- Previous Lecture:
  - Probability and random numbers
  - 1-d array—vector
- Today's Lecture:
  - More examples on vectors
  - Simulation
- Announcement:
  - Project 3 posted. Due 3/10.
  - Prelim 2 on 3/17. Please let us know now (email Randy Hess, rbhess@cs.cornell.edu) if you have a university-scheduled conflict.

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
```

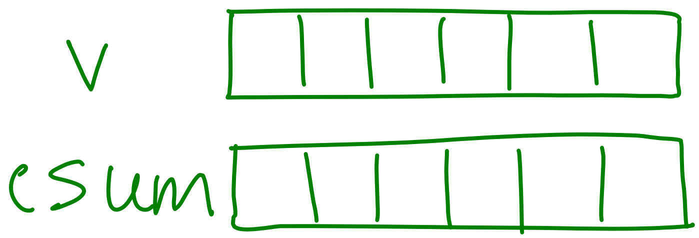
```
% Given a vector v
k = 1;
while k <= length(v)

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

   k = k+1;

end
```

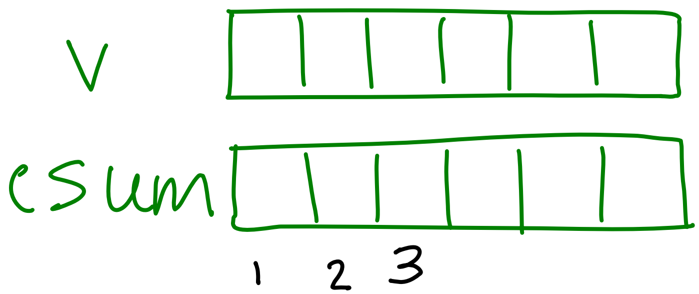
Example

- Write a program fragment that calculates the cumulative sums of a given vector **v**.
- The cumulative sums should be stored in a vector of the same length as **v**.

1, 3, 5, 0 **v**

1, 4, 9, 9 cumulative sums of **v**


v
csum



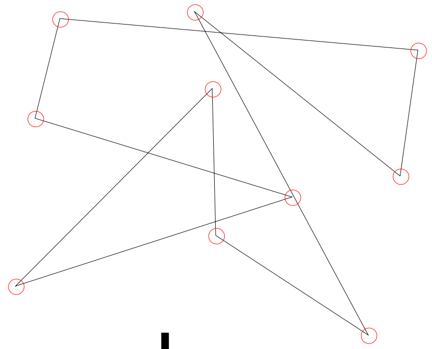
$csum(k) = csum(k-1) + v(k)$

$csum(3) = v(1) + v(2) + v(3)$

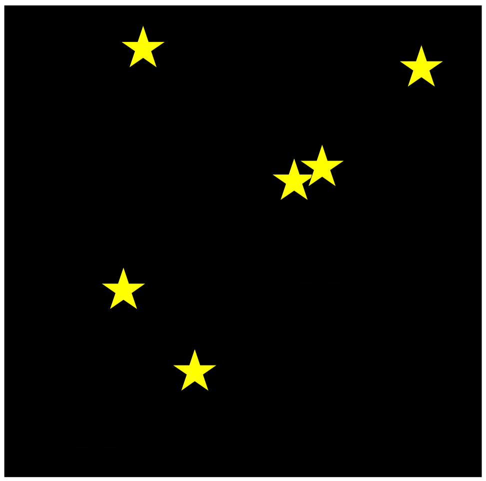
$csum(4) = \underbrace{v(1) + v(2) + v(3)}_{csum(3)} + v(4)$

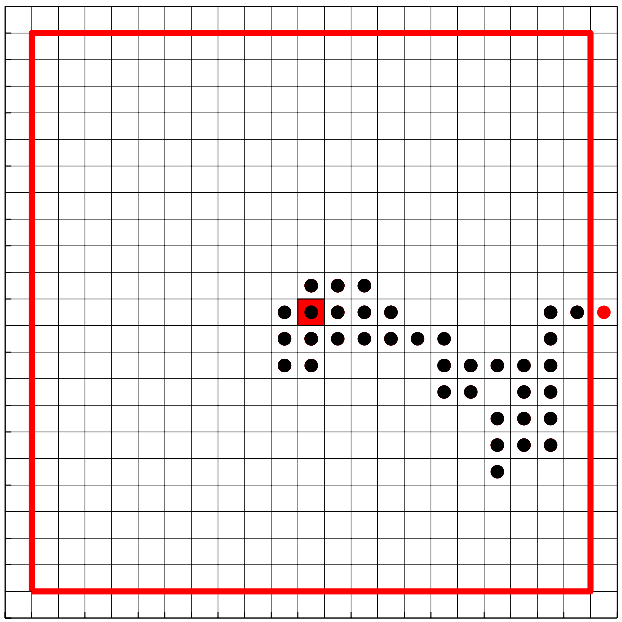
```
csum(1) = v(1);
for k = 2:length(v)
    csum(k) = csum(k-1) + v(k);
end
```

Simulation



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



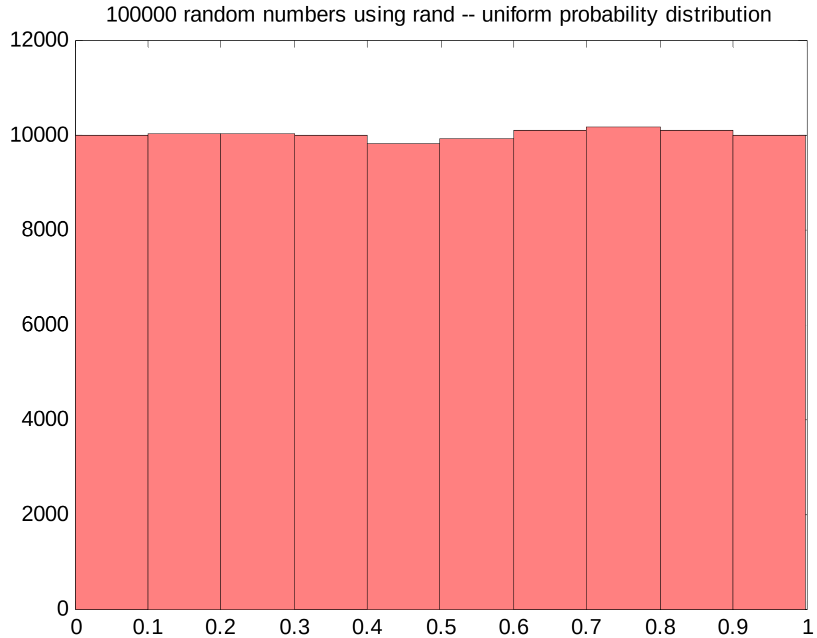


L

Random numbers

- *Pseudorandom* numbers in programming
- Function `rand(…)` generates random real numbers in the interval (0,1). All numbers in the interval (0,1) are equally likely to occur—uniform probability distribution.
- Examples:

`rand(1)`	one random # in (0,1)
`6*rand(1)`	one random # in (0,6)
`6*rand(1)+1`	one random # in (1,7)

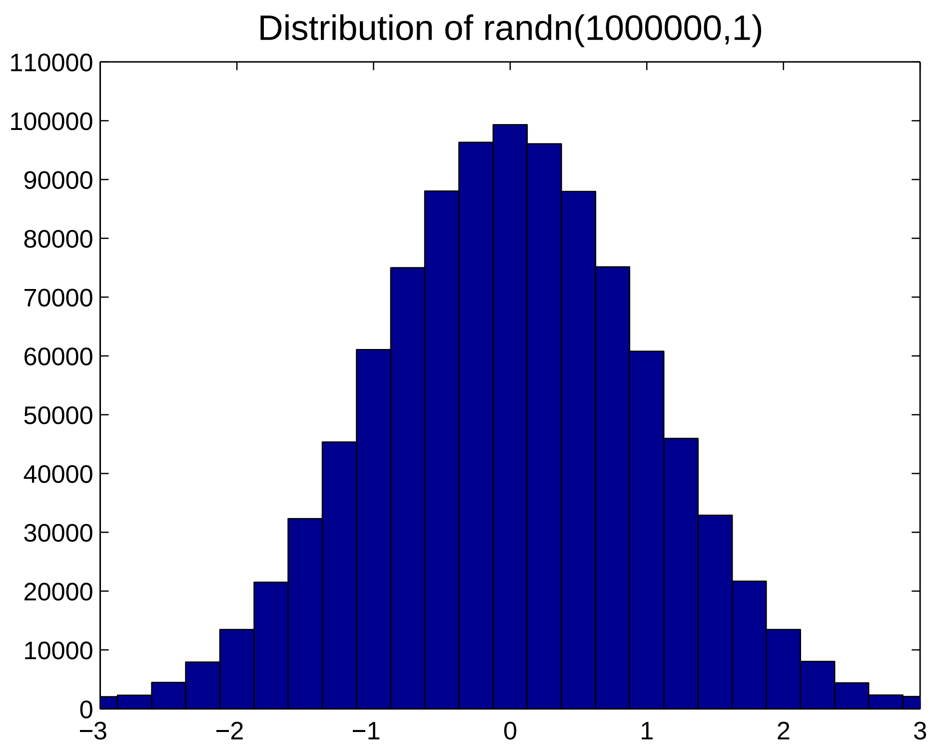


Normal distribution with zero mean and unit standard deviation

randn

Uniform probability distribution in (0,1)

rand



Sanity check: rand and randn

