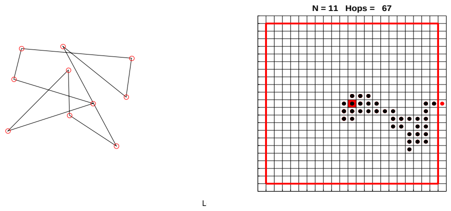
- Previous Lecture:
  - 1-d array—vector
  - Probability and random numbers
- Today's Lecture:
  - More examples on vectors and simulation
- Announcements:
  - Discussion this week in the computer lab
  - Project 3 due on Mon 10/3

Simulation

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

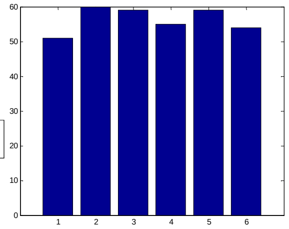


Simulation result

Data in bins

`bar(1:6, count)`

Bin numbers



51	60	59	55	59	54
1	2	3	4	5	6

Keep tally on repeated rolls of a fair die

```
Repeat the following:

    % roll the die

    % increment correct "bin"
```

```
function count = rollDie(rolls)

FACES= 6;                     % #faces on die
count= zeros(1,FACES);

% Count outcomes of rolling a FAIR die
for k= 1:rolls
    % Roll kth die
    face= ceil(rand*FACES);
    % Increment the appropriate bin

end

% Show histogram of outcome
```

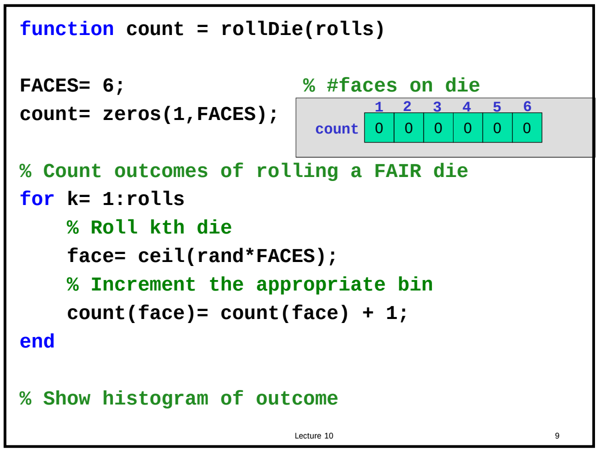
	1	2	3	4	5	6
count	0	0	0	0	0	0

```
% Count outcomes of rolling a FAIR die
count= zeros(1,6);
for k= 1:100
    face= ceil(rand*6);
    if face==1
        count(1)= count(1) + 1;
    elseif face==2
        count(2)= count(2) + 1;
    ⋮
    elseif face==5
        count(5)= count(5) + 1;
    else
        count(6)= count(6) + 1;
    end
end
```

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

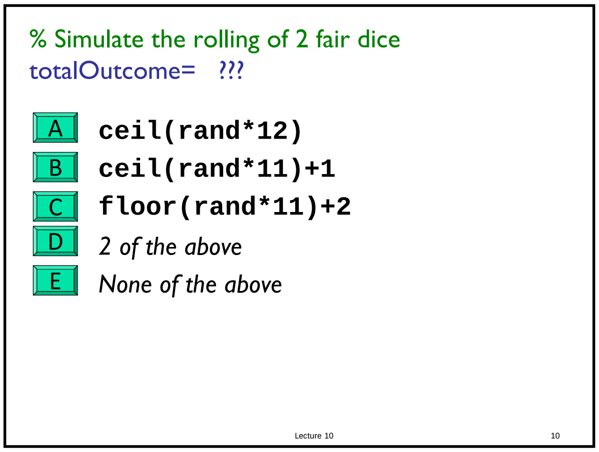
Looks redundant … is it? Is there a more concise way?

From `rollDieV1.m`



```
function count = rollDie(rolls)

FACES= 6;                     % #faces on die
count= zeros(1,FACES);

% Count outcomes of rolling a FAIR die
for k= 1:rolls
    % Roll kth die
    face= ceil(rand*FACES);
    % Increment the appropriate bin
    count(face)= count(face) + 1;
end

% Show histogram of outcome
```

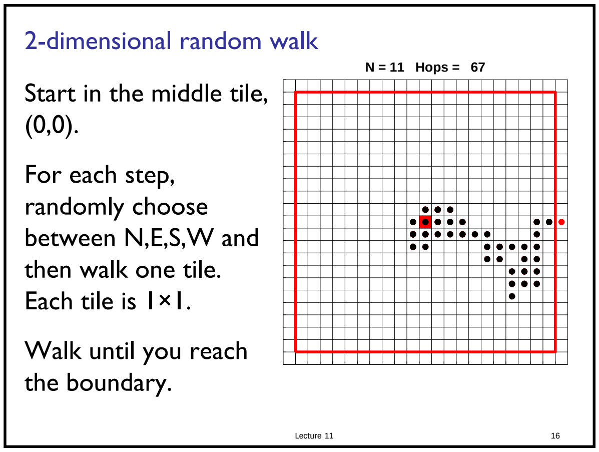
	1	2	3	4	5	6
count	0	0	0	0	0	0



% 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*



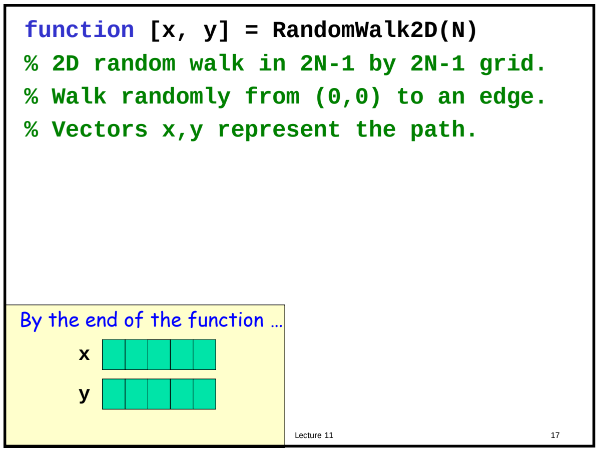
2-dimensional random walk

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.

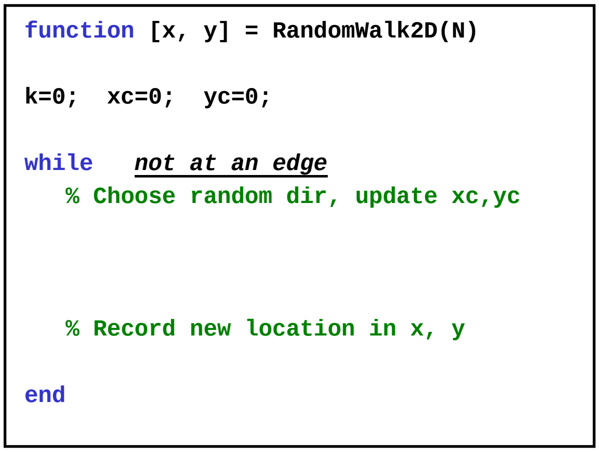
N = 11 Hops = 67



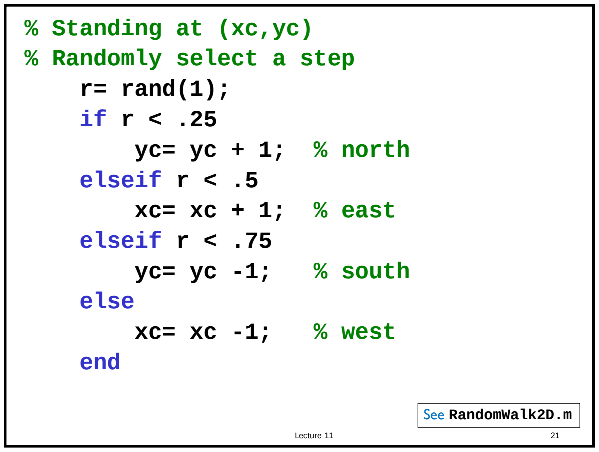
```
function [x, y] = RandomWalk2D(N)
% 2D random walk in 2N-1 by 2N-1 grid.
% Walk randomly from (0,0) to an edge.
% Vectors x,y represent the path.
```

By the end of the function …

x					
y					

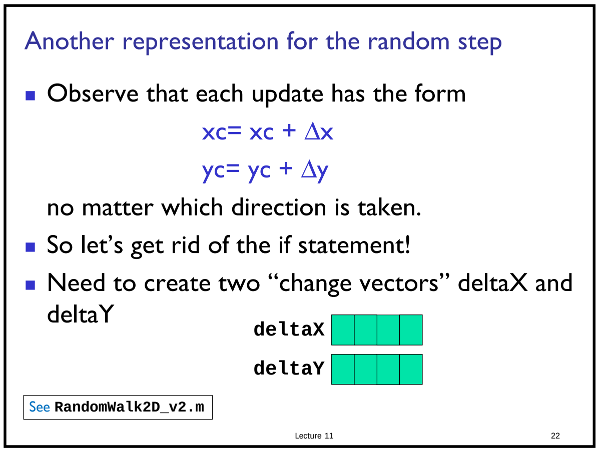


```
function [x, y] = RandomWalk2D(N)

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

while   not at an edge
   % Choose random dir, update xc,yc



   % Record new location in x, y

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 + Δx

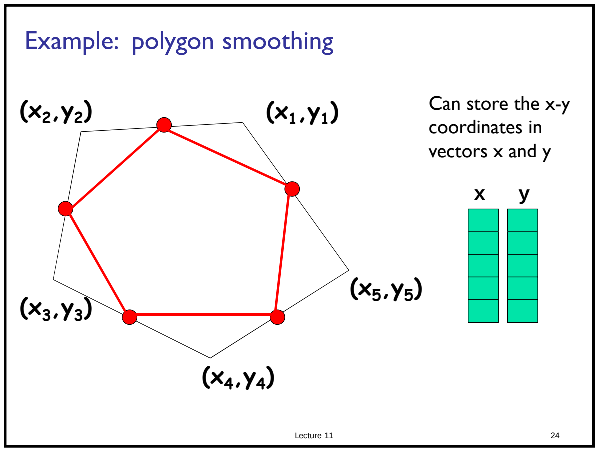
  yc= yc + Δy

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

deltaX 🟩🟩🟩🟩

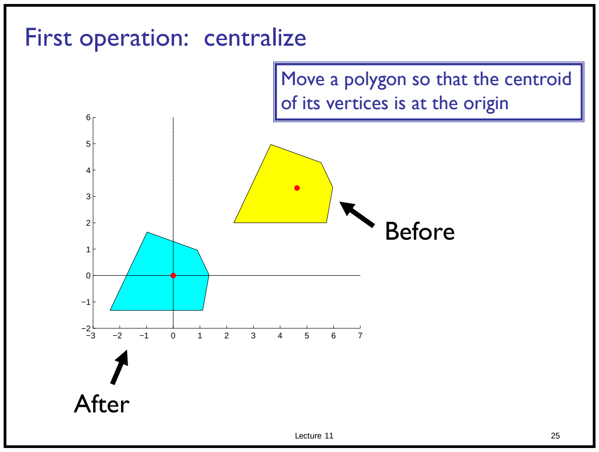
deltaY 🟩🟩🟩🟩

See `RandomWalk2D_v2.m`



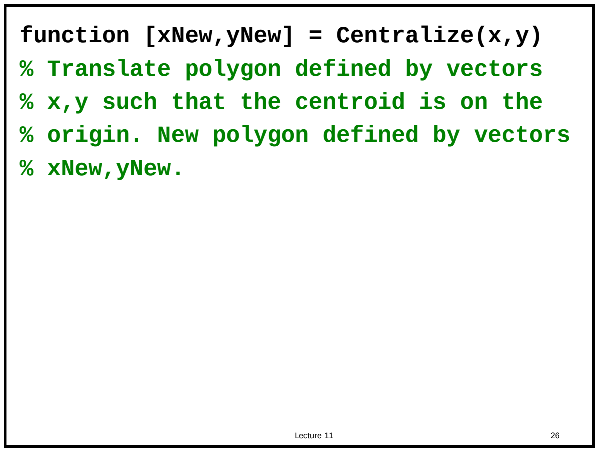
Example: polygon smoothing

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

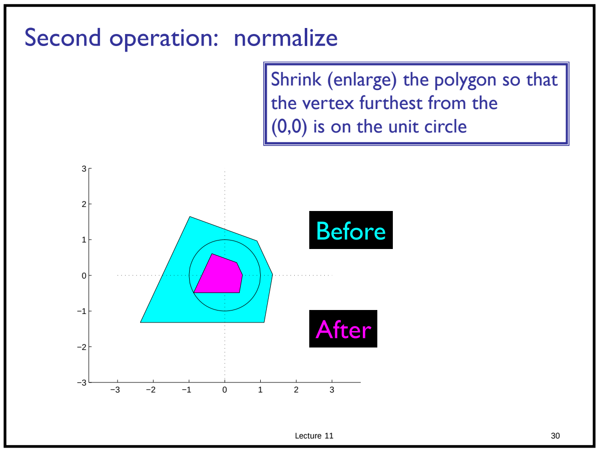


First operation: centralize

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

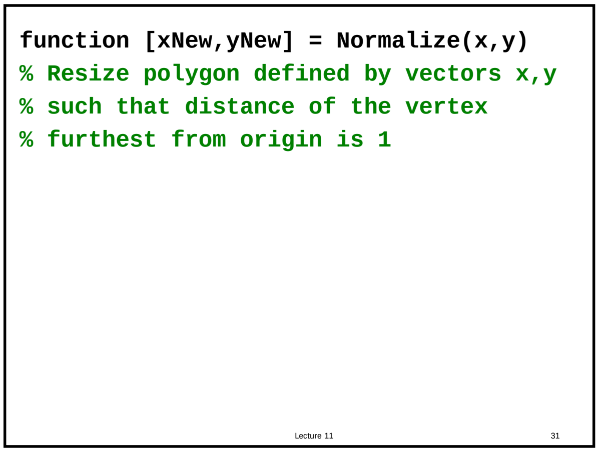


