Math 105: Trigonometry and f(x)=0

ECE 111 Introduction to ECE

Jake Glower - Lecture #3

Please visit Bison Academy for corresponding lecture notes, homework sets, and solutions

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.*

Trigonometry is fundamental to electrical and computer engineering.

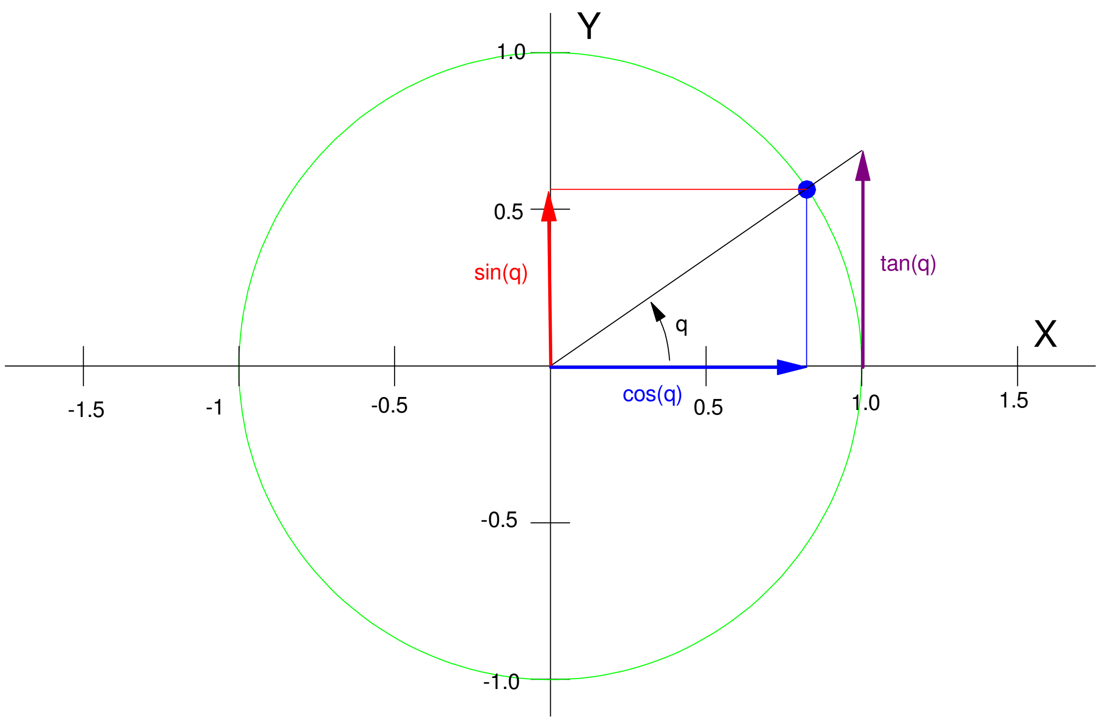
- Power is transmitted as a 60Hz sine wave
- Filters, such as subwoofers, operate on sine waves
- AC motors, such as a quad-copter motor, are driven by sine waves
- Analysis of systems described by differential equations (read: everything) depends upon being able to use complex numbers - which have their 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.

- The x-coordinate of that point is cos (θ)
- The y-coordinate of that point is sin (θ)
- 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 (θ)

