

- Previous Lecture:

- Probability and random numbers
- 1-d array—vector

- Today's Lecture:

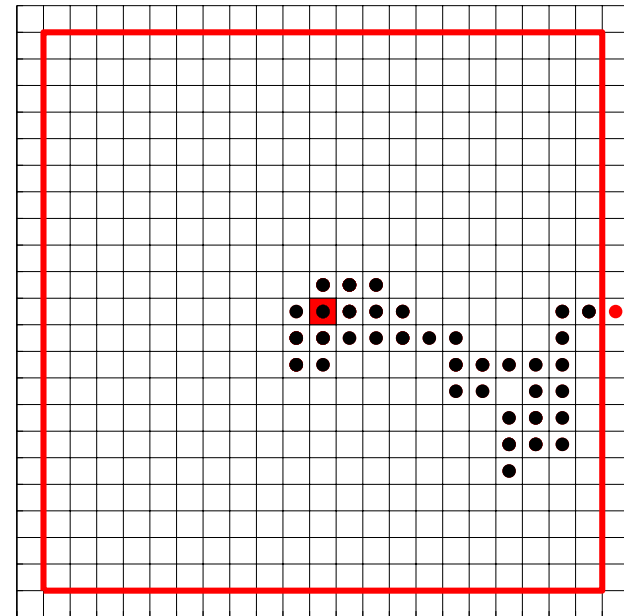
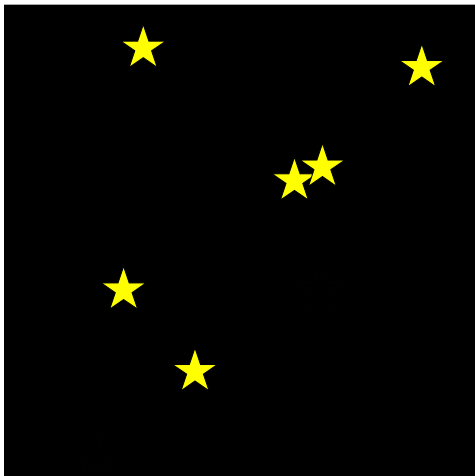
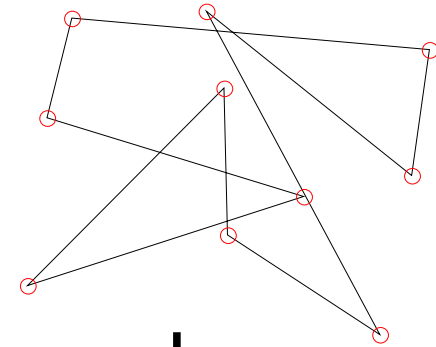
- More examples on vectors
- Simulation

- Announcement:

- Discussion this week in computer lab UP B7
- Project 3 due Oct 14. Use the lab computers if the simulator doesn't work with your computer.

