
Math 105: Trigonometry and *fminsearch*

Objectives

- Relate $\sin()$ and $\cos()$ to unit circles
- Convert from rectangular to polar coordinates
- Calculate the position of a robotic arm (forward kinematics)
- Calculate the angles of a robotic arm (inverse kinematics)
- Use the Matlab function `fminsearch()`

Introduction

From Wikipedia,

Trigonometry (from Greek trigonon, "triangle" and metron, "measure"[1]) is a branch of mathematics that studies relationships involving lengths and angles of triangles. The field emerged in the Hellenistic world during the 3rd century BC from applications of geometry to astronomical studies. www.Wikipedia.com

Trigonometry is fundamental to electrical and computer engineering.

- Power is transmitted as a 60Hz sine wave
- Analysis of filters, such as the bass boost on your stereo, relies upon expressing an audio signal in terms of a sum of sinusoids,
- AC motors, such as a quad-copter motor, are driven by sinusoidal signals where the frequency of the sine wave determines the speed of the motor
- Analysis of systems described by differential equations (read: everything) depends upon being able to use complex numbers - which have their origin in $\sin()$ and $\cos()$ functions.

Likewise, trigonometry may seem like an archaic topic which deals only with architecture and triangles. Actually, it's much more than that.

$\sin()$, $\cos()$, $\tan()$

Trigonometry is the study of the unit circle. If you draw a unit circle and take a point an angle θ from the x-axis, then

- The x-coordinate of that point is $\cos(\theta)$
- The y-coordinate of that point is $\sin(\theta)$
- If you extend the line from the origin to the point on the unit circle to $x=1$, the length of the line to the x-axis is $\tan(\theta)$

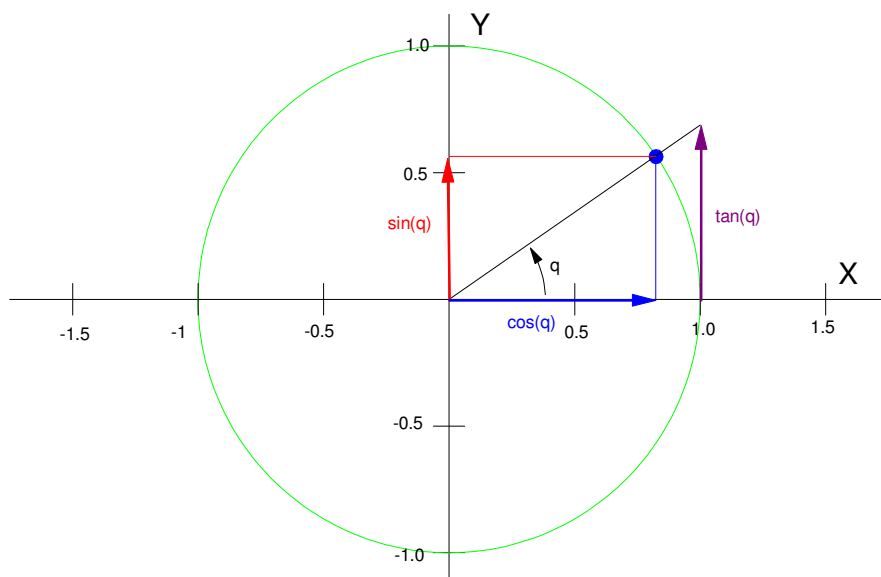


Figure 1: $\sin()$, $\cos()$, and $\tan()$ are all related to the unit circuit

It you let the angle increase with time as

$$\theta = \omega t$$

then you get a sine wave. In Matlab:

```
>> t = [0:0.01:6]';  
>> w = 1;  
>> x = cos(w*t);  
>> y = sin(w*t);  
>> plot(t,x,t,y)  
>> xlabel('Time (seconds)');
```

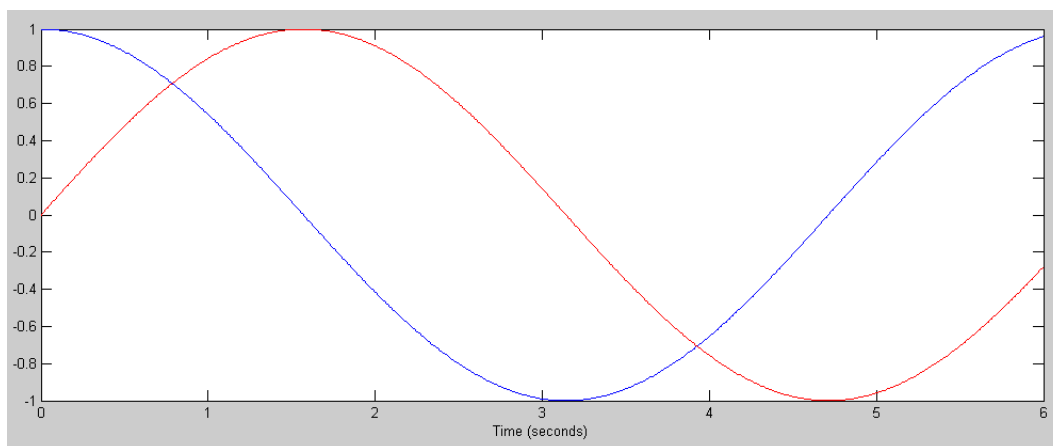


Figure 2: 1 rad/sec sine wave: $\cos(t)$ (blue) and $\sin(t)$ (red)

Note that

- $\cos()$ and $\sin()$ go between -1 and +1. This isn't surprising since these are just the x and y coordinates as you go around the unit circle.

- The period of $\cos()$ and $\sin()$ is 2π (6.28 seconds).

The default units for $\cos()$ and $\sin()$ is radians. If you want to use degrees, the conversion is

$$360 \text{ degrees} = 2\pi \text{ radians}$$

$$1 \frac{\text{cycle}}{\text{second}} = 1 \text{ Hz} = 2\pi \frac{\text{rad}}{\text{sec}}$$

Pretty much, anything English isn't natural. You'll find in engineering that the math works out a lot nicer if you use natural units - such as radians.