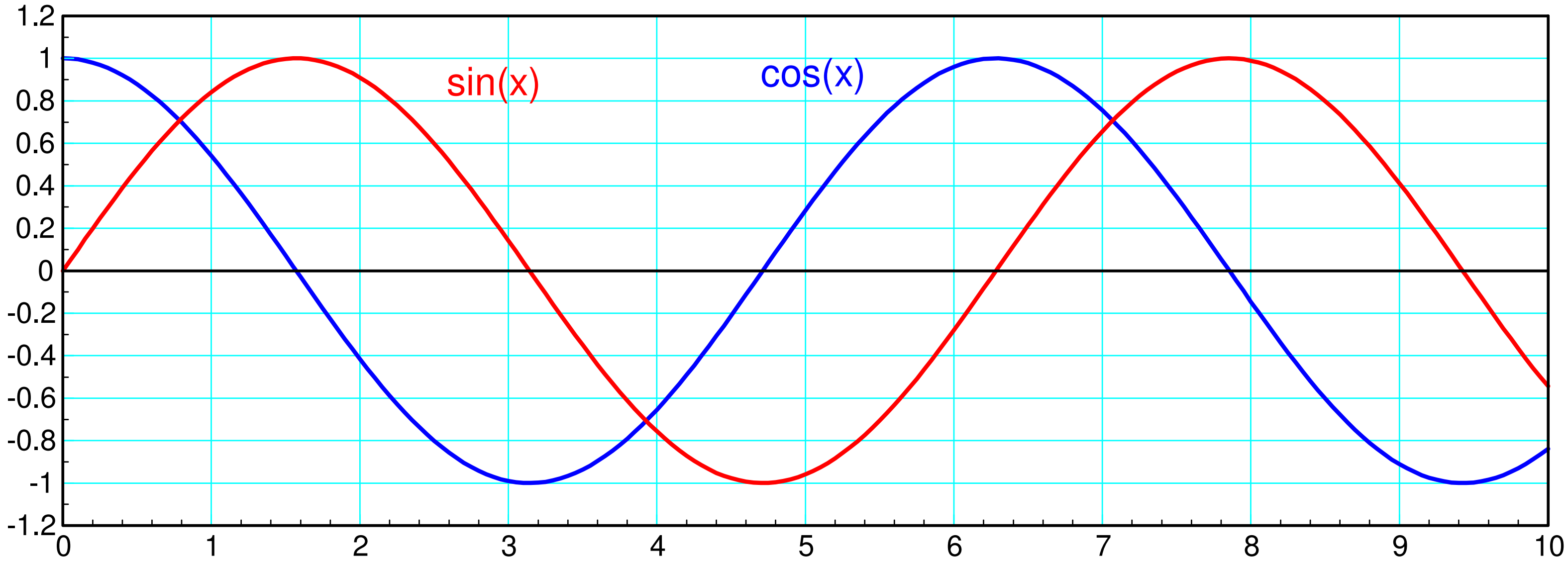


It you let the angle increase with time as

$$\theta = \omega t$$

then you get a sine wave. In Matlab:

```
>> t = [0:0.01:10]';
>> w = 1;
>> x = cos(w*t);
>> y = sin(w*t);
>> plot(t,x,t,y)
```



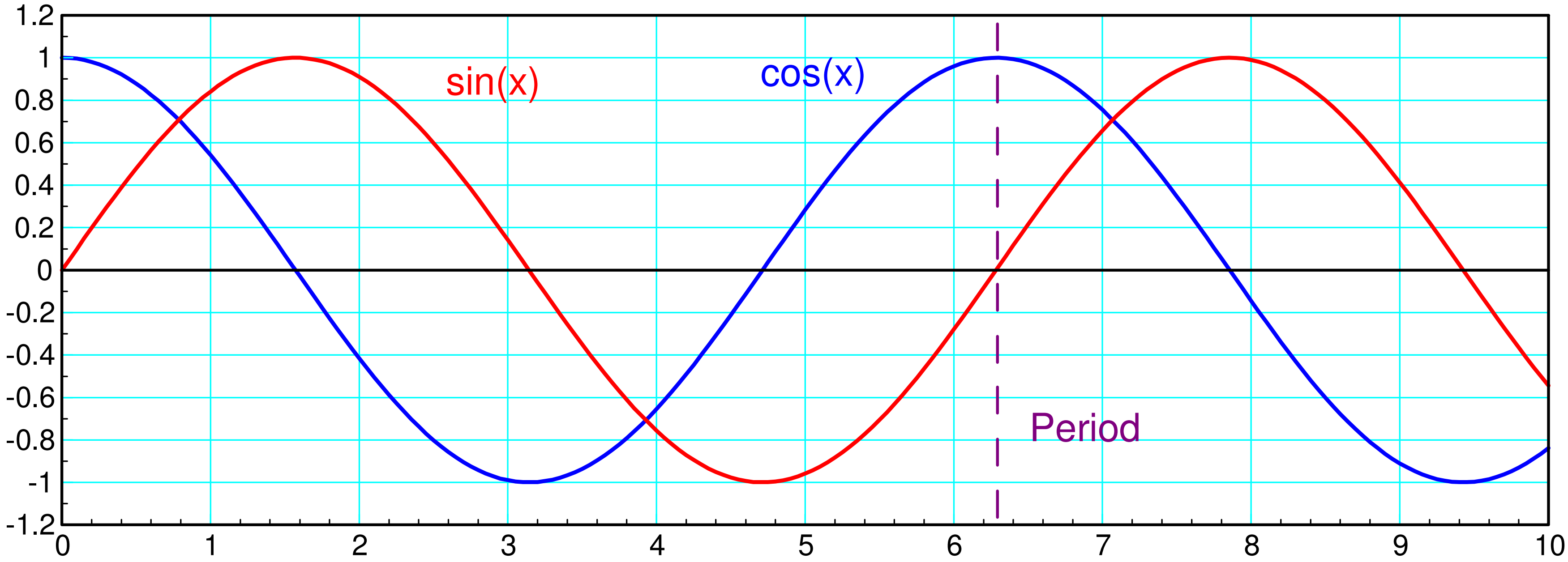
Note that

- cos() and sin() go between -1 and +1.

  Not surprising since these are just the x and y coordinates of a unit circle

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

  The function repeats every 6.28 seconds



Also note:

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

  360 degrees = 2π radians

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

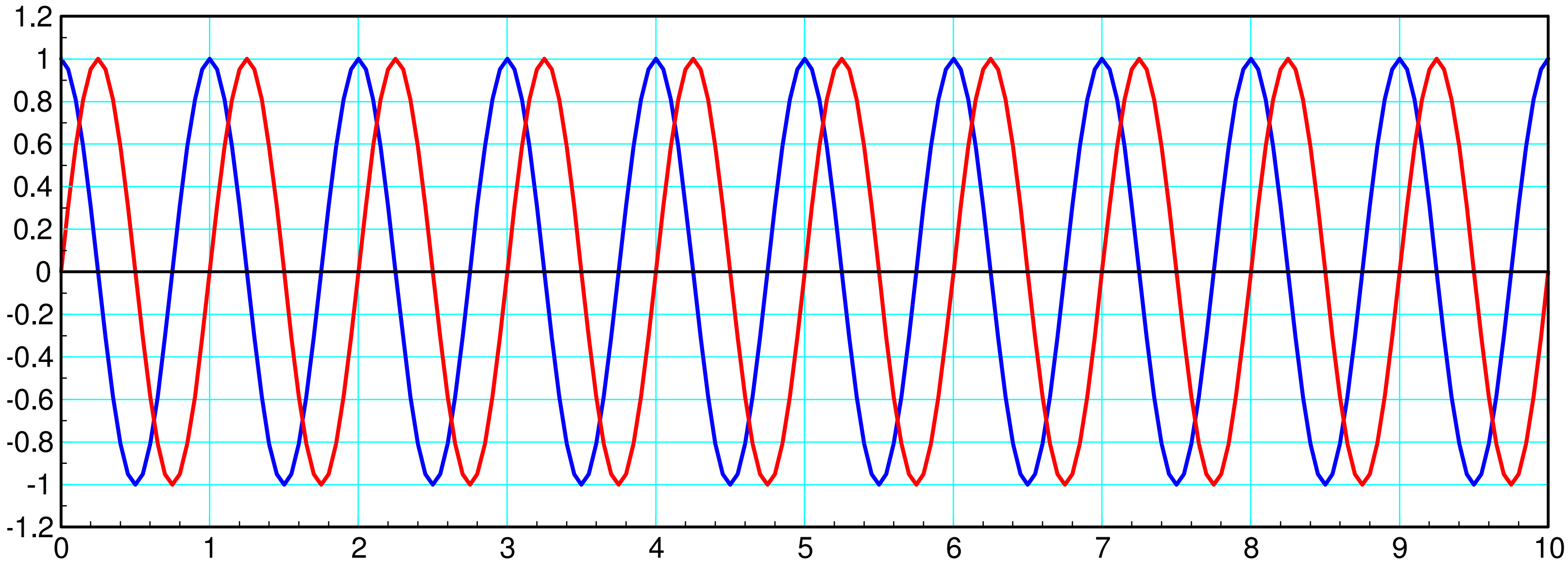
Pretty much, anything English isn't natural.