```
>> n= 1000000;
>> x= rand(n,1);
>> ave= sum(x)/n
ave =
    0.5004
```

```
>> y= randn(n,1);
>> ave= sum(y)/n
ave =
    0.0018
>> stdDev= std(y)
stdDev =
    1.0001
```

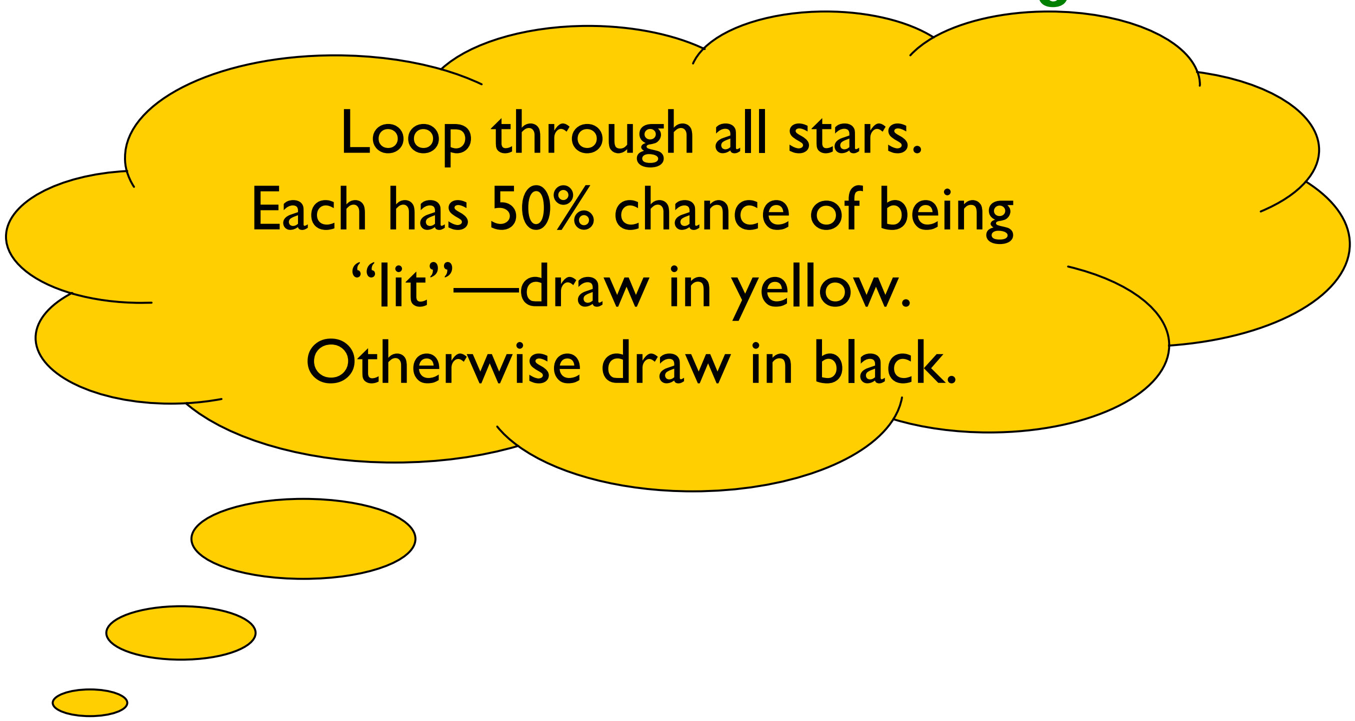
Simulate twinkling stars

- Get 10 user mouse clicks as locations of 10 stars—our constellation
- Simulate twinkling
  - Loop through all the stars; each has equal likelihood of being bright or dark
  - Repeat many times

- Can use DrawStar, DrawRect

```
% No. of stars and star radius
  N=10;  r=.5;
% Get mouse clicks, store coords in vectors x,y
  [x,y] = ginput(N);
% Twinkle!
  for k= 1:20  % 20 rounds of twinkling







  end
```

```
% No. of stars and star radius
  N=10;  r=.5;
% Get mouse clicks, store coords In vectors x,y
  [x,y] = ginput(N);
% Twinkle!
  for k= 1:20  % 20 rounds of twinkling

  end
```



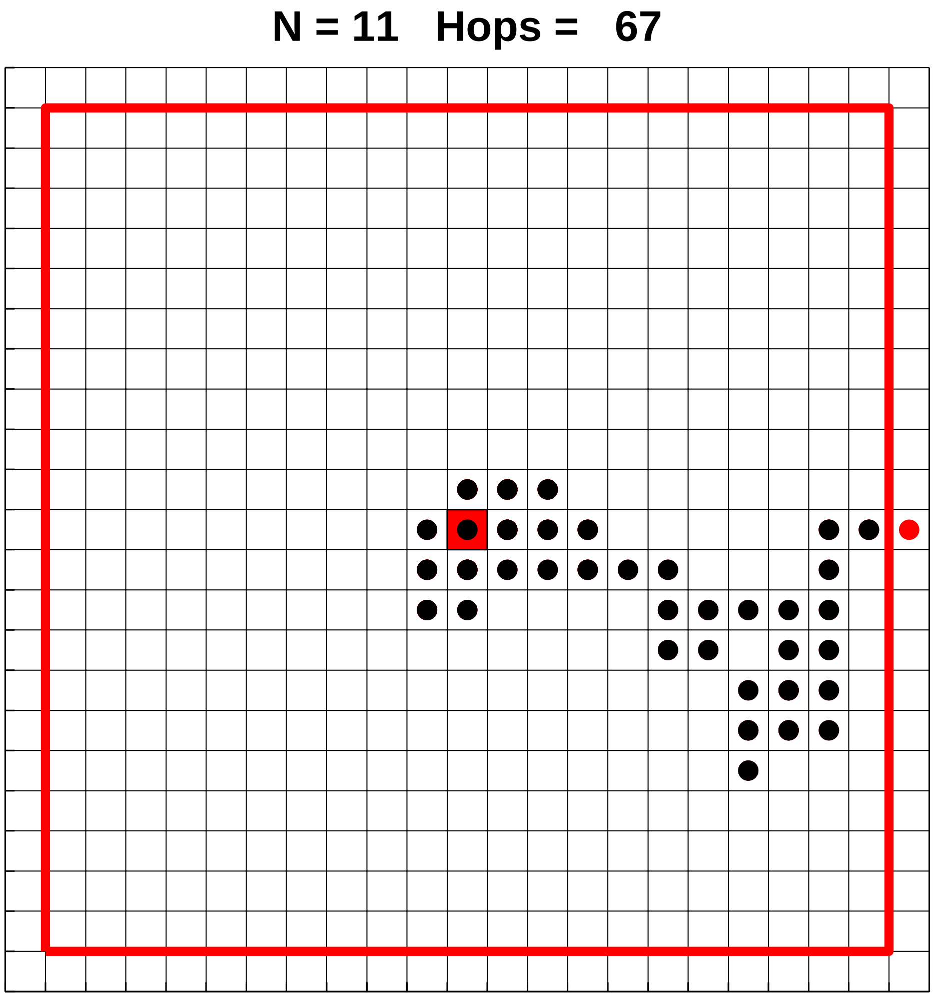
Twinkle.m

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

```
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
```

```
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

end
```

```
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
```

RandomWalk2D.m

Another representation for the random step

- Observe that each update has the form

$$xc = xc + \Delta x$$

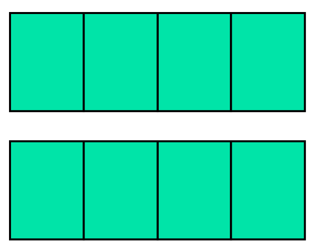
$$yc = yc + \Delta y$$

no matter which direction is taken.

- So let's get rid of the if statement!
- Need to create two "change vectors" deltaX and deltaY

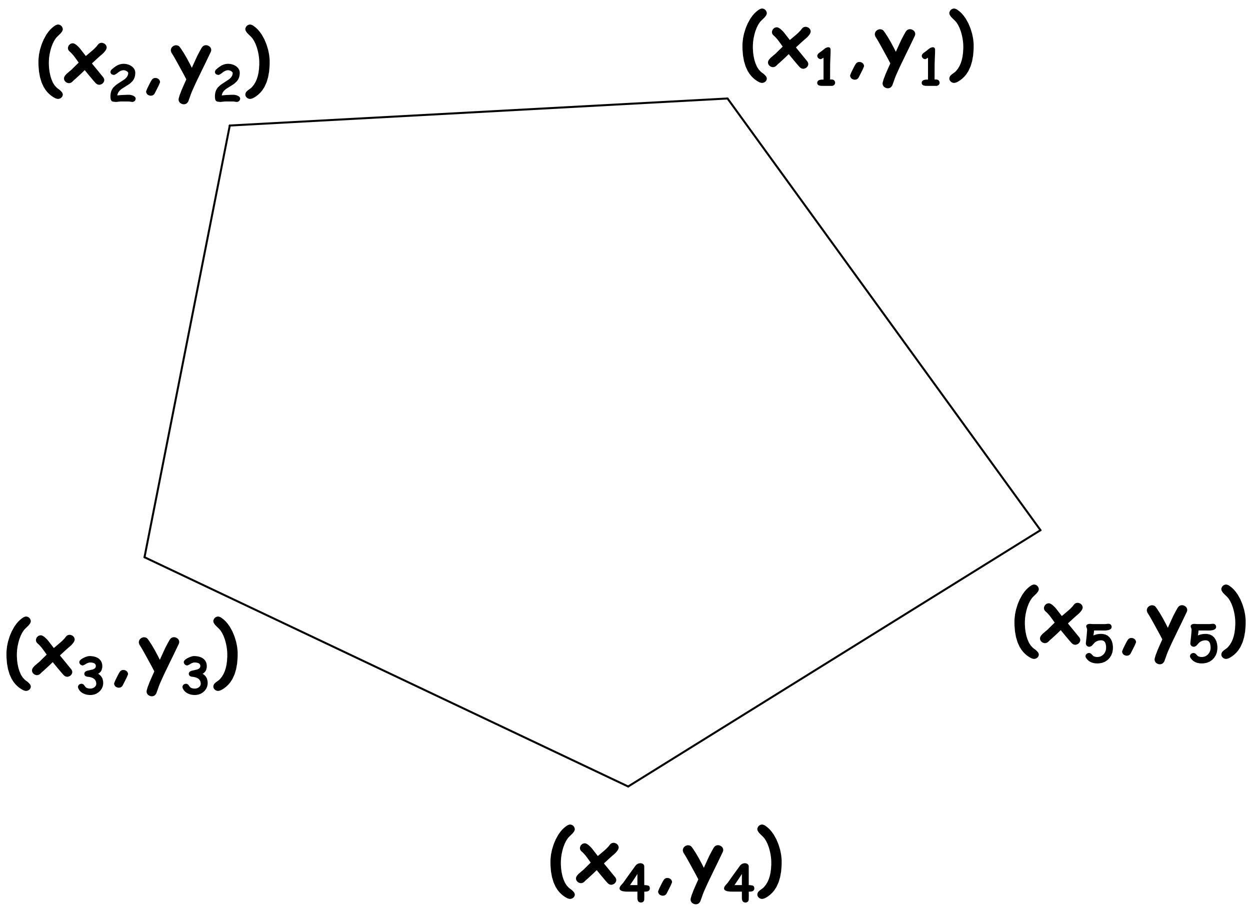
deltaX

deltaY

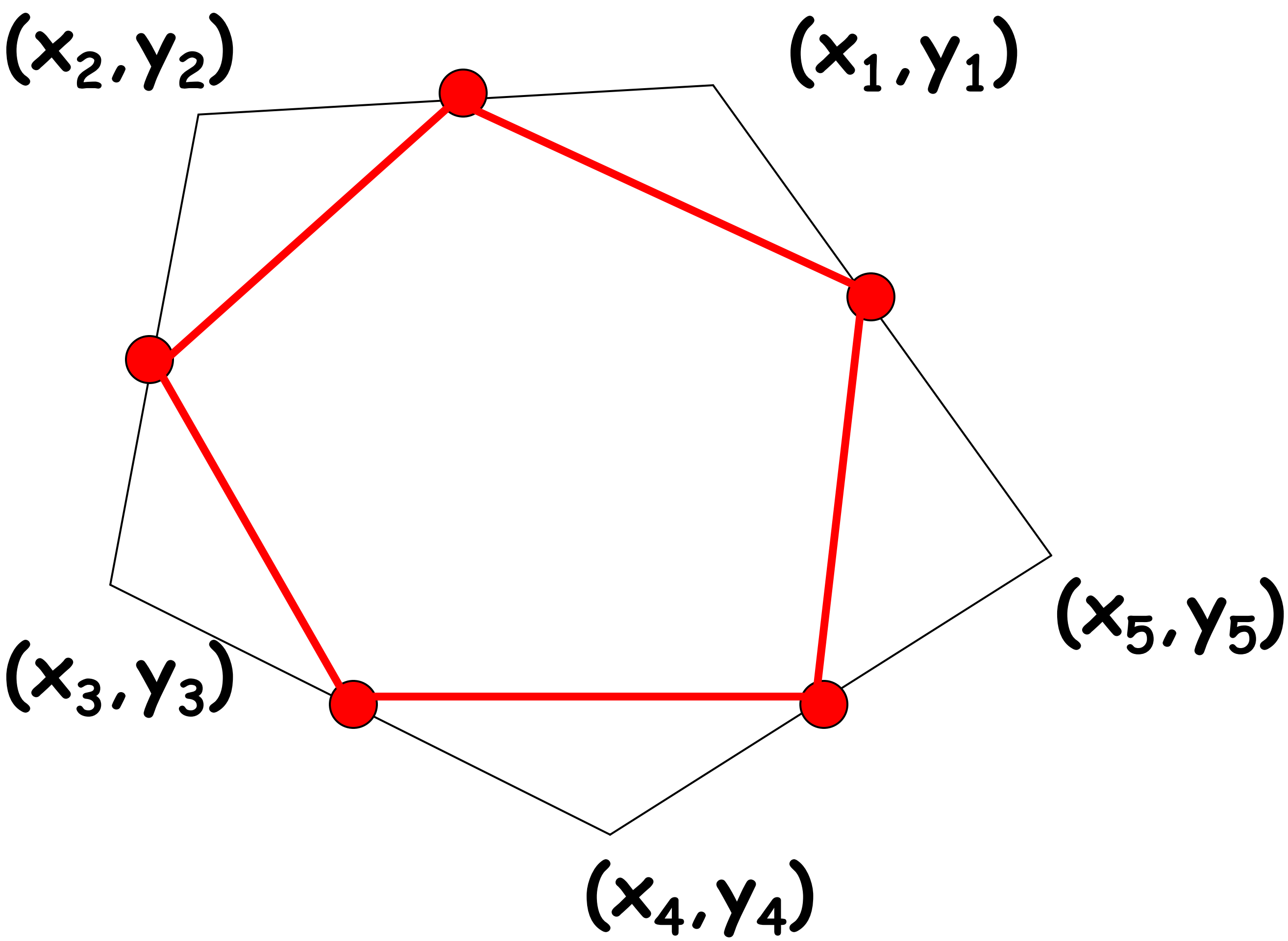


RandomWalk2D_v2.m

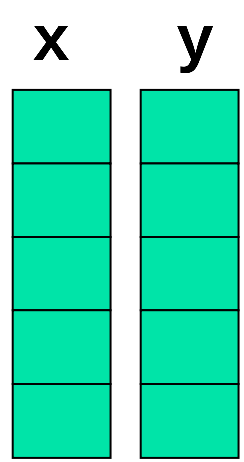
Example: polygon smoothing



Example: polygon smoothing

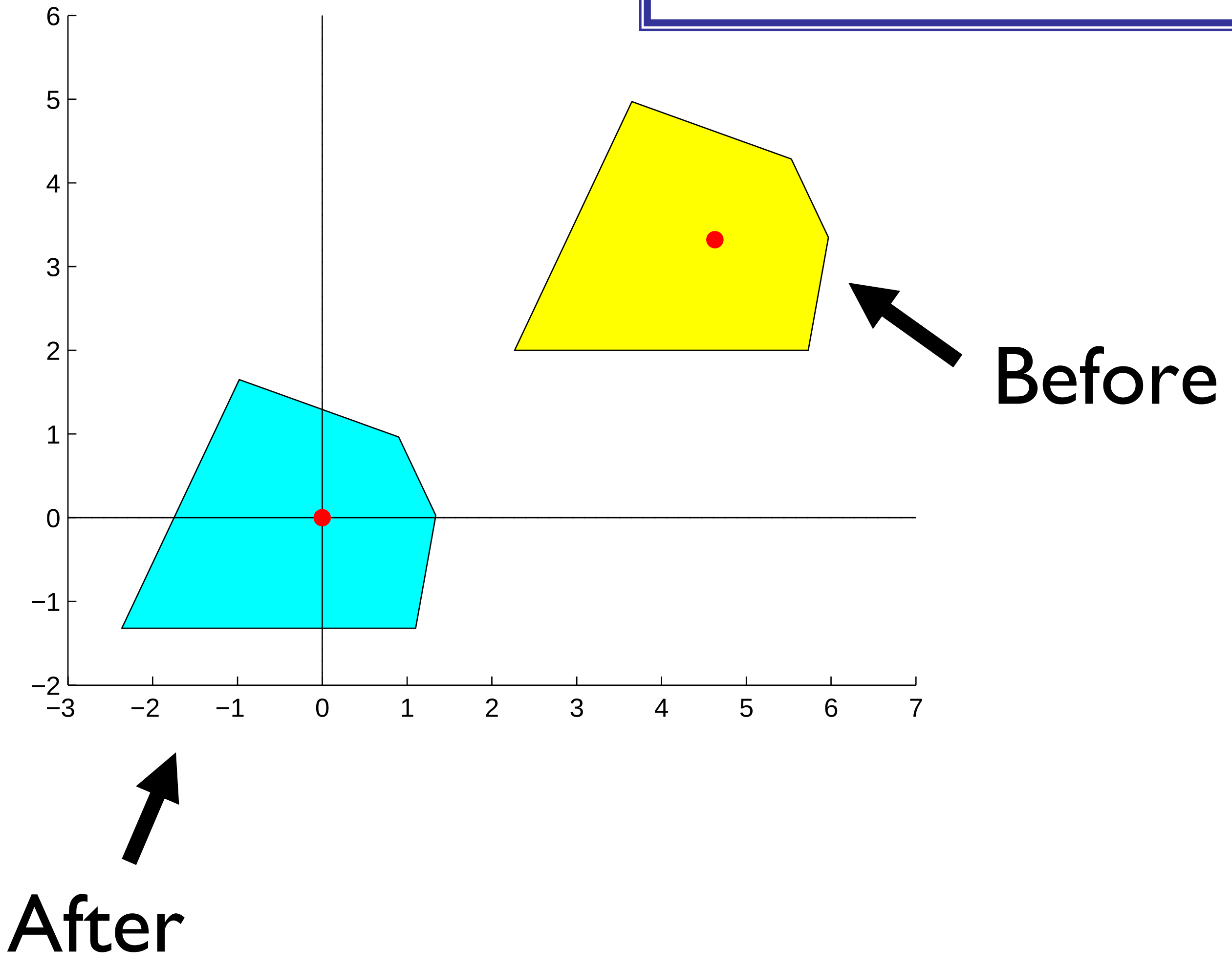


Can store the x-y coordinates in vectors x and y



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.

n = length(x);
xBar = sum(x)/n;
yBar = sum(y)/n;
xNew = x-xBar;
yNew = y-yBar;
```

Vectorized code

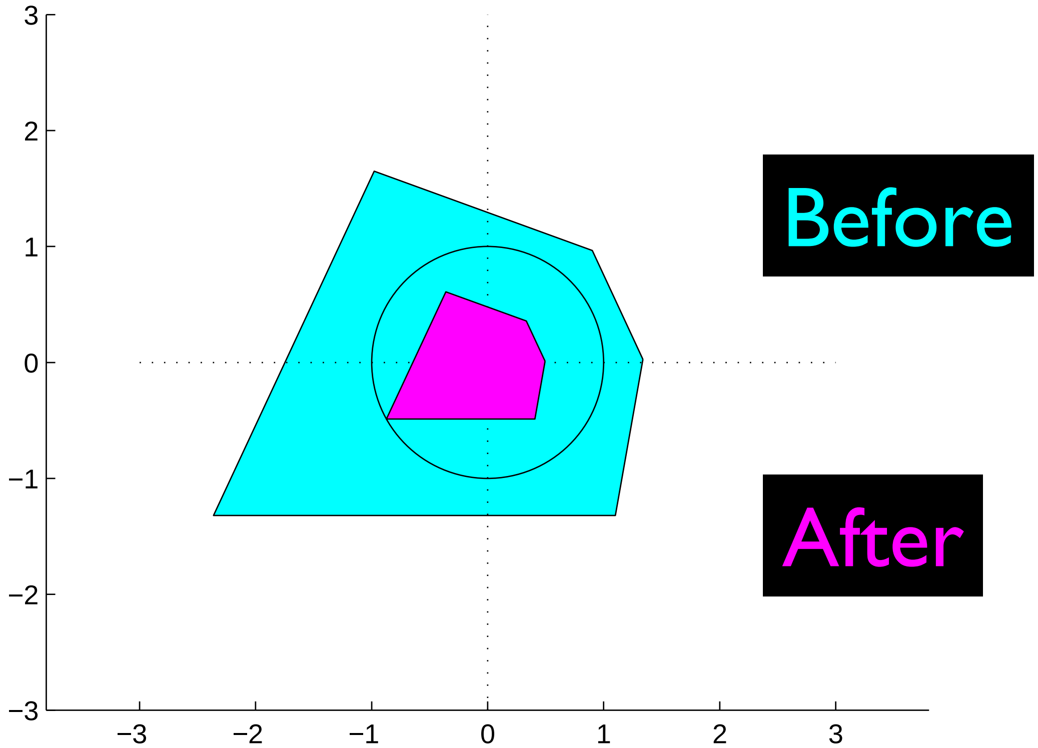
```
function [xNew,yNew] = Centralize(x,y)
% Translate polygon defined by vectors
% x,y such that the centroid is on the
% origin. New polygon defined by vectors
% xNew,yNew.

n = length(x);
xBar = sum(x)/n;
yBar = sum(y)/n;
xNew = x-xBar;
yNew = y-yBar;
```

Vectorized code

```
xNew = zeros(n,1);
yNew = zeros(n,1);
for k=1:n
    xNew(k) = x(k)-xBar;
    yNew(k) = y(k)-yBar;
end
```

Second operation: normalize

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



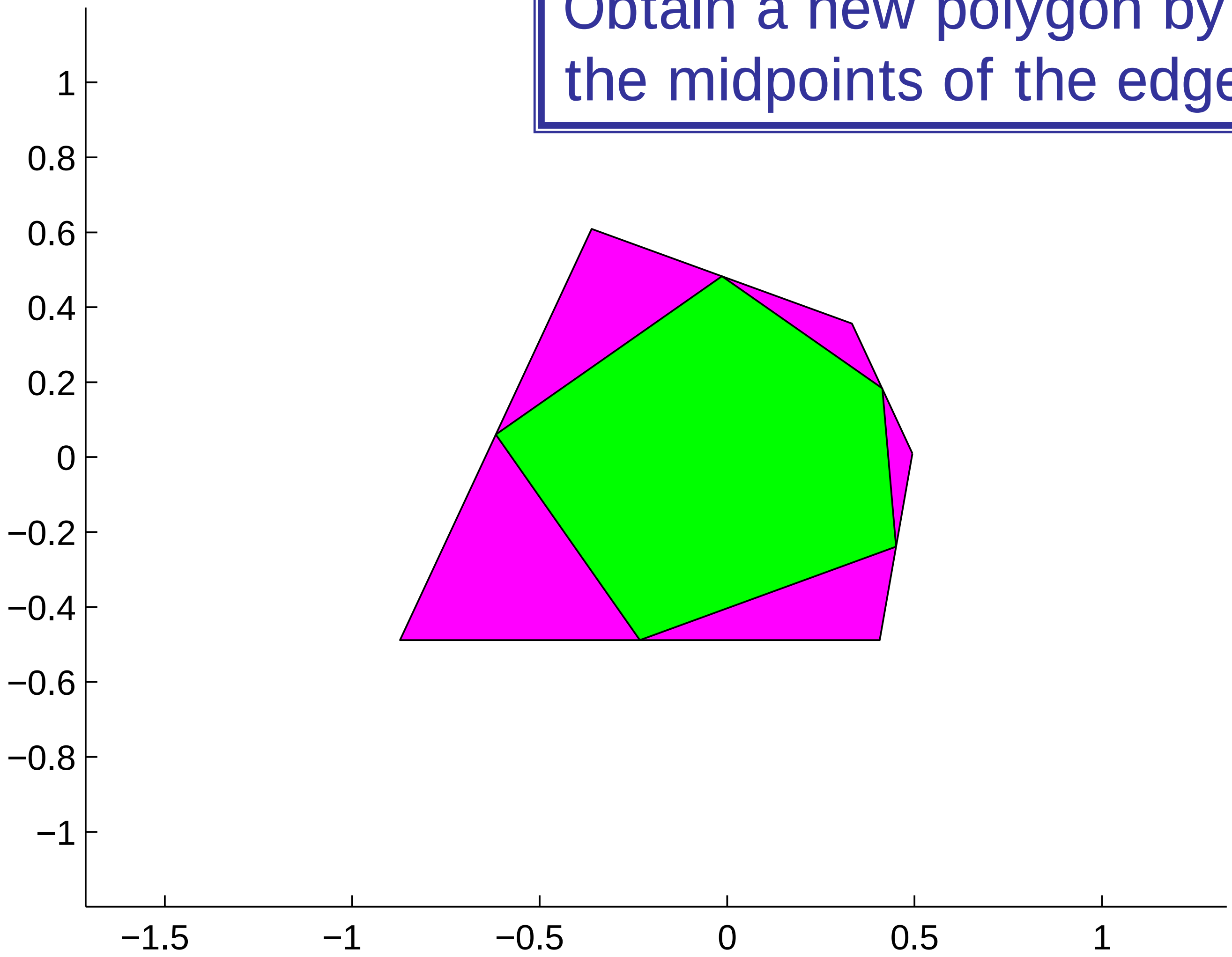
```
function [xNew,yNew] = Normalize(x,y)
% Resize polygon defined by vectors x,y
% such that distance of the vertex
% furthest from origin is 1

 d = max(sqrt(x.^2 + y.^2));
 xNew = x/d;
 yNew = y/d;
```

Vectorized ops

Applied to a vector, **max** returns the largest value in the vector

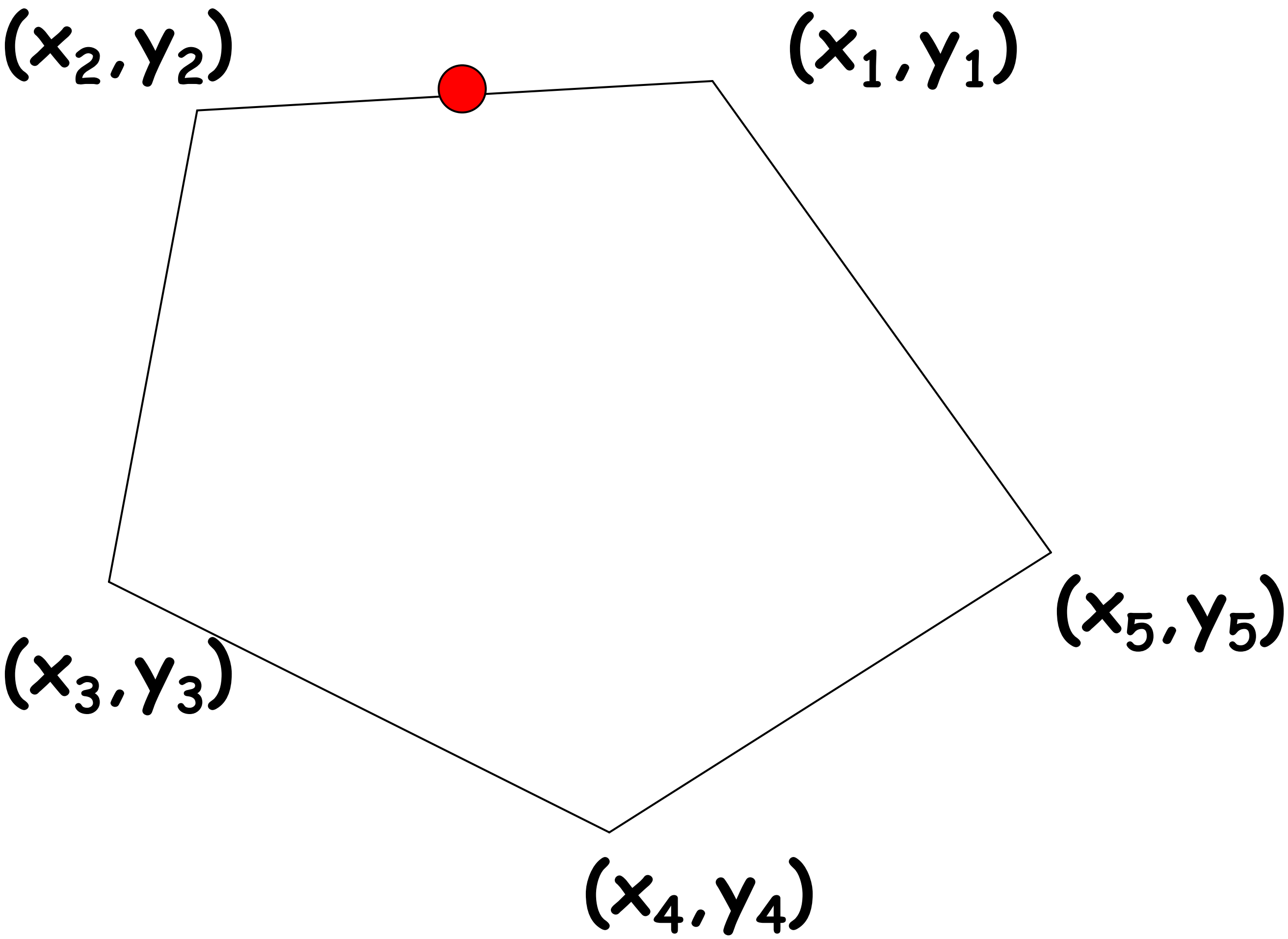
Third operation: smooth

Obtain a new polygon by connecting the midpoints of the edges

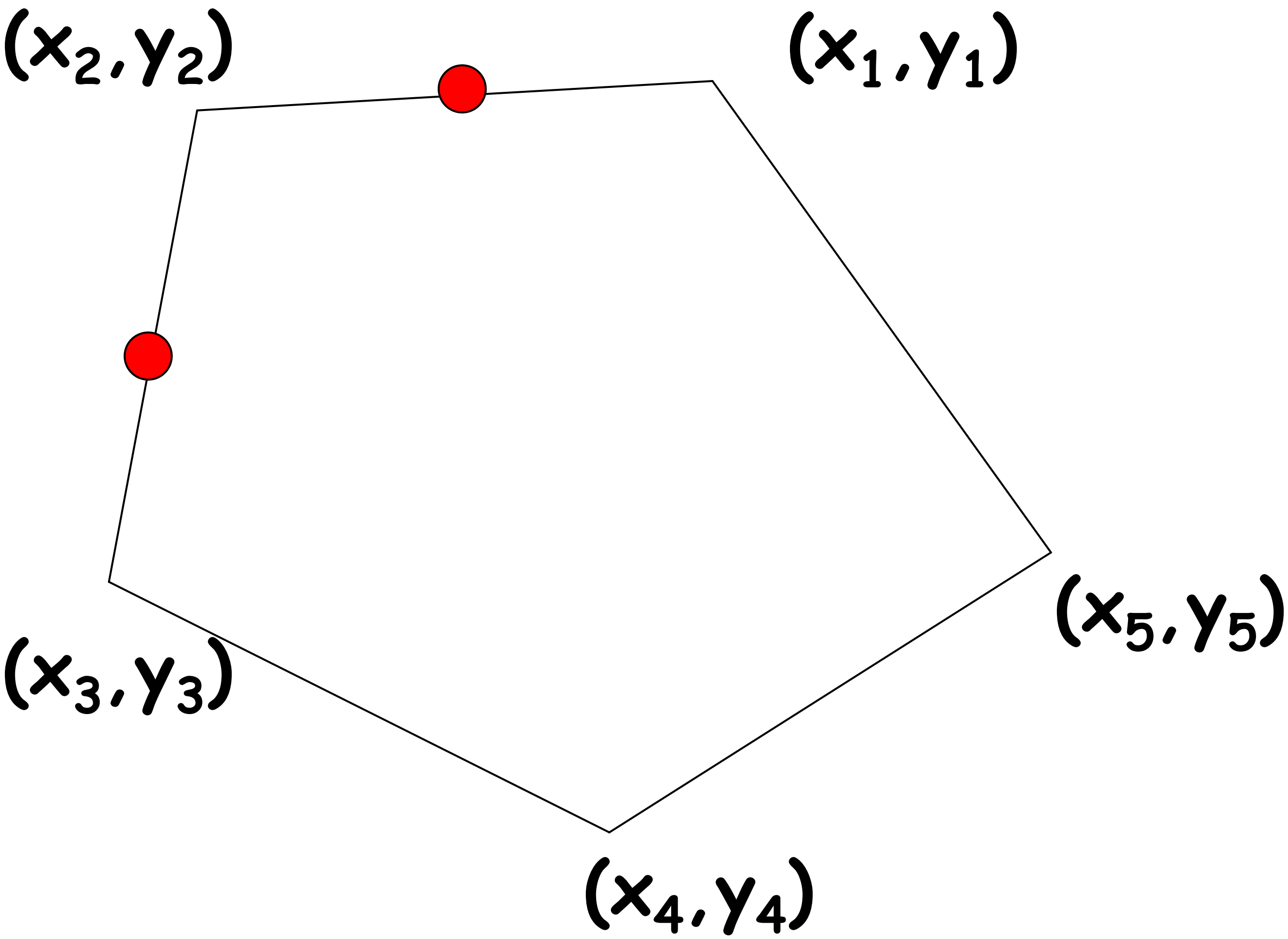


```
function [xNew,yNew] = Smooth(x,y)
% Smooth polygon defined by vectors x,y
% by connecting the midpoints of
% adjacent edges

n = length(x);
xNew = zeros(n,1);
yNew = zeros(n,1);

for i=1:n
   Compute the midpt of ith edge.
   Store in xNew(i) and yNew(i)
end
```

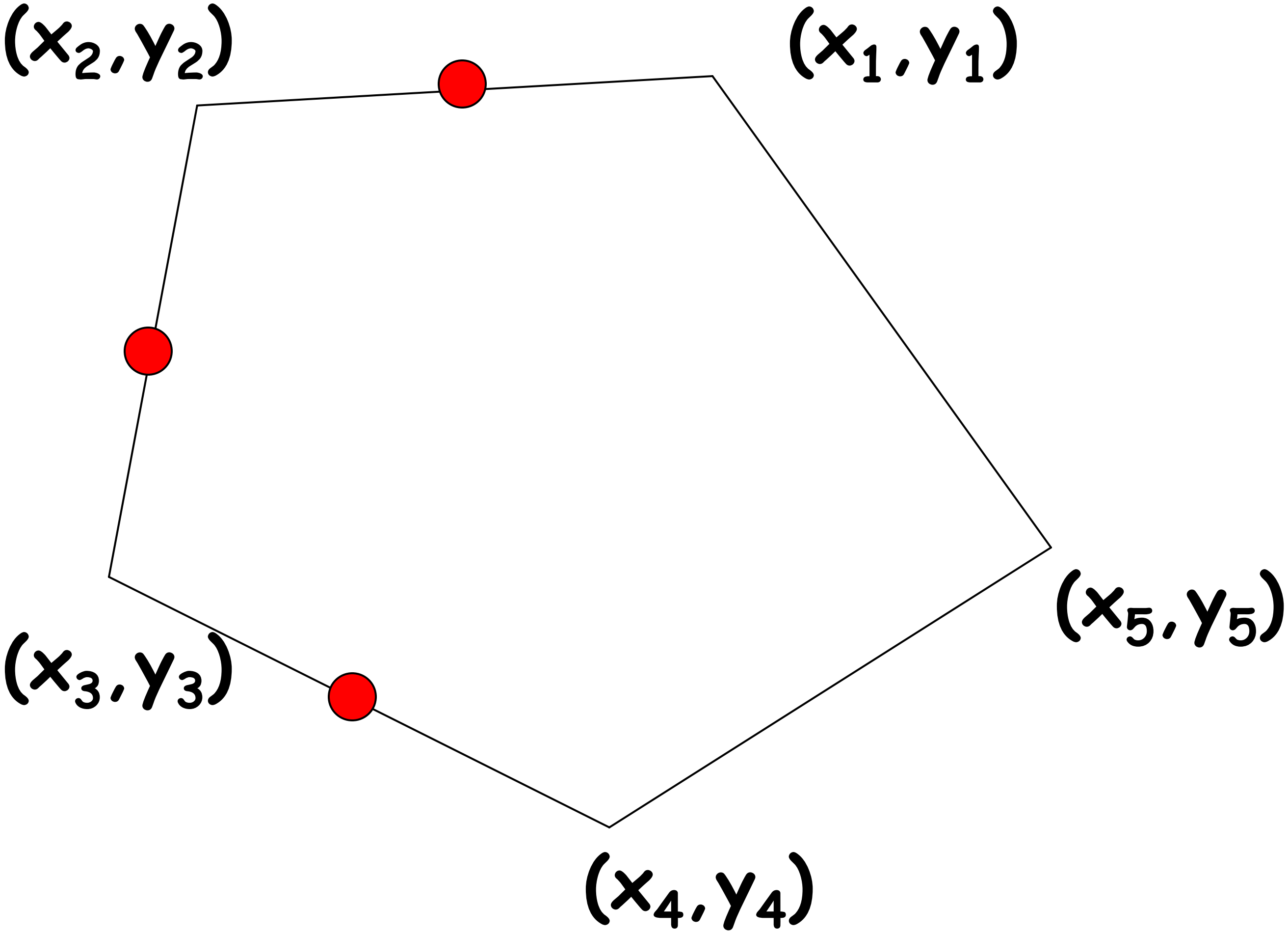
```
xNew(1) = (x(1)+x(2))/2
yNew(1) = (y(1)+y(2))/2
```