If you increase the frequency, you get a sine wave that is quicker. A 1Hz sine wave $\left(2\pi \frac{\text{rad}}{\text{sec}}\right)$ looks like the following:

```
>> w = 2*pi;
>> x = cos(w*t);
>> y = sin(w*t);
>> plot(t,x,'b',t,y,'r')
>> xlabel('Time (seconds)');
```

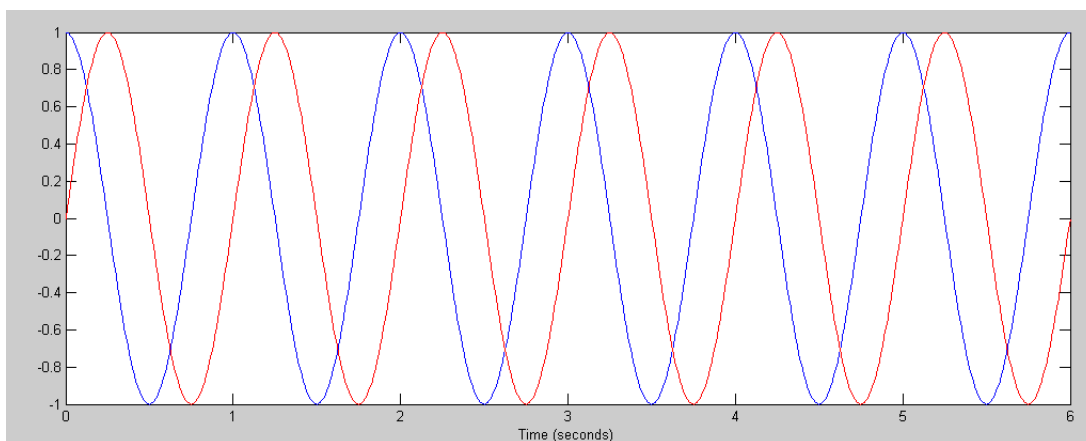


Figure 3: 1Hz Sine Wave: $\cos(6.28t)$ (blue) and $\sin(6.28t)$ (red)

Amplitude, Frequency, Phase

A generalized sine wave can be written as

$$y(t) = a \cos(\omega t) + b \sin(\omega t)$$

or

$$y(t) = r \cos(\omega t + \theta)$$

Here

- r is the amplitude
- ω is the frequency in rad/sec, and
- θ is the phase shift, also in radians.

The relationship between rectangular and polar form is

$$r^2 = a^2 + b^2$$

$$\tan(\theta) = \frac{b}{a}$$

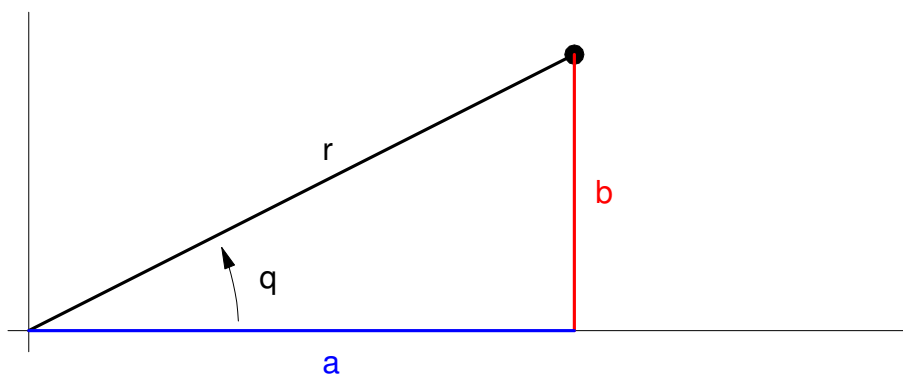


Figure 4: Conversion from rectangular form and polar form for sinusoids

For example,

$$y = 5 \cos(6t - 1)$$

looks like the following:

```
>> t = [0:1/250:1]';
>> y = 5*cos(6*t-1);
>> plot(t,y);
```

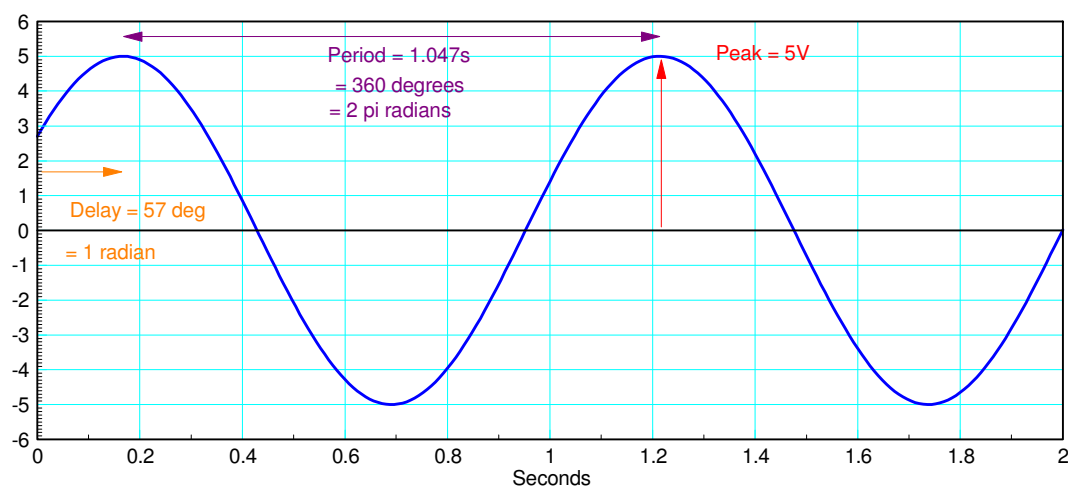


Figure 5: Plot of $y(t) = 5 \cos(6t - 1)$

The peak is 5 Volts

The frequency is 6 rad/sec, meaning the period is

- $period = 6T = 2\pi$
- $T = \frac{2\pi}{6} = 1.047 \text{ sec}$

The phase shift is 1 radian, meaning

- The delay is $\left(\frac{1 \text{ radian}}{6 \text{ rad/sec}}\right) = \frac{1}{6} \text{ sec} = 0.166 \text{ sec}$

This is important in electrical and computer engineering:

- Given the amplitude, frequency, and phase shift, you can sketch a sine wave
- Given a sine wave, you can determine the amplitude, frequency, and phase shift.

Sine Waves and Circles

What shouldn't be surprising is that if you plot $\cos()$ vs $\sin()$ you get a circle

```
>> x = cos(w*t);  
>> y = sin(w*t);  
>> plot(x,y)
```

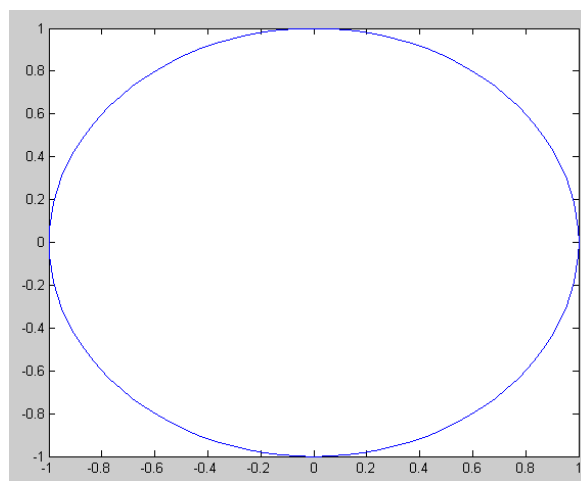


Figure 6: $\cos(t)$ vs. $\sin(t)$ produces the unit circle

It also shouldn't be surprising that

$$\cos^2(t) + \sin^2(t) = 1$$

That just says that the circle you're using has a radius of one. That's sort of the definition of $\cos()$ and $\sin()$.

Polar Coordinates

Given any point, you can express it in Cartesian coordinates with its x and y values:

$$P = (x, y)$$

You can also express this in polar form as

$$P = r\angle\theta$$

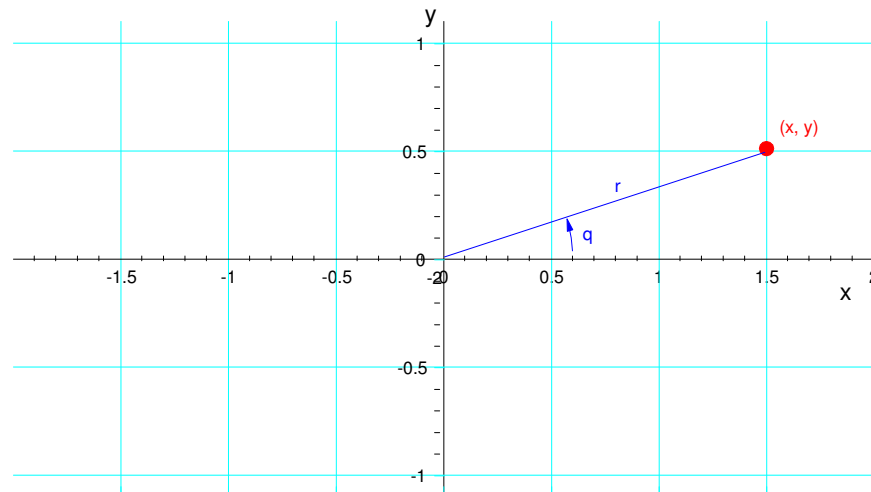


Figure 7: A point P can be expressed in Cartesian coordinates (x,y) or polar coordinates ($r\angle\theta$)

The conversion from one to the other is

$$x = r \cos \theta$$

$$y = r \sin \theta$$

or

$$r = \sqrt{x^2 + y^2}$$

$$\theta = \arctan(y, x)$$

There are two arctan() functions in Matlab

- atan(y/x) returns the angle from $-\pi/2$ to $+\pi/2$ (-90 degrees to +90 degrees)
- atan2(y, x) returns the angle

The problem with atan is that if both x and y are negative, the signs cancel. To get the actual angle, you need to use atan2()

Fun with Polar Coordinates

You can create some pretty plots using polar coordinates. The trick in Matlab is to convert these functions to cartesian coordinates since the `plot(x,y)` command is limited to plotting x vs. y.

Example 1: Circles.

Trig functions are all about circles. Not surprisingly, both `sin()` and `cos()` plot as circles

$$r = \cos \theta$$

Matlab Code:

```
q = [0:0.005:1]' * 2 * pi;  
r = cos(q);  
  
x = r .* cos(q);  
y = r .* sin(q);  
  
plot(x,y);
```

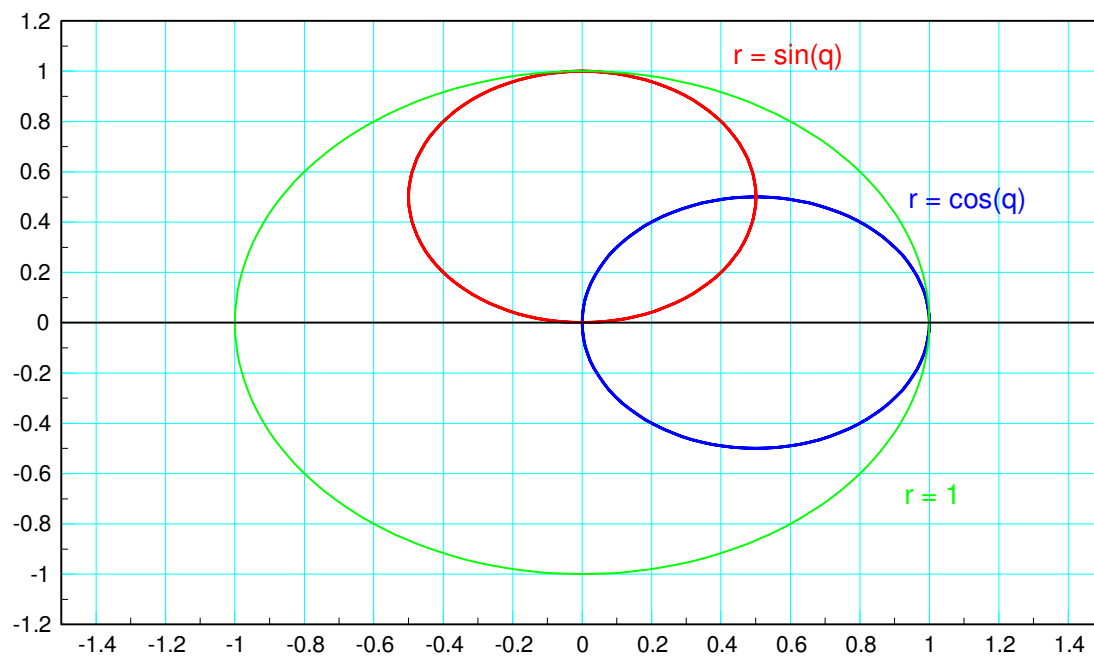


Figure 8: $r = \cos(q)$ and $r = \sin(q)$ plot as circles

Example 2: 4-Leaf Clover

$$r = \cos(2\theta)$$

Matlab Code:

```
q = [0:0.005:1]' * 2 * pi;  
r = cos(2*q);  
  
x = r .* cos(q);  
y = r .* sin(q);  
  
plot(x,y);
```