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:

```
>> t = [0:0.01:10]';
>> w = 2*pi;
>> x = cos(w*t);
>> y = sin(w*t);
>> plot(t,x,t,y)
```



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

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

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

Example,

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

looks like the following:

```
>> t = [0:0.01:2]';
>> y = 5*cos(6*t-1);
>> plot(t,y);
```

The peak is 5 Volts

The frequency is 6 rad/sec

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

The phase shift is 1 radian

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

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)
```

It also shouldn't surprising that

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

This just says that

- The radius of a circle with a radius of one
- is one

That's sort of the definition of cos() and sin().

Polar Coordinates

Any point, P, can be expressed

- In cartesian coordinates

  P = (x, y)

- Or polar coordinates

  $P = r\angle\theta$

The conversion is

$$x = r\cos\theta$$

$$y = r\sin\theta$$

or

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

$$\theta = \mathrm{atan2}(y, x)$$

Note: 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 from -pi to +pi

  (-180 degres to +180 degrees)

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 corrdinates

- *plot(x,y)* plots in cartesian coordinates

Example 1: Circles.

Trig functions are all about circles.

Both sin() and cos() plot as circles

$$r = \sin\theta$$

$$r = \cos\theta$$

Matlab Code:

```
q = [0:0.005:1]' * 2 * pi;
r = cos(q);

x = r .* cos(q);
y = r .* sin(q);

plot(x,y);
```

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);
```

Linear Spiral

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

A spiral with equal spacing

- You can also make this spin
- Matlab does animation pretty well

```
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
```

Lissajous Figures

Another pretty shape

- A staple of mad-scientists

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

$$x = \cos\theta$$

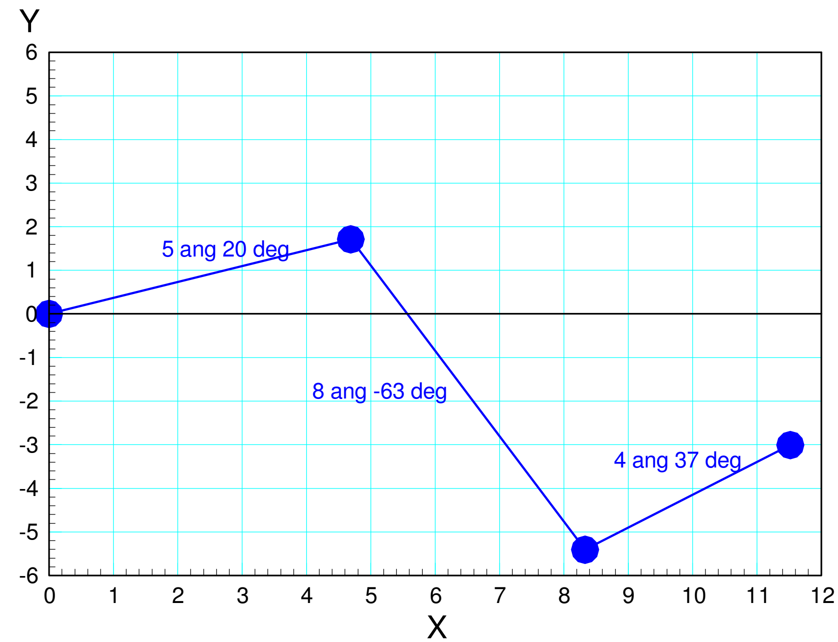
Add a small offset to y to make it rotate;

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
```

Calculations using Polar Coordinates

- Useful when adding vectors
- Convert to rectangular form
- The the x and y coordinates add.



Example, find y:

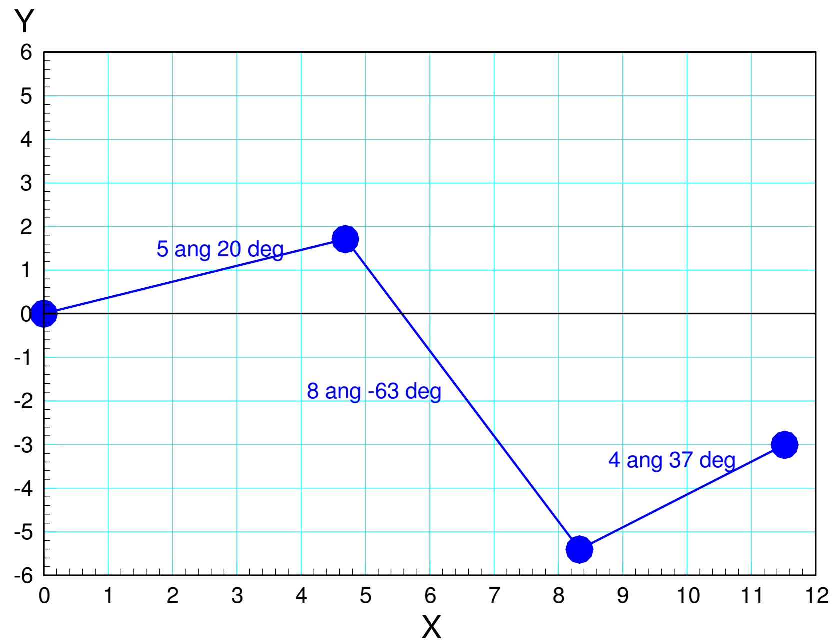
$$y = 5\angle 20^0 + 8\angle -63^0 + 4\angle 37^0$$

Convert to rectangular corrdinates

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

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 x and y terms add:

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

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

$$5\angle 20^0 + 8\angle -63^0 + 4\angle 37^0 = (11.5249, -3.0107)$$

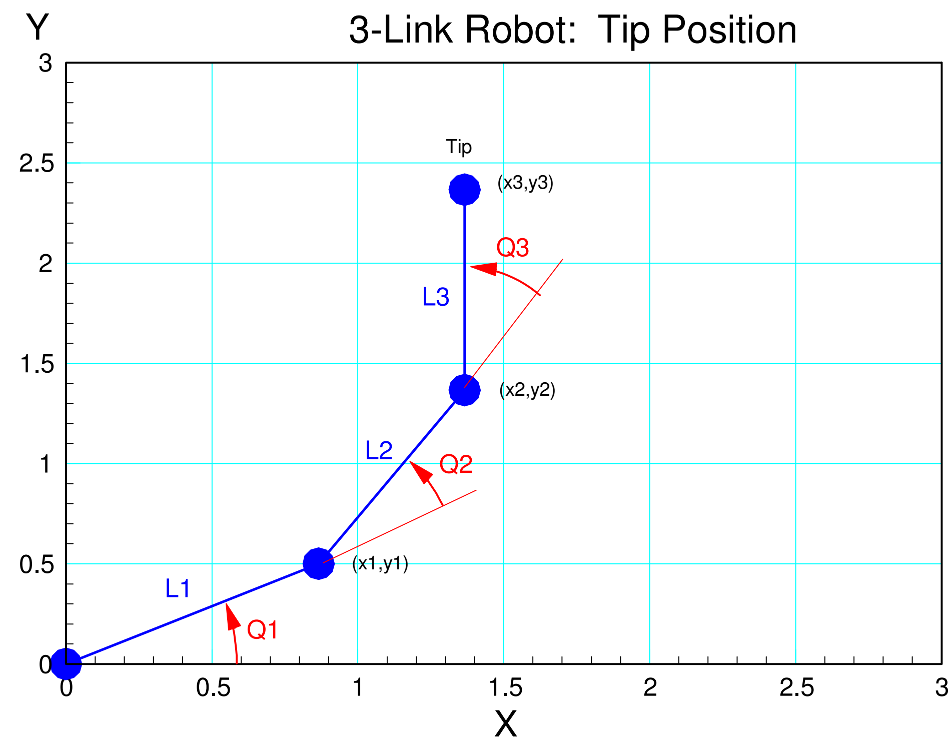
Robotics: Forward Kinematics

Given the joint angles

- Find the tip position

Example:

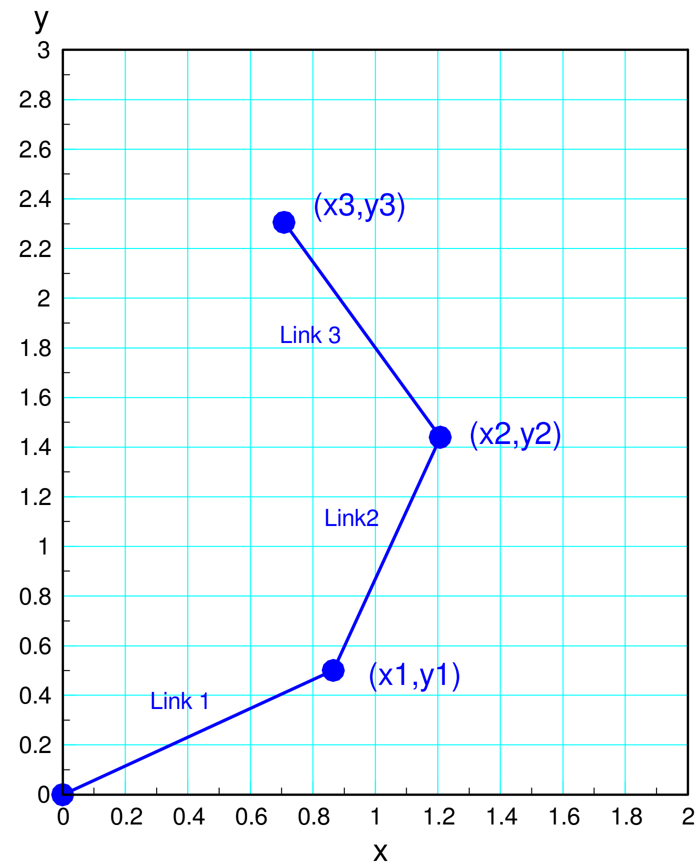
- 2D robot
- 3 rotational links
- Each link is 1m



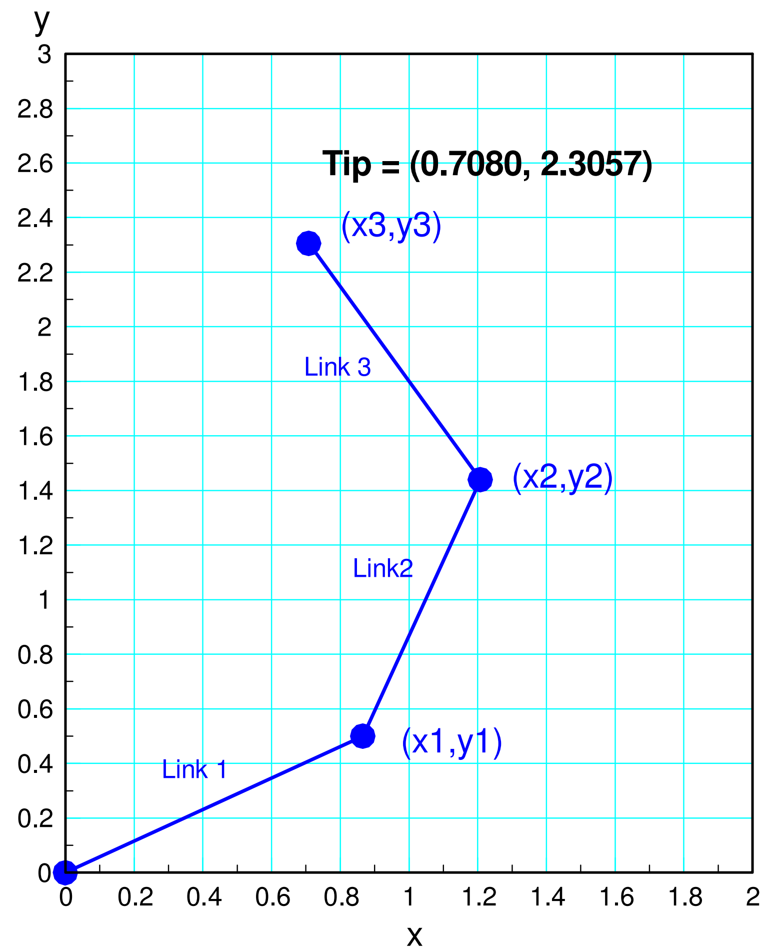
Problem 1: Find the tip position

- Angles = $\{30^0, 40^0, 50^0\}$

```
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);
```

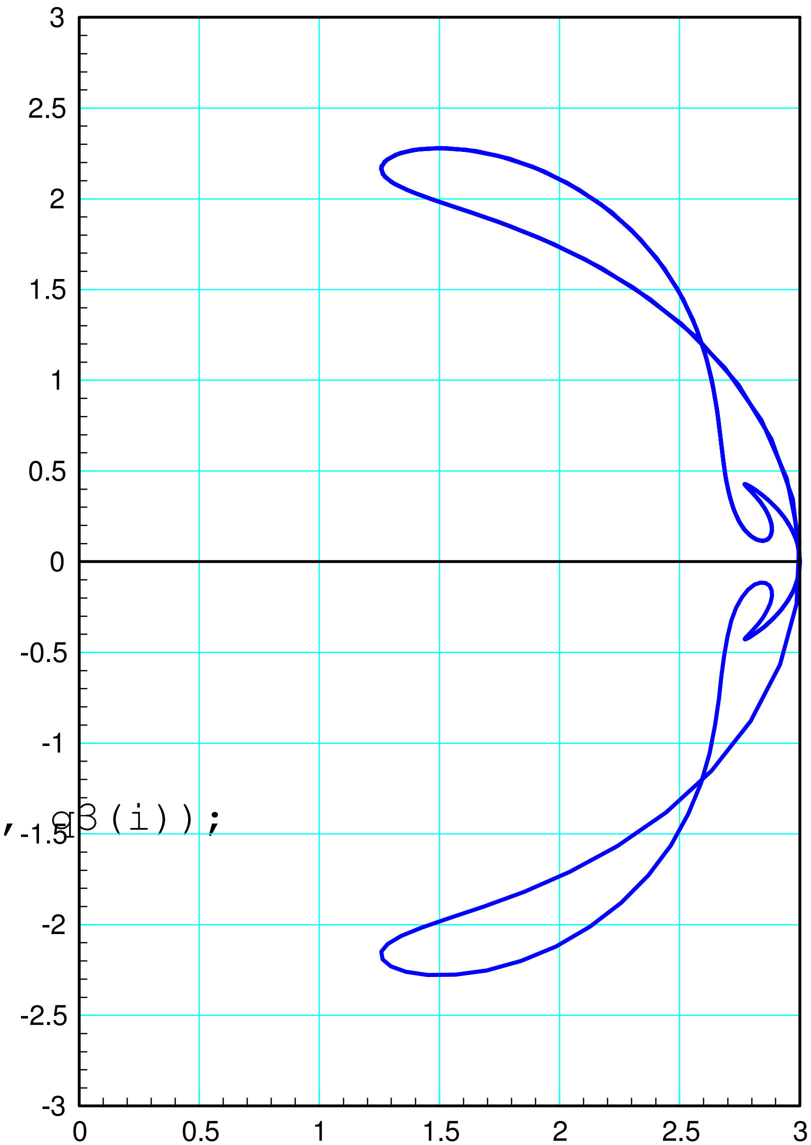


```
plot([x0,x1,x2,x3],[y0,y1,y2,y3],'b.-');
xlim([0,3]);
ylim([0,3]);

pause(0.01);

end

>> [Px,Py] = RRR(30,40,50)
Px =     0.7080
Py =     2.3057
```



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

- Q1 = 30*sin(t) degrees
- Q2 = 40*sin(2t) degrees
- Q3 = 50*sin(3t) degrees



Solution:

```
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: Given the joint angles, determine the tip position

- Example: 3-link robot

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

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

Inverse Kinematics: Given the tip position, determine the joint angles

- Not an easy problem to solve
- Fortunately, there's Matlab & *fminsearch()*

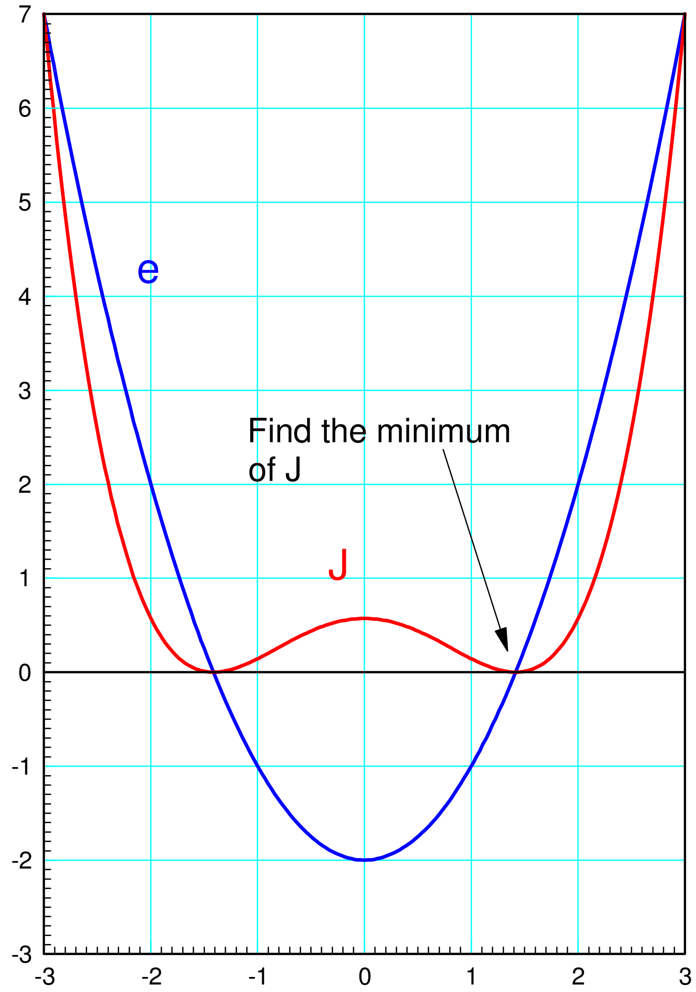
fminsearch()

- A really useful Matlab command
- Finds the minimum of a function.

Example 1: Find $\sqrt{2}$

Step 1: Create a function whose minimum is your solution.

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



Step 2: Find the minimum

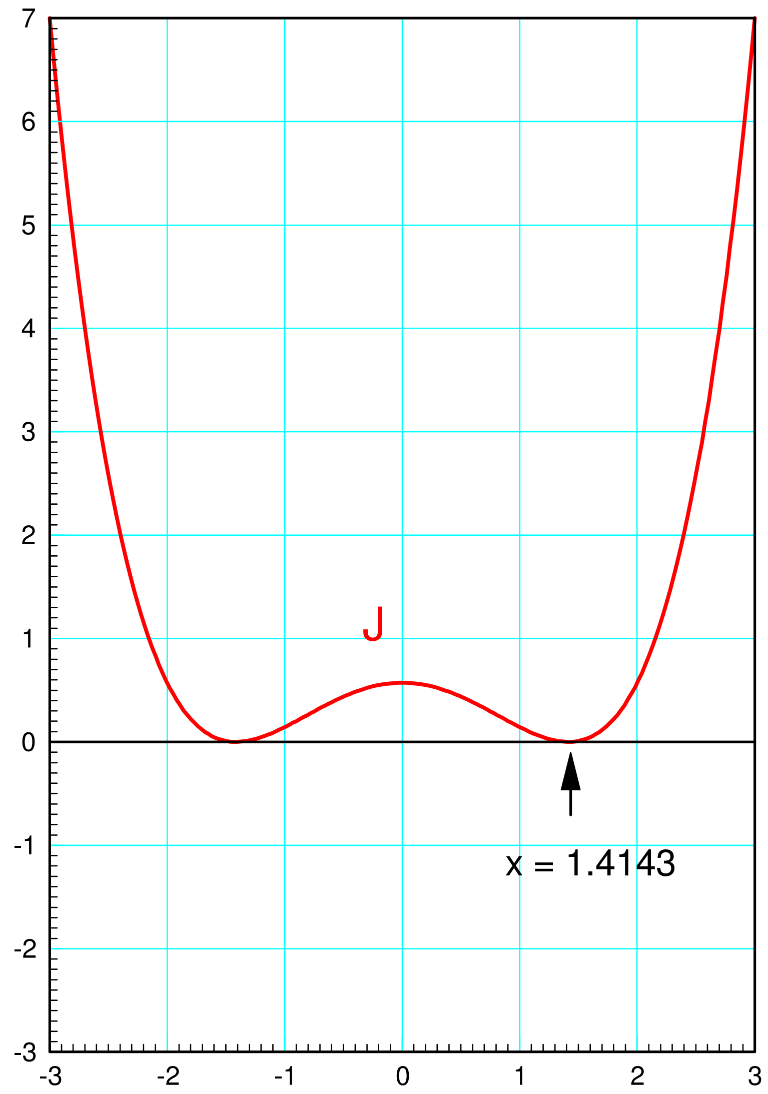
Option 1: Guess and guess again

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

>> root2(2)
ans =     4

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

Option 2: Let Matlab guess for you

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

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

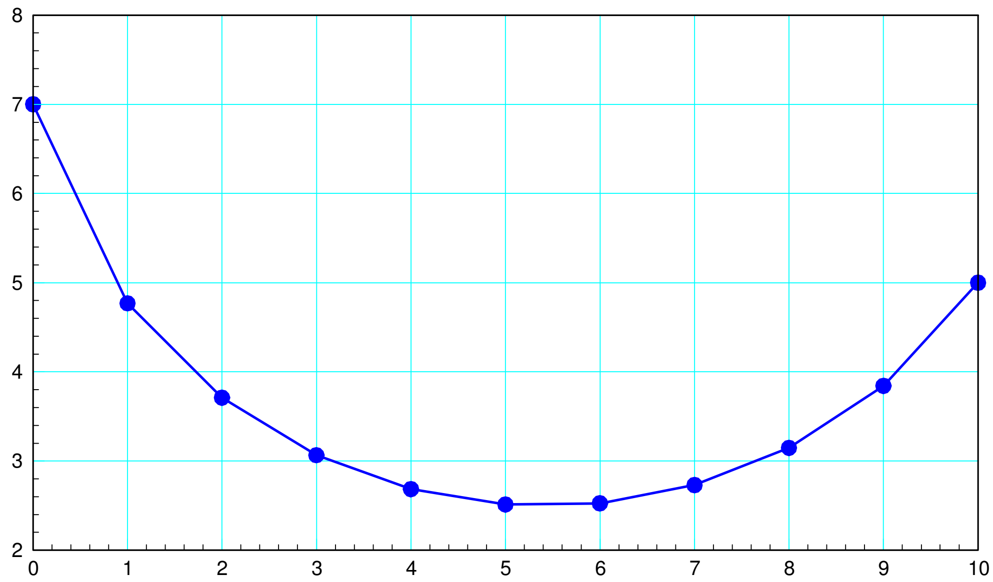
Solution: $\sqrt{2} = 1.4143$



Example 2: Find 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 lenngth 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
```

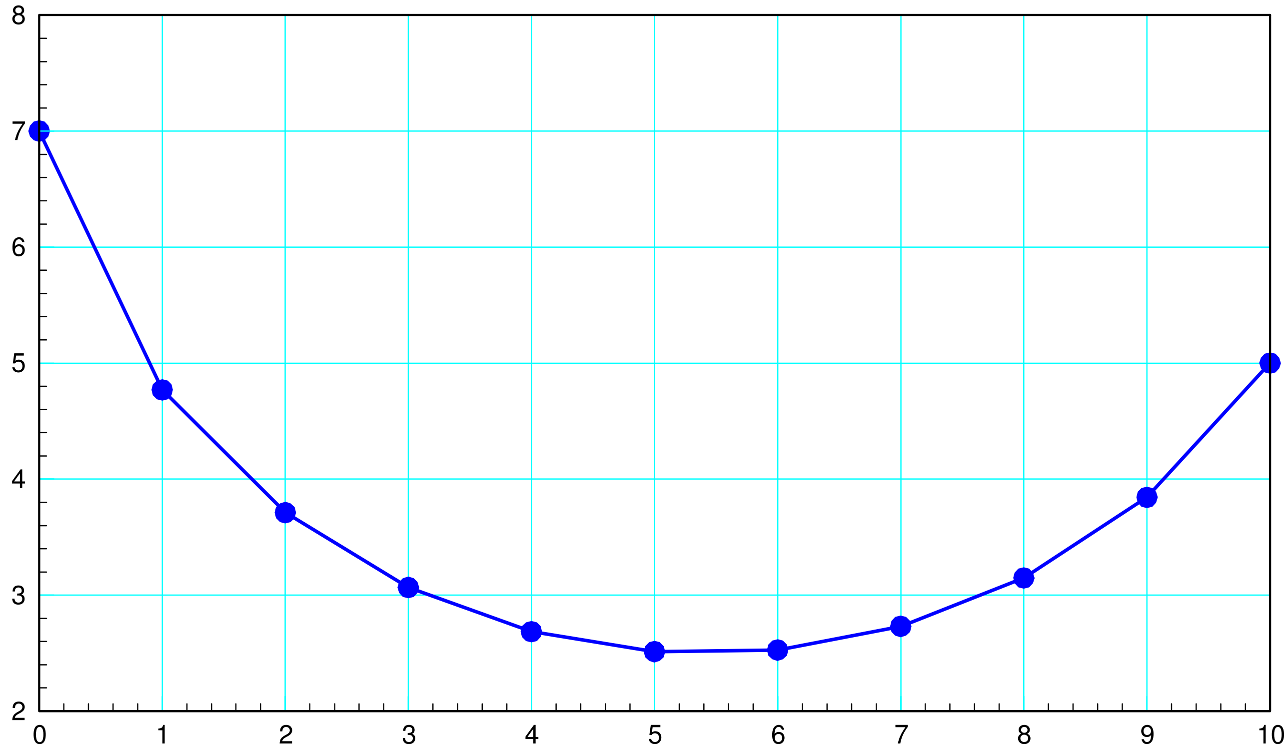
Let *fminsearch* try to optimize this funciton

```
>> [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.



Shape of a hanging chain found 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

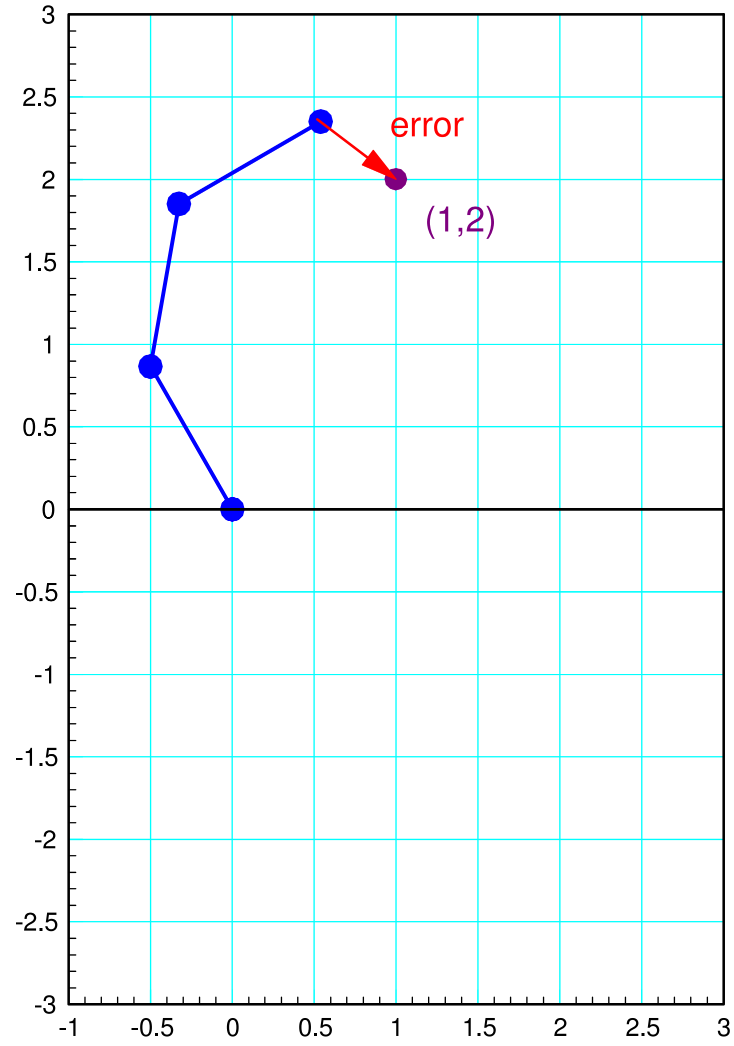
- Is passed the joint angles
- Computes the tip position,
- Computes the error in the tip position, and
- Returns the sum-squared error

Cost Function:

```
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
```

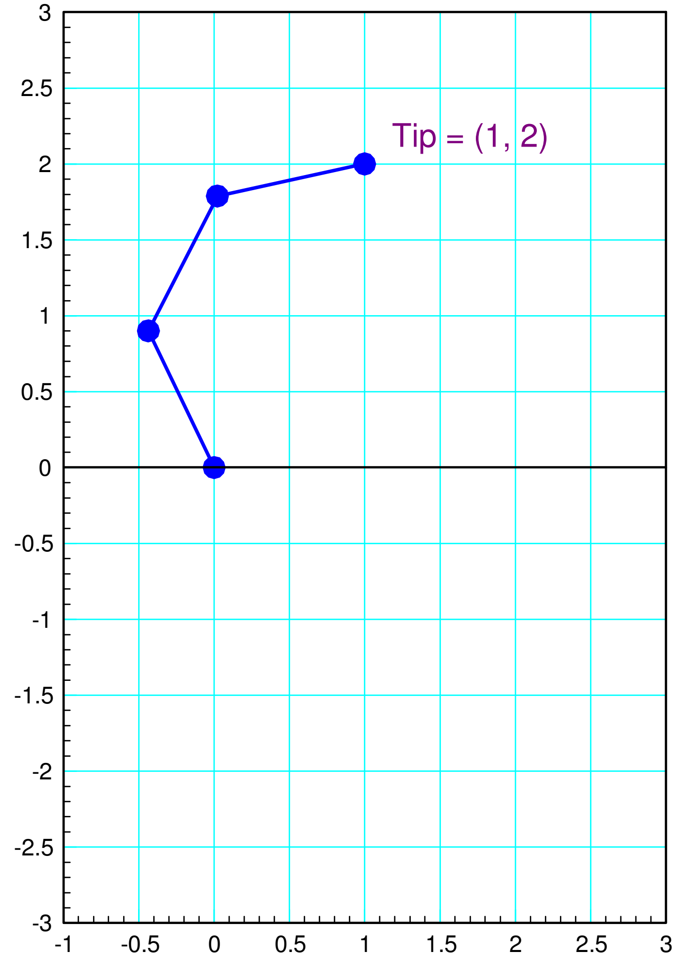
Check by calling this function from the command window:

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

ans =     0.3350
```



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
```



Solution:

- q1 = 115.8522 degrees
- q2 = -53.1532 degrees
- q3 = -50.4906 degrees

(there are other solutions)

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 *fmisearch*, you can compute the joint angles which place the tip position of a robot