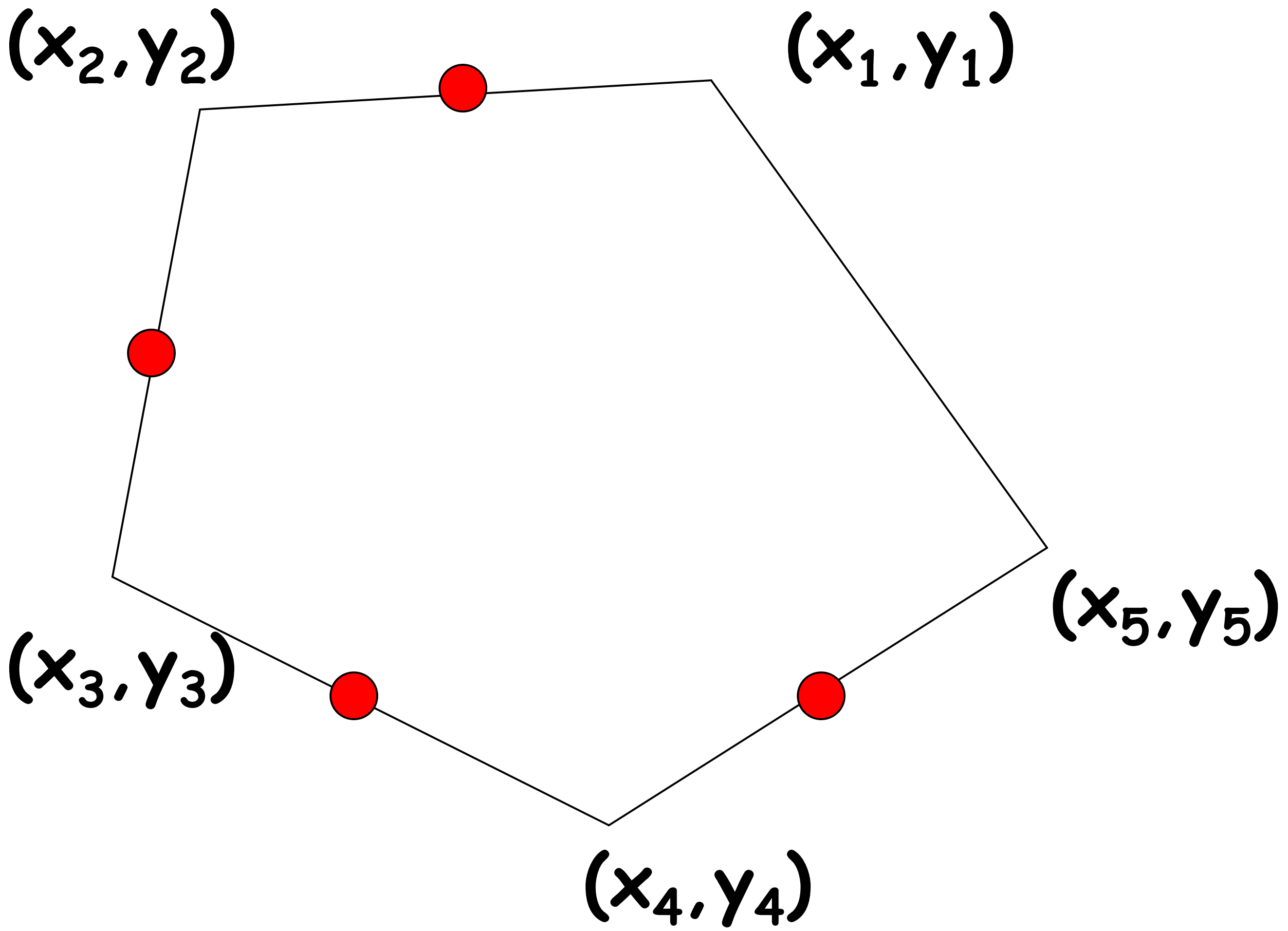
```
xNew(2) = (x(2)+x(3))/2
yNew(2) = (y(2)+y(3))/2
```



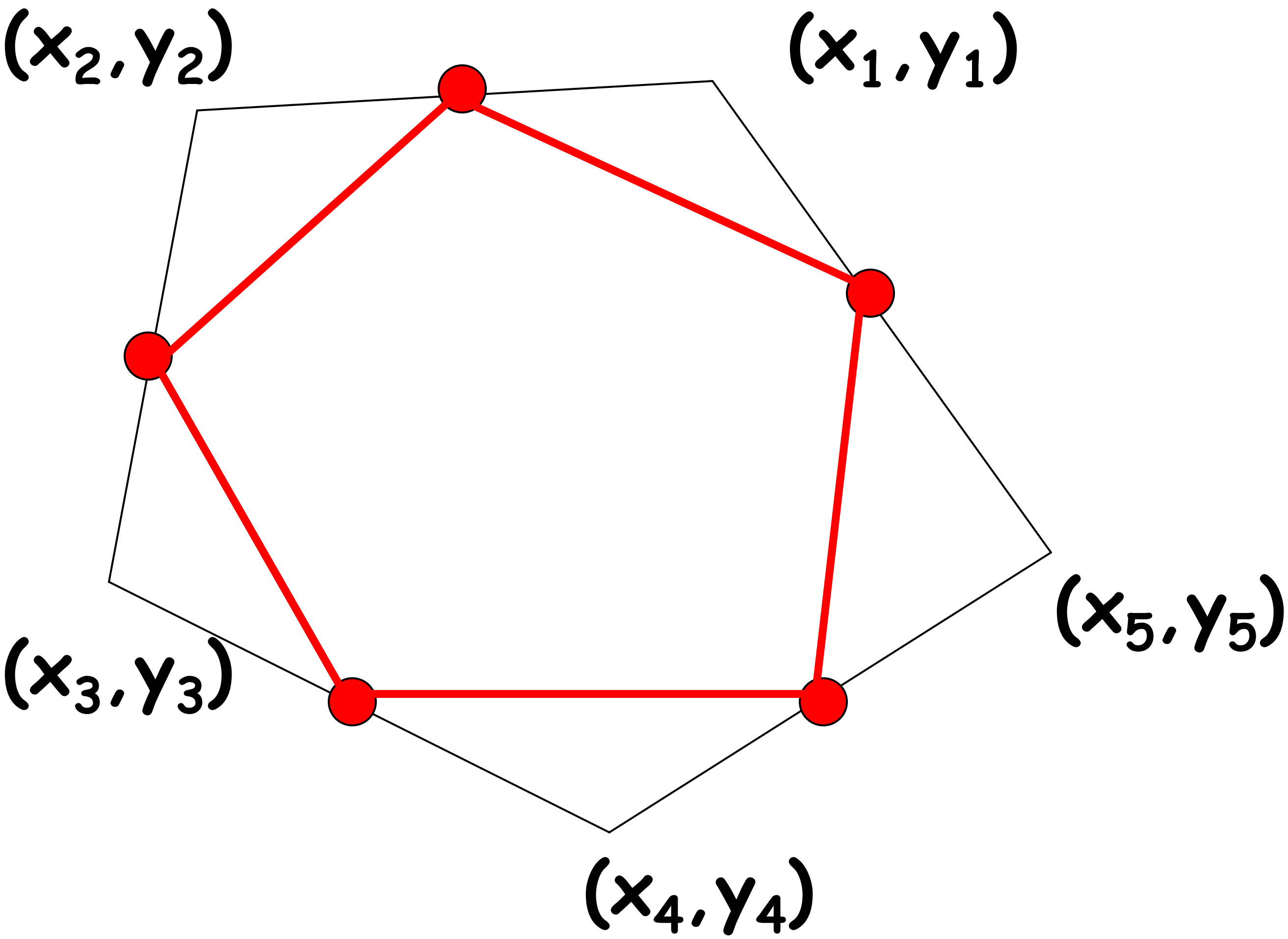
```
xNew(3) = (x(3)+x(4))/2
yNew(3) = (y(3)+y(4))/2
```



```
xNew(4) = (x(4)+x(5))/2
yNew(4) = (y(4)+y(5))/2
```



```
xNew(5) = (x(5)+x(1))/2
yNew(5) = (y(5)+y(1))/2
```



Smooth

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

Will result in a subscript
out of bounds error when i is n.

Smooth

```
for i=1:n
   if i<n
      xNew(i) = (x(i) + x(i+1))/2;
      yNew(i) = (y(i) + y(i+1))/2;
   else
      xNew(n) = (x(n) + x(1))/2;
      yNew(n) = (y(n) + y(1))/2;
   end
end
```

Smooth

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

Show a simulation of polygon smoothing

Create a polygon with randomly located vertices.

Repeat:

- Centralize
- Normalize
- Smooth

ShowSmooth.m