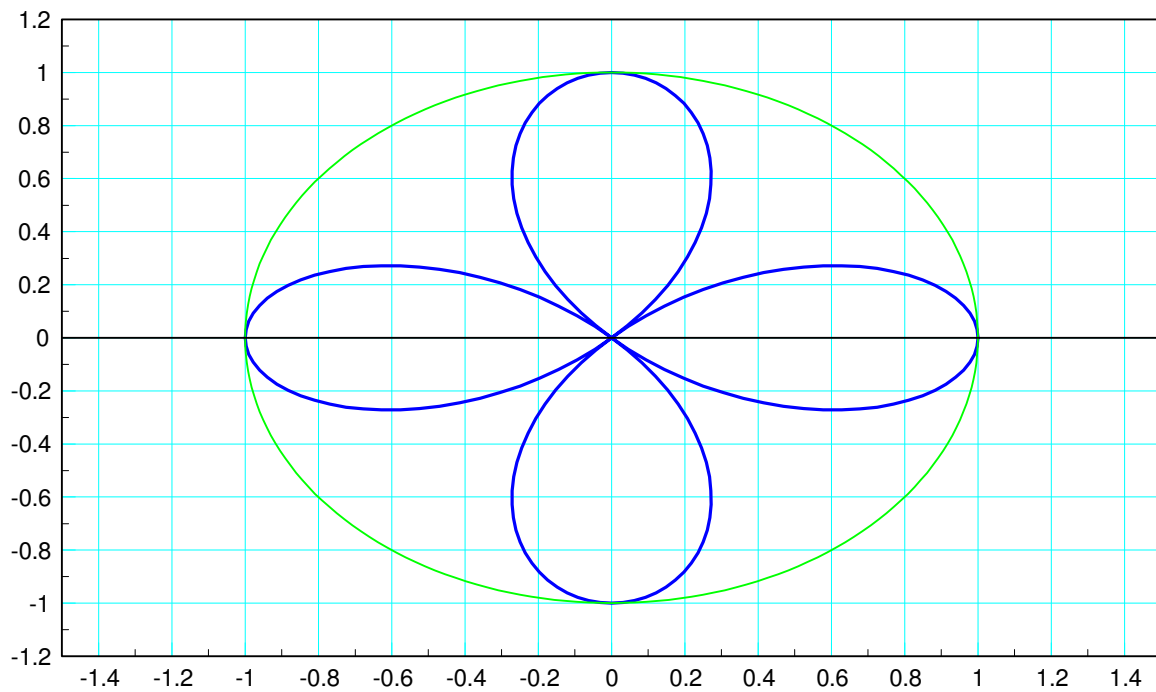


Figure 9: $r = \cos(2q)$ plots as a 4-leaf clover

Linear Spiral

$$r = \frac{1}{30} \cdot \theta$$

This plots as a spiral with equal spacing between lines

Matlab Code:

```
q = [0:0.005:5]' * 2 * pi;  
r = q/30;  
  
x = r .* cos(q);  
y = r .* sin(q);  
  
plot(x,y);
```

Matlab actually has pretty good animation. You can make this spin with the following code

```
q = [0:0.005:5]' * 2 * pi;  
  
for i=1:1000  
    dq = i/100;  
    r = q/30;  
    x = r .* cos(q+dq);  
    y = r .* sin(q+dq);  
  
    plot(x,y);  
    xlim([-1.5,1.5]);  
    ylim([-1.2,1.2]);  
    pause(0.01);  
end
```

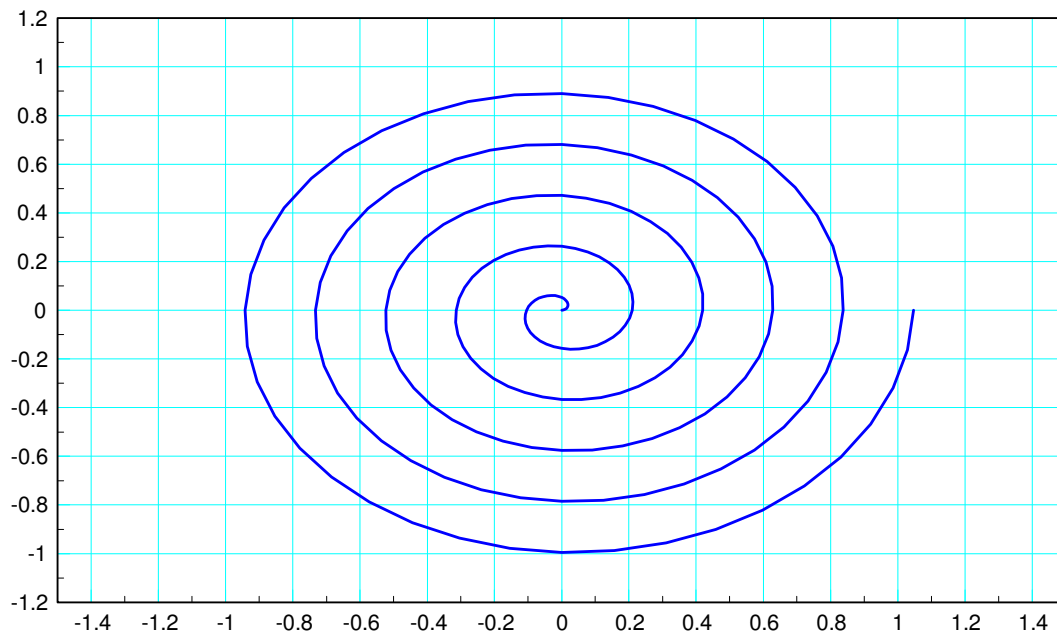


Figure 10: $r = q/30$ plots as a linear spiral

Lissajous Figures

Lissajous figures are pretty shapes you can display on oscilloscopes using two sine-wave generators:

- One driving the x-axis
- One driving the y-axis.

If the frequency of the y-axis is an integer multiple of the x-axis, you get a shape like a crown. If the frequency is slightly off, the crown rotates. Pretty much every sci-fi movie from the 50's had an oscilloscope showing one of these somewhere in the background.

For example, let y-axis be 3x the frequency of the x-axis

$$y = \sin(3\theta)$$

$$x = \cos \theta$$

gives the shape below. Add a small offset to y to make it rotate;

Matlab Code

```
q = [0:0.005:1]' * 2*pi;  
for i=1:1000  
    dq = i/100;  
    x = cos(q + dq);  
    y = sin(3*q);  
    plot(x,y);  
    pause(0.01);  
end
```

