)+12*np.sin(theta_135[0])-13*np.sin(theta_135[1]))]

rx_180 = np.array([0,
                   11*np.cos(a_180),
                   11*np.cos(a_180)+12*np.cos(theta_180[0]),
                   11*np.cos(a_180)+12*np.cos(theta_180[0])-13*np.cos(theta_180[1]))]
ry_180 = np.array([0,
                   11*np.sin(a_180),
                   11*np.sin(a_180)+12*np.sin(theta_180[0]),
                   11*np.sin(a_180)+12*np.sin(theta_180[0])-13*np.sin(theta_180[1]))]

plt.plot(rx_45,ry_45,'o-',label=r'$\theta_1=45^{\circ}$')
plt.plot(rx_135,ry_135,'o-',label=r'$\theta_1=135^{\circ}$')
plt.plot(rx_180,ry_180,'o-',label=r'$\theta_1=180^{\circ}$')
# plt.axis([-0.5, 1.5, -0.6, 0.6])
plt.legend(bbox_to_anchor = [1,1.1])
```

```
Out[ ]: <matplotlib.legend.Legend at 0x148ec0d2d90>
```



In []: