- Previous Lecture:
  - 1-d array—vector
  - Probability and random numbers
- Today's Lecture:
  - Plots using vectors
  - More examples on vectors and simulation
- Announcement:
  - Discussion this week in Upson B7 lab
  - Project 3 due on Thurs March 5
  - Prelim 1 on Tues March 10, 7:30-9pm
  - Review session on Sunday 1:30-3pm, Kimball B11. Note that Daylight Saving Time begins on Sunday.

Plot makes use of vectors: Drawing a single line segment

```
a= 0;  % x-coord of pt 1
b= 1;  % y-coord of pt 1
c= 5;  % x-coord of pt 2
d= 3;  % y-coord of pt 2
plot([a c], [b d], '-*')
```

x-values (a vector)

y-values (a vector)

Line/marker format

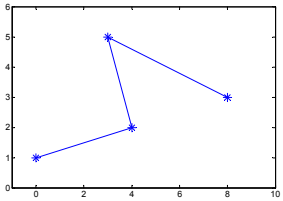
Making an x-y plot

```
a= [0 4 3 8];  % x-coords
b= [1 2 5 3];  % y-coords
plot(a, b, '-*')
```

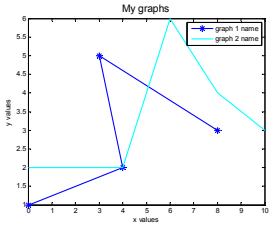
x-values (a vector)

y-values (a vector)

Line/marker format



Making an x-y plot with multiple graphs (lines)

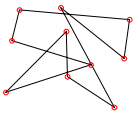


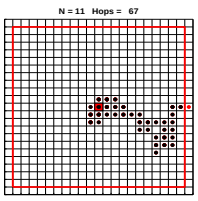

```
a= [0 4 5 8];
b= [1 2 5 3];
f= [0 4 6 8 10];
g= [2 2 6 4  3];
plot(a,b,'-*',f,g,'c')
legend('graph 1 name', 'graph 2 name')
xlabel('x values')
ylabel('y values')
title('My graphs', 'Fontsize',14)
```

See `plotComparison2.m`

Simulation

- Imitates real system
- Requires judicious use of random numbers
- Requires many trials
- → opportunity to practice working with vectors!





Loop patterns for working with a vector

```
% Given a vector v

for k = 1:length(v)

   % Work with v(k)
   % E.g.,  disp(v(k))

end
```

```
% Given a vector v
k = 1;
while k<=length(v)

   % Work with v(k)
   % E.g.,  disp(v(k))

   k = k+1;

end
```

% Simulate the rolling of 2 fair dice
totalOutcome= ???

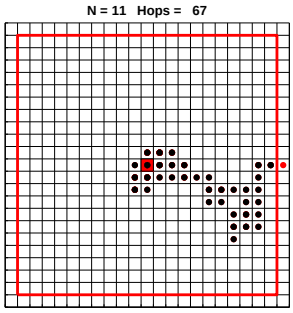
A `ceil(rand*12)`
B `ceil(rand*11)+1`
C `floor(rand*11)+2`
D *2 of the above*
E *None of the above*

2-dimensional random walk

Start in the middle tile, (0,0).

For each step, randomly choose between N,E,S,W and then walk one tile. Each tile is 1×1.

Walk until you reach the boundary.




```
function [x, y] = RandomWalk2D(N)

k=0;  xc=0;  yc=0;

while  abs(xc)<N && abs(yc)<N
   % Choose random dir, update xc,yc



   % Record new location in x, y
   k=k+1;  x(k)=xc;  y(k)=yc;
end
```

```
% Standing at (xc,yc)
% Randomly select a step
    r= rand(1);
    if r < .25
        yc= yc + 1;  % north
    elseif r < .5
        xc= xc + 1;  % east
    elseif r < .75
        yc= yc -1;   % south
    else
        xc= xc -1;   % west
    end
```

See `RandomWalk2D.m`

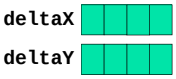
Another representation for the random step

- Observe that each update has the form

  $xc = xc + \Delta x$

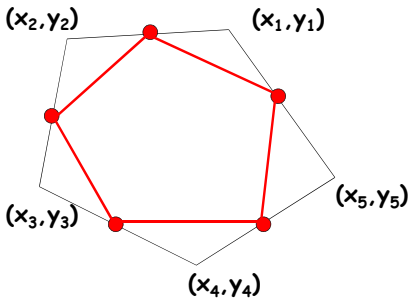
  $yc = yc + \Delta y$

  no matter which direction is taken.
- So let's get rid of the if statement!
- Need to create two "change vectors" deltaX and deltaY

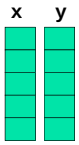


See `RandomWalk2D_v2.m`

Example: polygon smoothing

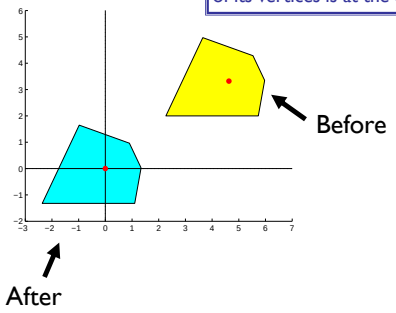


Can store the x-y coordinates in vectors x and y



First operation: centralize

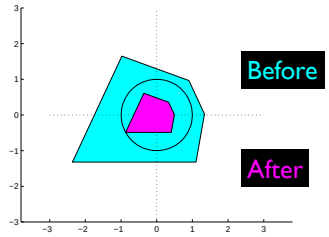
Move a polygon so that the centroid of its vertices is at the origin




```
function [xNew,yNew] = Centralize(x,y)
% Translate polygon defined by vectors
% x,y such that the centroid is on the
% origin. New polygon defined by vectors
% xNew,yNew.
```


Second operation: normalize

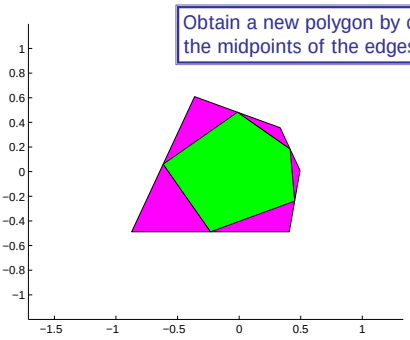
Shrink (enlarge) the polygon so that the vertex furthest from the (0,0) is on the unit circle




```
function [xNew,yNew] = Normalize(x,y)
% Resize polygon defined by vectors x,y
% such that distance of the vertex
% furthest from origin is 1
```


Third operation: smooth

Obtain a new polygon by connecting the midpoints of the edges



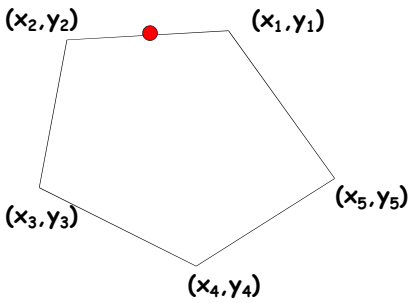

```
function [xNew,yNew] = Smooth(x,y)
% Smooth polygon defined by vectors x,y
% by connecting the midpoints of
% adjacent edges

n = length(x);
xNew = zeros(n,1);
yNew = zeros(n,1);
for i=1:n
  %Compute midpt of ith edge. Store in xNew(i), yNew(i)



end
```

```
xNew(1) = (x(1)+x(2))/2
yNew(1) = (y(1)+y(2))/2
```



Lecture 11 40

Polygon Smoothing

```
% Given n, x, y
for i=1:n
     xNew(i) = (x(i) + x(i+1))/2;
     yNew(i) = (y(i) + y(i+1))/2;
end
```

Does above fragment compute the new n-gon?

A: Yes

B: No

Lecture 11 43

Show a simulation of polygon smoothing

Create a polygon with randomly located vertices.

Repeat:

- Centralize
- Normalize
- Smooth

See **ShowSmooth.m**

Lecture 11 49