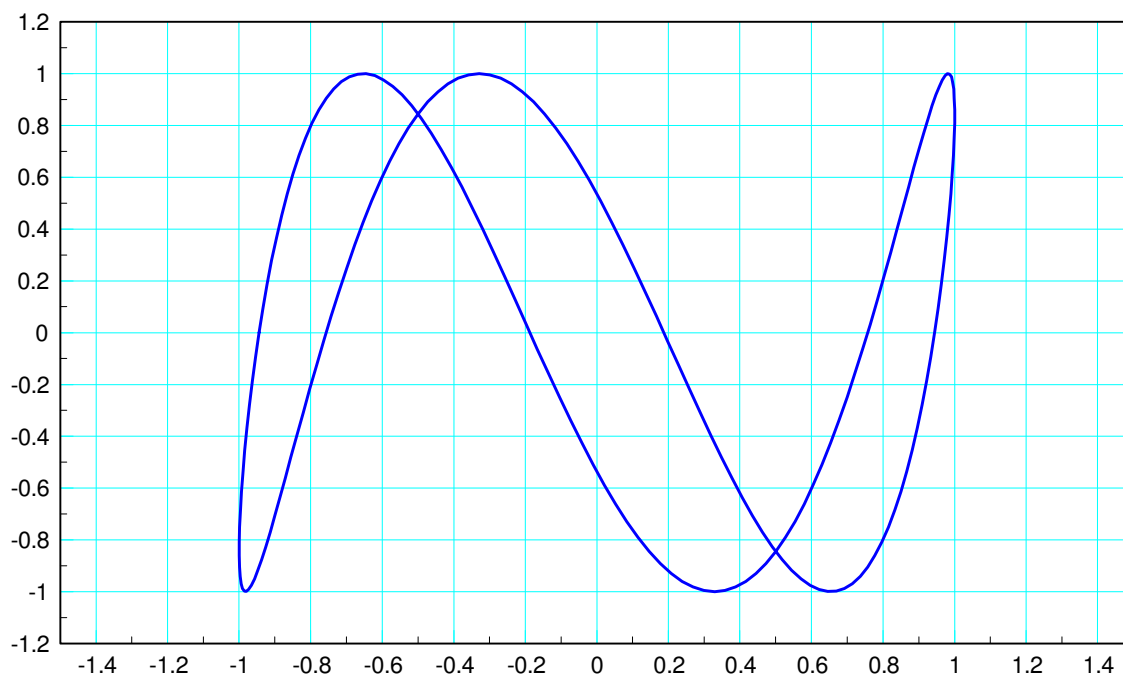


Figure 11: Lissajous Figure with the the y-axis being 3x the frequency of the x-axis

Calculations using Polar Coordinates

Another use of polar coordinates is to add vectors. The trick is to convert to rectangular form. Then, the x and y coordinates add.

For example, determine y:

$$y = 5\angle 20^\circ + 8\angle -63^\circ + 4\angle 37^\circ$$

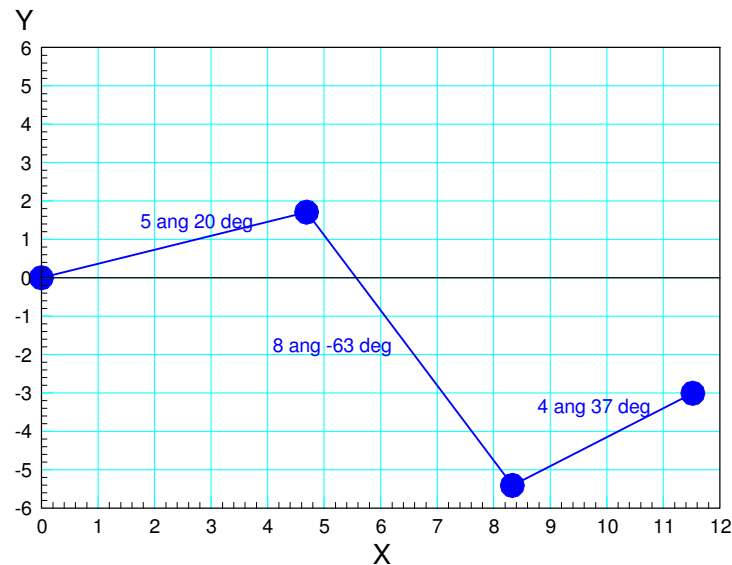


Figure 12: When adding numbers in polar form, the x-component add and the y-component adds

To determine y, calculate the x and y coordinate of each term

$$r\angle\theta = (r \cos \theta, r \sin \theta)$$

In Matlab: (note: Matlab uses radians for angles, not degrees)

```
>> x1 = 5*cos(20*pi/180)
x1 = 4.6985

>> y1 = 5*sin(20*pi/180)
y1 = 1.7101

>> x2 = 8*cos(-63*pi/180)
x2 = 3.6319

>> y2 = 8*sin(-63*pi/180)
y2 = -7.1281

>> x3 = 4*cos(37*pi/180)
x3 = 3.1945

>> y3 = 4*sin(37*pi/180)
y3 = 2.4073
```

The sum is the sum of the x-portion and the sum of the y-portion:

```
>> X = x1+x2+x3
X = 11.5249

>> Y = y1+y2+y3
Y = -3.0107
```

The sum is

$$5\angle 20^\circ + 8\angle -63^\circ + 4\angle 37^\circ = (11.5249, -3.0107)$$

Robotics: Forward Kinematics

This type of solution applies directly to robotics and the problem of forward kinematics. Forward kinematics is the problem of computing where the tip of a robotic arm is given the joint angles. For example, assume

- A 2-dimensional robot
- With three rotational joints, and
- Each link has a length of 1m.

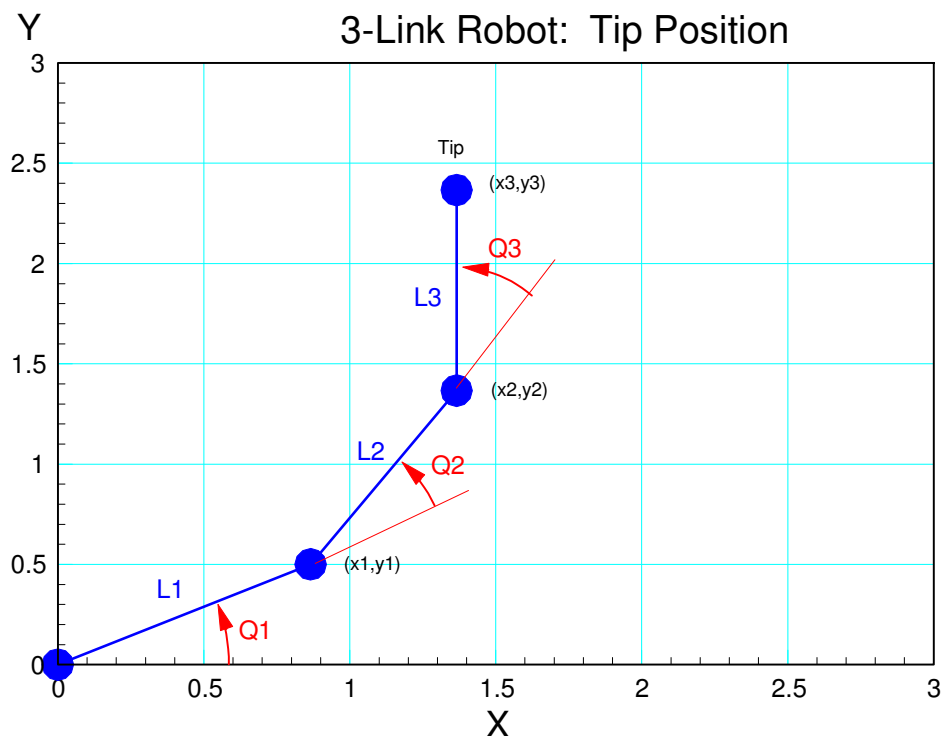


Figure 13: The position of a robotic arm is determined by adding polar numbers

Problem 1: Determine the tip position when the joint angles are {30 degrees, 40 degrees, 50 degrees}

```
function [x3, y3] = ForwardKinematics(q1, q2, q3)

q1 = q1 * pi/180;
q2 = q2 * pi/180;
q3 = q3 * pi/180;

L1 = 1;
L2 = 1;
L3 = 1;

x0 = 0;
y0 = 0;

x1 = L1*cos(q1);
y1 = L1*sin(q1);

x2 = x1 + L2*cos(q1+q2);
y2 = y1 + L2*sin(q1+q2);

x3 = x2 + L3*cos(q1+q2+q3);
y3 = y2 + L3*sin(q1+q2+q3);

plot([x0,x1,x2,x3],[y0,y1,y2,y3],'b.-');
xlim([0,3]);
ylim([0,3]);

pause(0.01);

end

>> [Tx,Ty] = ForwardKinematics(30,40,50)
Tx =    0.7080
Ty =    2.3057
```

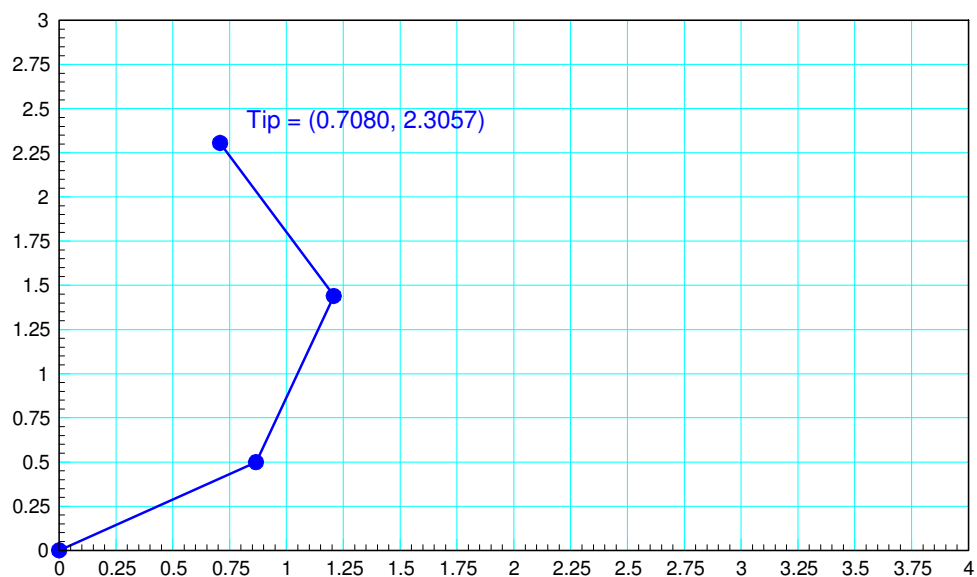


Figure 14: Position of robot with angles = {30 deg, 40 deg, 50 deg}

Problem 2: Determine the tip position when the joint angles are

- $Q1 = 30 \cdot \sin(t)$ degrees
- $Q2 = 40 \cdot \sin(2t)$ degrees
- $Q3 = 50 \cdot \sin(3t)$ degrees

Solution:

```
t = [0:0.05:10]';  
q1 = 30*sin(t);  
q2 = 40*sin(2*t);  
q3 = 50*sin(3*t);  
  
Tx = 0*t;  
Ty = 0*t;  
  
for i=1:length(t)  
    [Tx(i),Ty(i)] = ForwardKinematics(q1(i), q2(i), q3(i));  
    pause(0.01);  
end  
  
plot(Tx, Ty)
```

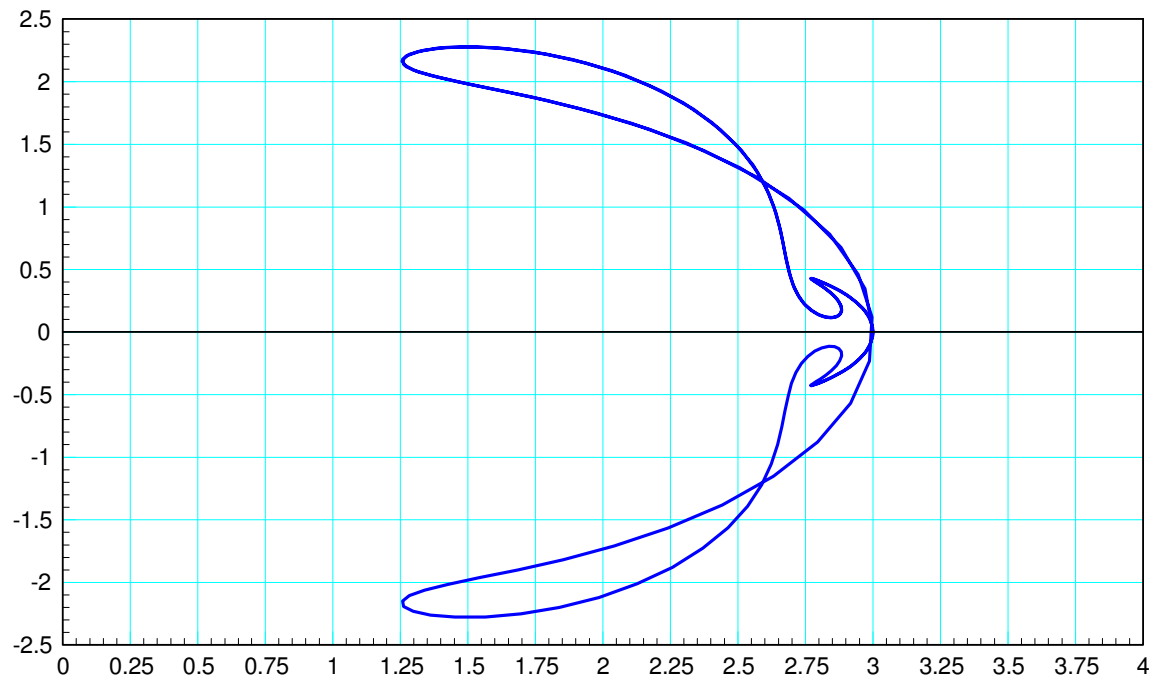


Figure 15: Tip position of robotic arm as

Problem 3: Robot Animation. Show the motion of the robot with the joint angles given in problem #2

```
function [x3, y3] = RRR(q1, q2, q3)

q1 = q1 * pi/180;
q2 = q2 * pi/180;
q3 = q3 * pi/180;

L1 = 1;
L2 = 1;
L3 = 1;

x0 = 0;
y0 = 0;

x1 = L1*cos(q1);
y1 = L1*sin(q1);

x2 = x1 + L2*cos(q1+q2);
y2 = y1 + L2*sin(q1+q2);

x3 = x2 + L3*cos(q1+q2+q3);
y3 = y2 + L3*sin(q1+q2+q3);

plot([x0,x1,x2,x3],[y0,y1,y2,y3],'b.-');
xlim([-3,3]);
ylim([-3,3]);

end
```

Calling Sequence:

```
t = [0:0.01:10]';
q1 = 30*sin(t);
q2 = 40*sin(2*t);
q3 = 50*sin(3*t);

Tx = 0*t;
Ty = 0*t;

for i=1:length(t)
    [Tx(i),Ty(i)] = RRR(q1(i), q2(i), q3(i));
    pause(0.01);
end

plot(Tx, Ty)
```

Robotics: Inverse Kinematics & *fminsearch*()

Forward kinematics determines the tip position of a robotic arm given the joint angles. For the three-link arm from before, the solution is

$$x_3 = \cos(\theta_1) + \cos(\theta_1 + \theta_2) + \cos(\theta_1 + \theta_2 + \theta_3)$$

$$y_3 = \sin(\theta_1) + \sin(\theta_1 + \theta_2) + \sin(\theta_1 + \theta_2 + \theta_3)$$

Inverse kinematics determines the joint angles of a robotic arm given the tip position. This is not that easy to do using algebra. Fortunately, there's a solution in Matlab: the function *fminsearch*()

Fminsearch() is a really useful Matlab command which finds the minimum of a function. For example, find the square root of two.

To use *fminsearch*, first create a function whose minimum is your solution. To find the square root of two, you could use this:

```
function [J] = root2(x)
    e = x*x - 2;
    J = e^2;
end
```

(save this as *root2.m*). This function has a minimum when $x = \sqrt{2}$. From the command window, you can guess and guess again to find the solution (the value of x which results in a result of zero):

```
>> root2(3)
ans =    49

>> root2(2)
ans =     4

>> root2(1.4)
ans =    0.0016
```

or you can let Matlab guess for you

```
>> [z,e] = fminsearch('root2',4)

z =    1.4143
e =    1.5665e-008
```

The first line calls function *fminsearch*, telling it to optimize function *root2*, starting with an initial guess of four. *fminsearch* then returns the best it could do:

```
z = 1.4143,
```

and the resulting minimum cost if could determine

e = 1.556e-008

That's probably not real impressive, so let's look at another example: determine the shape of a hanging chain

- Length = 13 meters
- $y(0) = 7$
- $y(10) = 5$

A hanging chain minimizes the potential energy of the chain. Since this is a minimization problem, it's perfect for *fminsearch*.

First, write a cost function which

- Is passed your guess for the y-coordinate of the chain from 1 to 9
- Computes the total length of the chain (it should be 13 meters), and
- The total potential energy of the chain

```
function [ J ] = cost_chain( z )
% [Z,e] = fminsearch('cost_chain', 10*rand(9,1))
% ECE 111 Lecture #3: fminsearch
% Shape of a hanging chain that's 13 meters long

Y = [7,z(1),z(2),z(3),z(4),z(5),z(6),z(7),z(8),z(9),5]';
PE = sum(Y);
L = 0;
for i=2:11
    L = L + sqrt(1 + (Y(i) - Y(i-1))^2);
end
E = 13-L;
J = PE + 100*E*E;
plot([0:10]', Y, '.-');
ylim([0,10]);
pause(0.01);
end
```

Start with an initial guess for the shape of the chain:

```
>> y = 10*rand(9,1);
>> cost_chain(y)

ans = 2.8806e+004
```

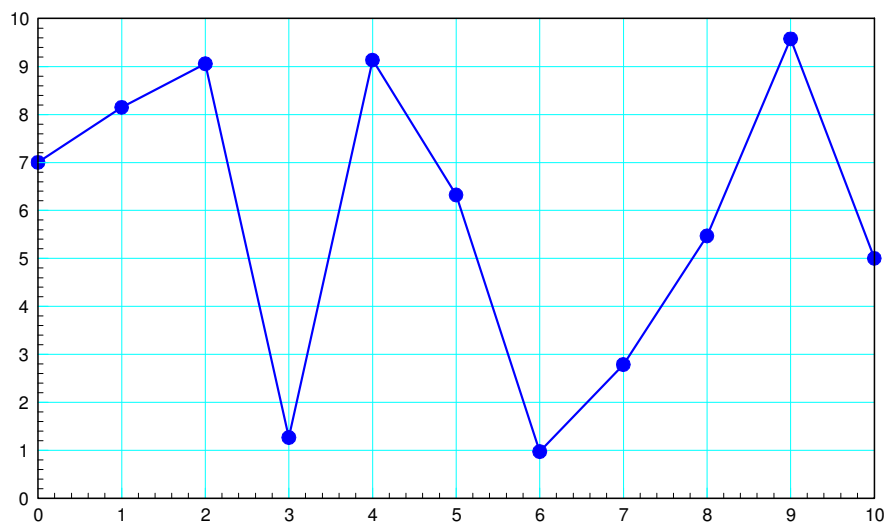


Figure 16: Initial guess for the shape of a hanging chain (bad guess)

Let *fminsearch* try to optimize this function

```
>> [z,e] = fminsearch('cost_chain', y)
```

```
Exiting: Maximum number of function evaluations has been exceeded  
- increase MaxFunEvals option.  
Current function value: 41.064042
```

Let *fminsearch* keep going, picking up where you left off:

```
>> [z,e] = fminsearch('cost_chain', z)
```

What you have is a numeric solution to the shape of a hanging chain.