```
function [xNew,yNew] = Centralize(x,y)
% Translate polygon defined by vectors
% x,y such that the centroid is on the
% origin. New polygon defined by vectors
% xNew,yNew.
```

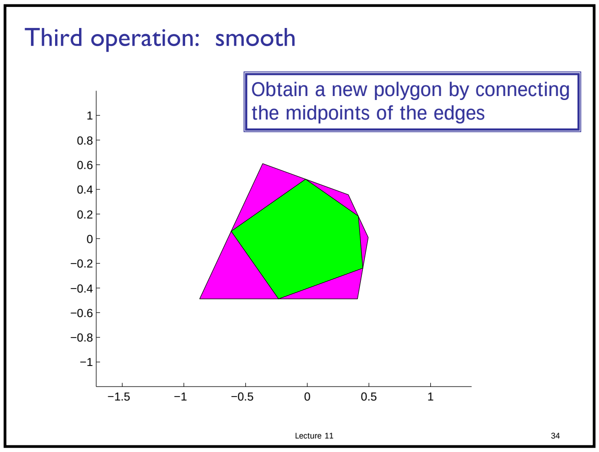


Second operation: normalize

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

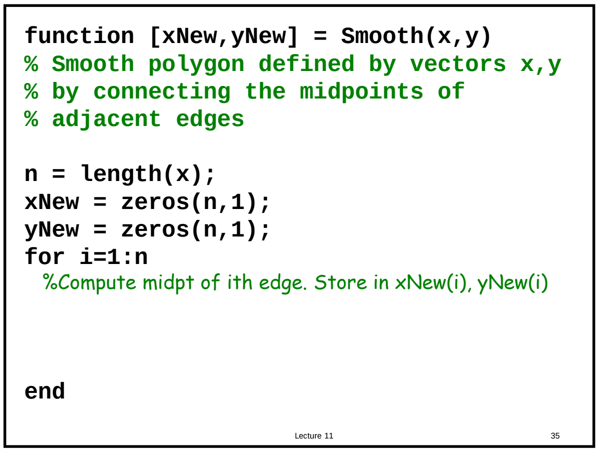


```
function [xNew,yNew] = Normalize(x,y)
% Resize polygon defined by vectors x,y
% such that distance of the vertex
% furthest from origin is 1
```

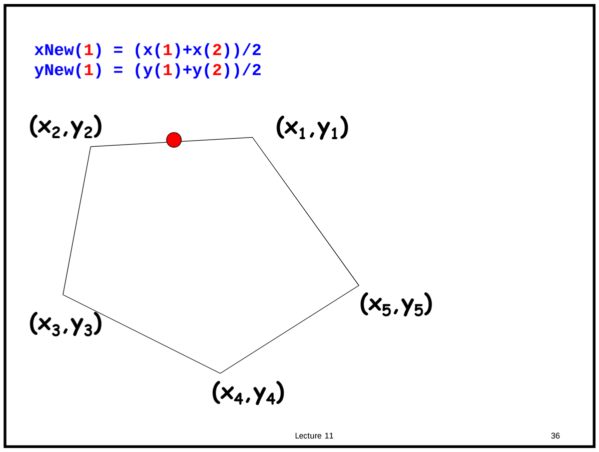


Third operation: smooth

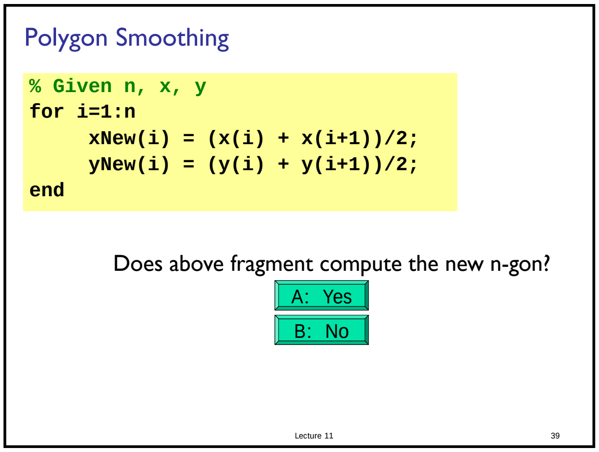
Obtain a new polygon by connecting the midpoints of the edges



```
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
```



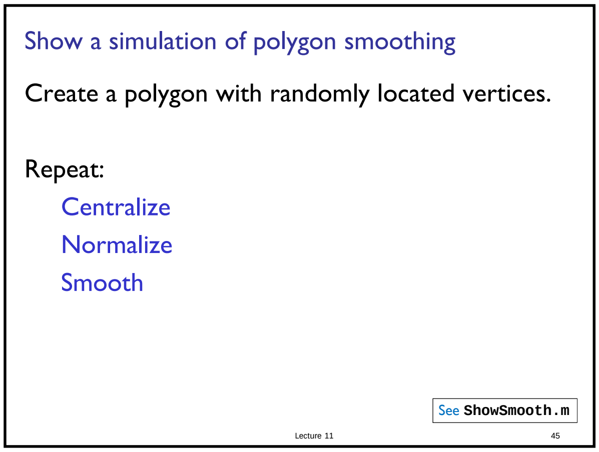
Polygon Smoothing

```
% 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



Show a simulation of polygon smoothing

Create a polygon with randomly located vertices.

Repeat:

- Centralize
- Normalize
- Smooth

See `ShowSmooth.m`