# Simulation

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



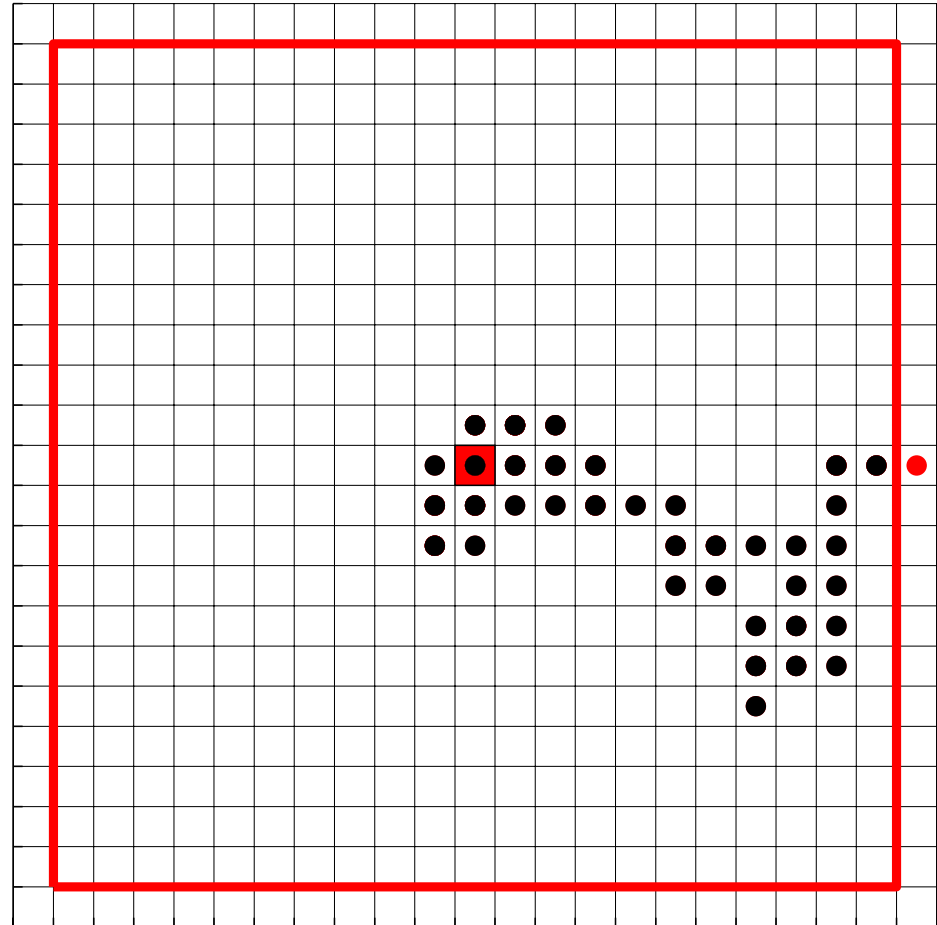
## 2-dimensional random walk

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  $1 \times 1$ .

Walk until you reach  
the boundary.

N = 11 Hops = 67



# RandomWalk2D.m

## Another representation for the random step

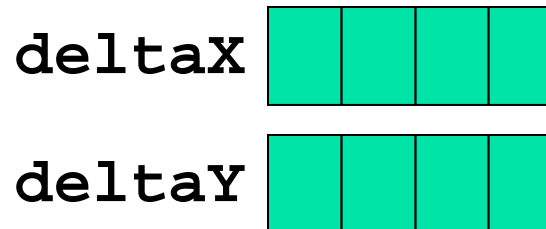
- Observe that each update has the form

$$x_c = x_c + \Delta x$$

$$y_c = y_c + \Delta y$$

no matter which direction is taken.

- So let's get rid of the if statement!
- Need to create two “change vectors”  $\Delta x$  and  $\Delta y$



# RandomWalk2D\_v2.m

## 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;

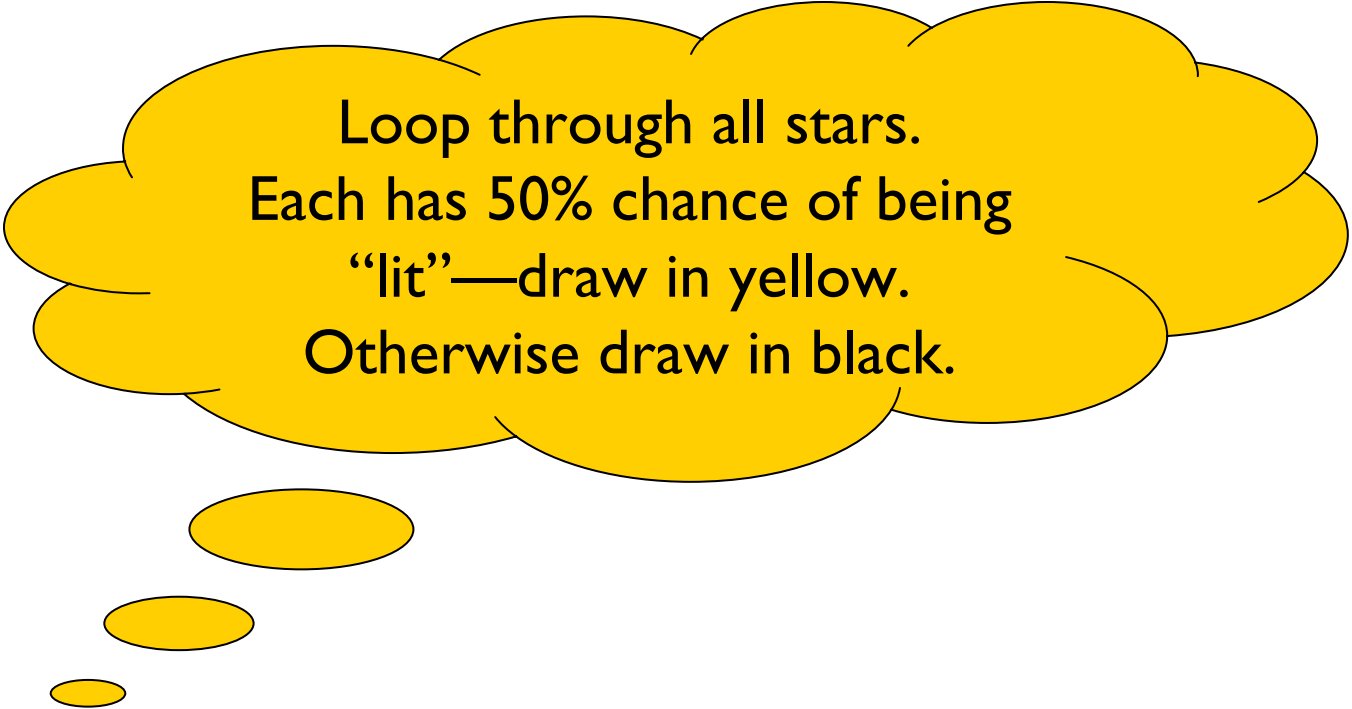
% Get mouse clicks, store coords in vectors x,y
[x,y] = ginput(N);

% 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
```

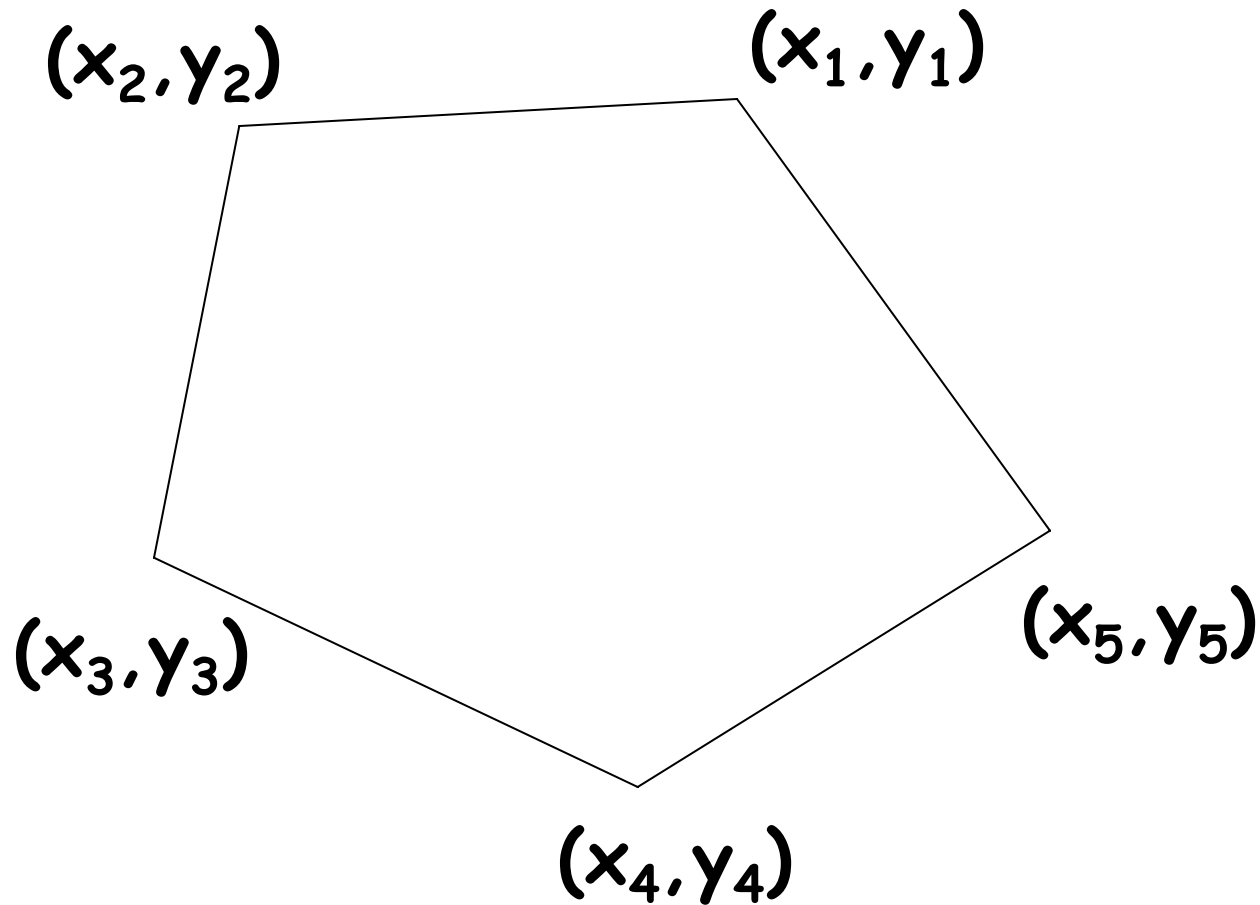


Loop through all stars.  
Each has 50% chance of being  
“lit”—draw in yellow.  
Otherwise draw in black.

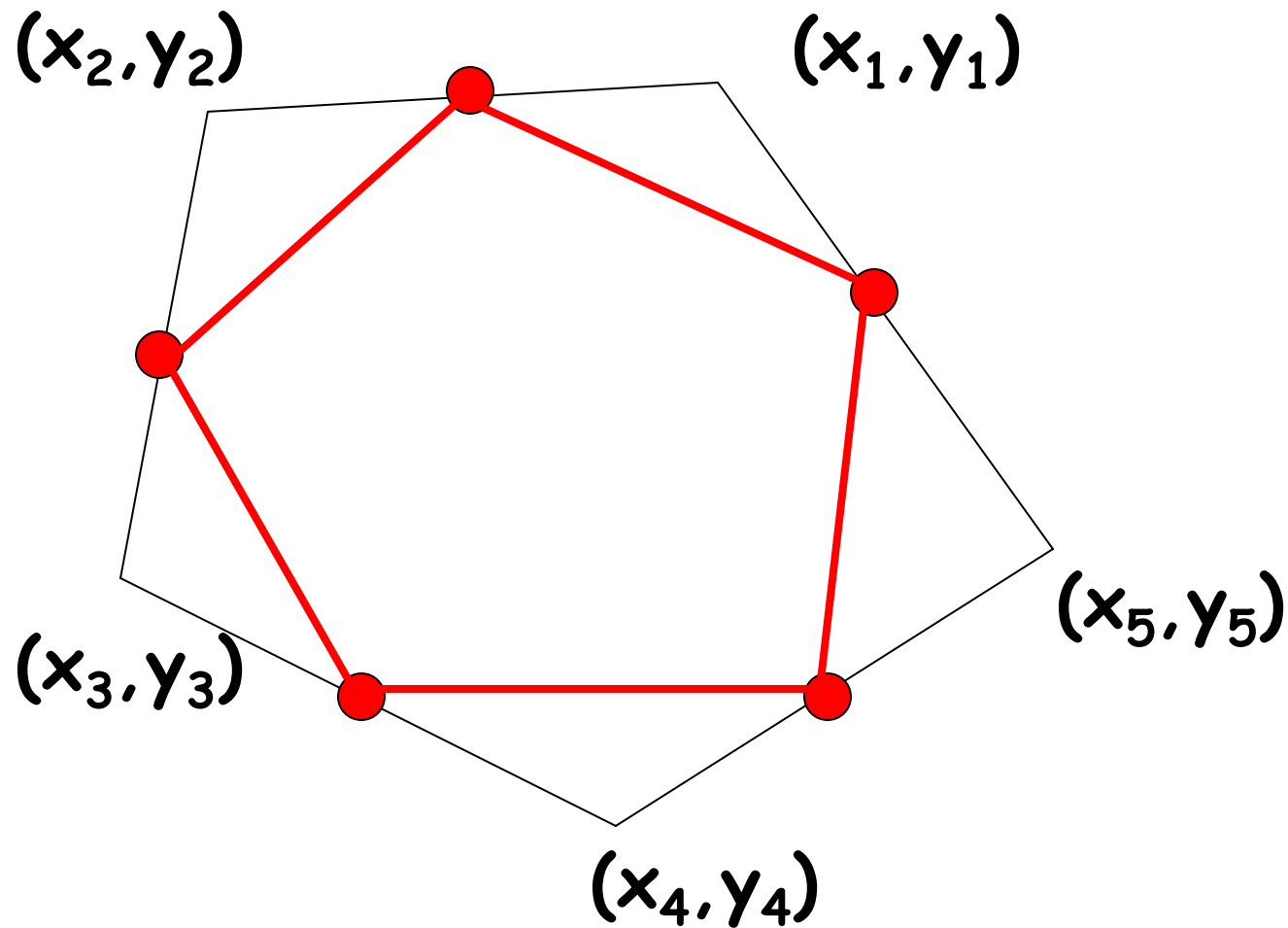
```
end
```