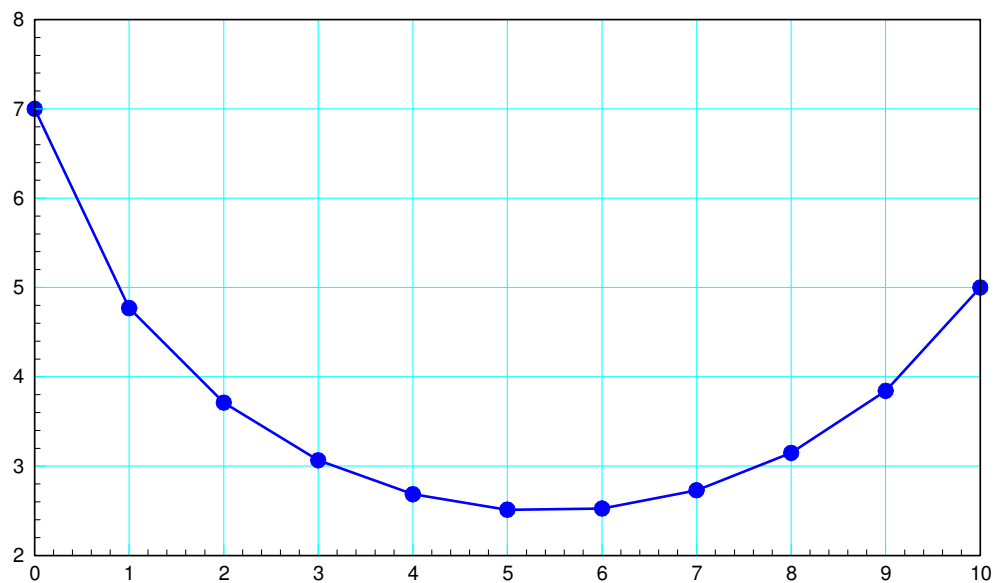


Figure 17: Shape of a hanging chain determine by using *fminsearch*()

Example 3: Find the joint angles that place a RRR robot at (x=1, y=2).

$$x_3 = 1 = \cos(\theta_1) + \cos(\theta_1 + \theta_2) + \cos(\theta_1 + \theta_2 + \theta_3)$$

$$y_3 = 2 = \sin(\theta_1) + \sin(\theta_1 + \theta_2) + \sin(\theta_1 + \theta_2 + \theta_3)$$

Solution: Create a function which

- Is passed the joint angles
- Computes the tip position,
- Computes the error in the tip position, and
- Returns the sum-squared error

```
function [J] = cost_RRR(Q)
% Tip position
Tx = 1;
Ty = 2;

[x3, y3] = RRR(Q(1), Q(2), Q(3));
pause(0.01);
Ex = x3 - Tx;
Ey = y3 - Ty;

J = Ex^2 + Ey^2;

end
```

Check by calling this function from the command window:

```
>> cost_RRR([120, -40, -50])

ans =    0.3350
```

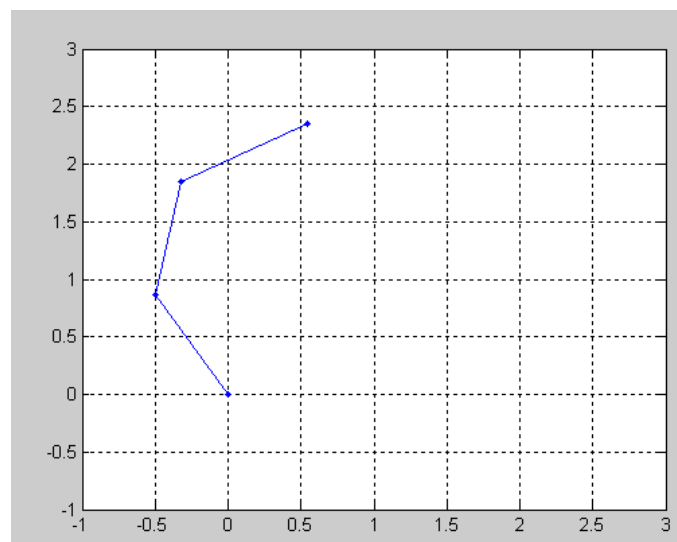


Figure 18: Matlab plot showing the position of a 3-link robot arm and the distance to (1, 2)

Optimize the function by using `fminsearch()`

```
>> [Q,e] = fminsearch('cost_RRR',[120,-40,-50])  
  
Q = 115.8522 -53.1532 -50.4906  
  
e = 3.4795e-013
```

One set of joint angles which place the tip at $(x = 1, y = 2)$ is

- $q_1 = 115.8522$ degrees
- $q_2 = -53.1532$ degrees
- $q_3 = -50.4906$ degrees

(there are other solutions)

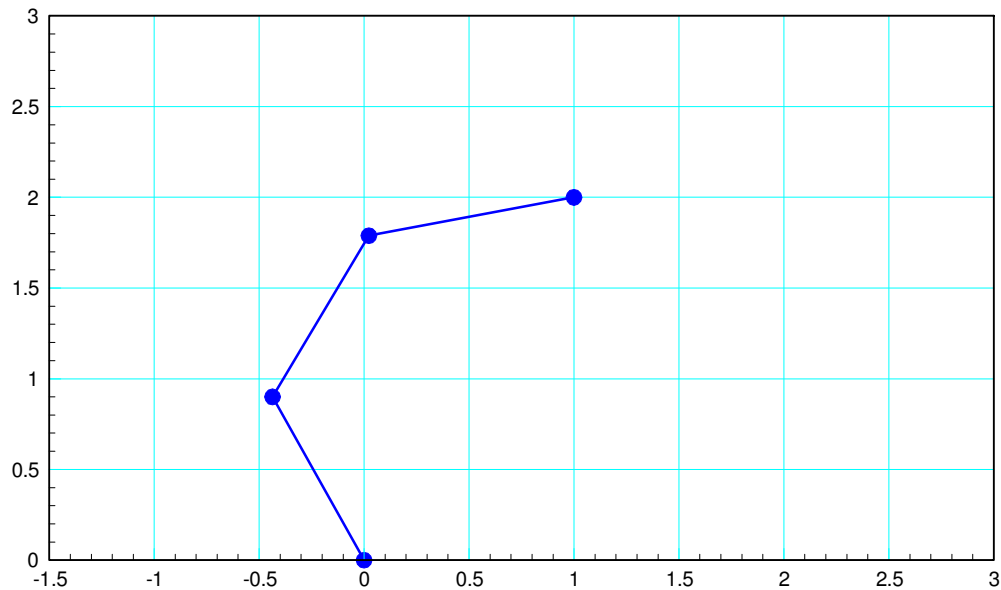


Figure 19: `fminsearch()` used to find the joint angles that place the tip at $(1, 2)$

Summary

- Trig is all about circles
- With sine and cosine functions, you can convert to and from polar coordinates
- With sine and cosine functions, you can compute the tip position of a robotic arm (forward kinematics), and
- With *fminsearch*, you can compute the joint angles which place the tip position of a robot