# Twinkle.m

## Example: polygon smoothing



## Example: polygon smoothing

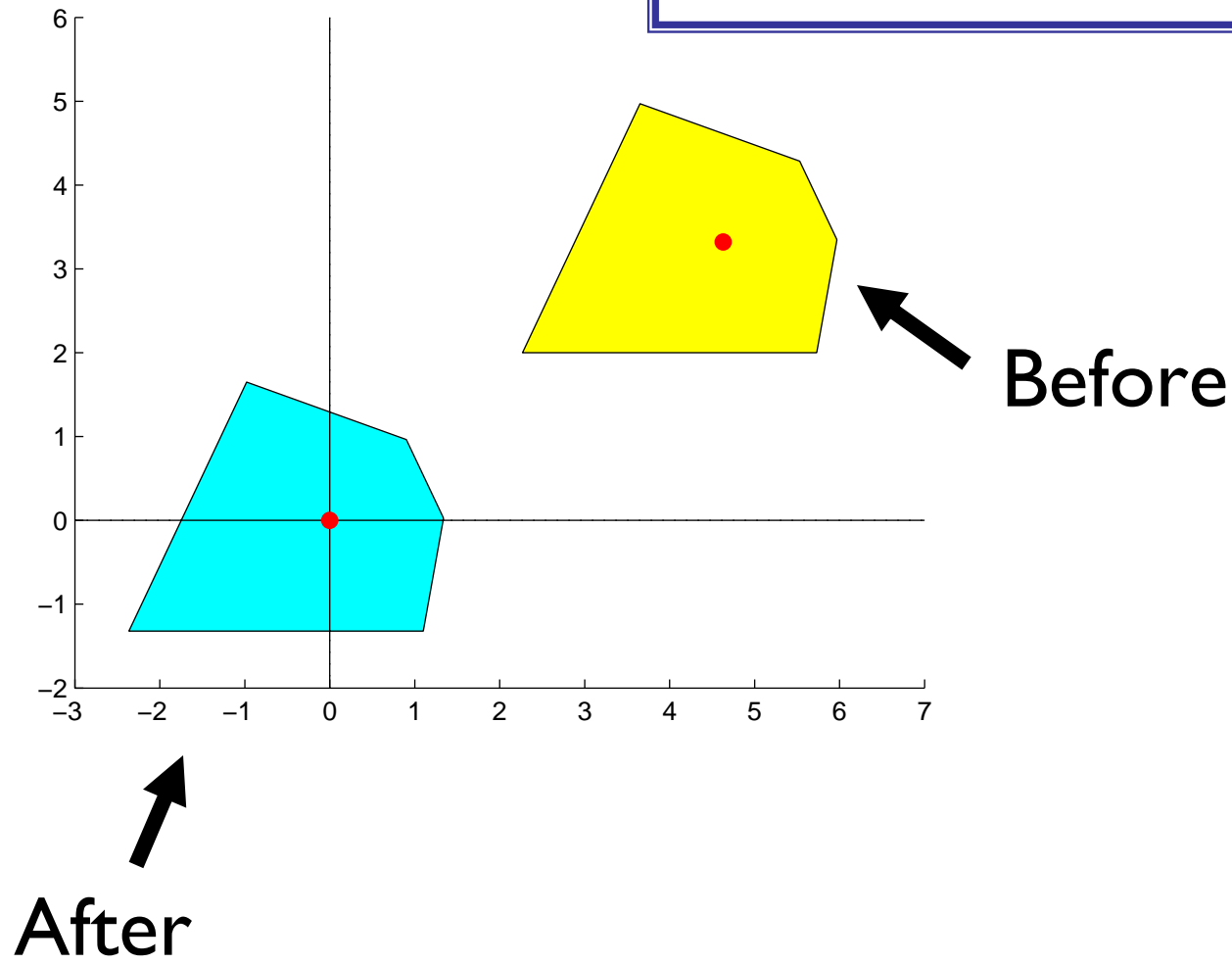


Can store the x-y coordinates in vectors  $x$  and  $y$

| $x$ | $y$ |
|-----|-----|
|     |     |
|     |     |
|     |     |
|     |     |
|     |     |

# 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.
```

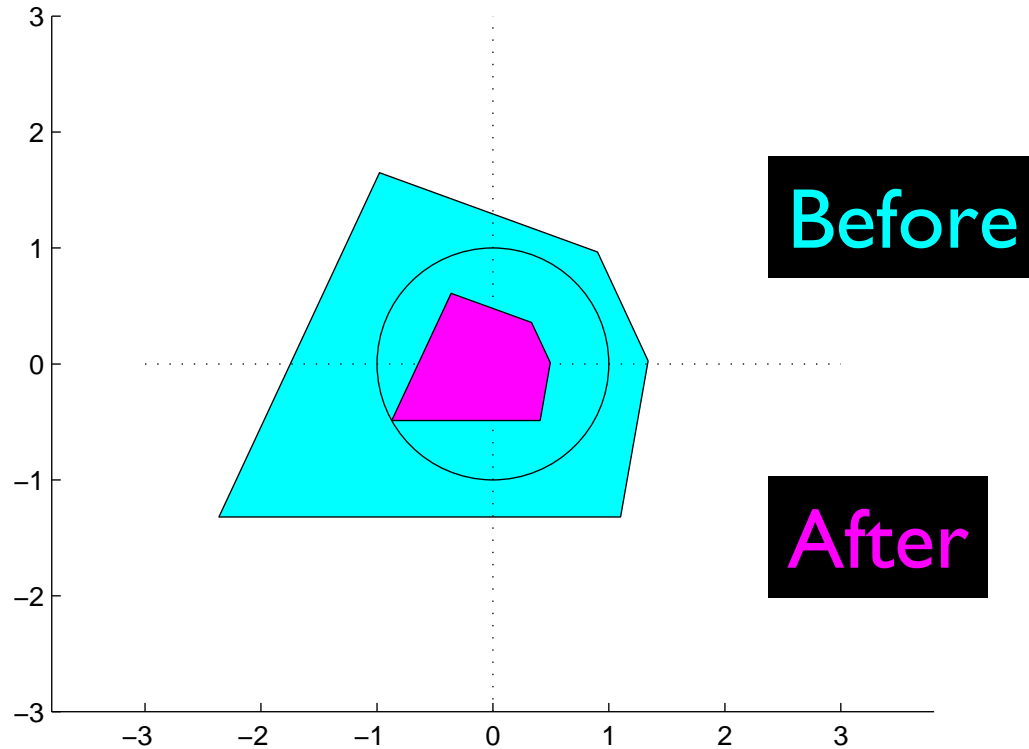
```
n = length(x);
xBar = sum(x)/n;
yBar = sum(y)/n;
xNew = x - xBar;
yNew = y - yBar;
```

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

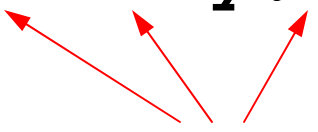
Shrink (enlarge) the polygon so that the vertex furthest from the  $(0,0)$  is on the unit circle





```
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;
```

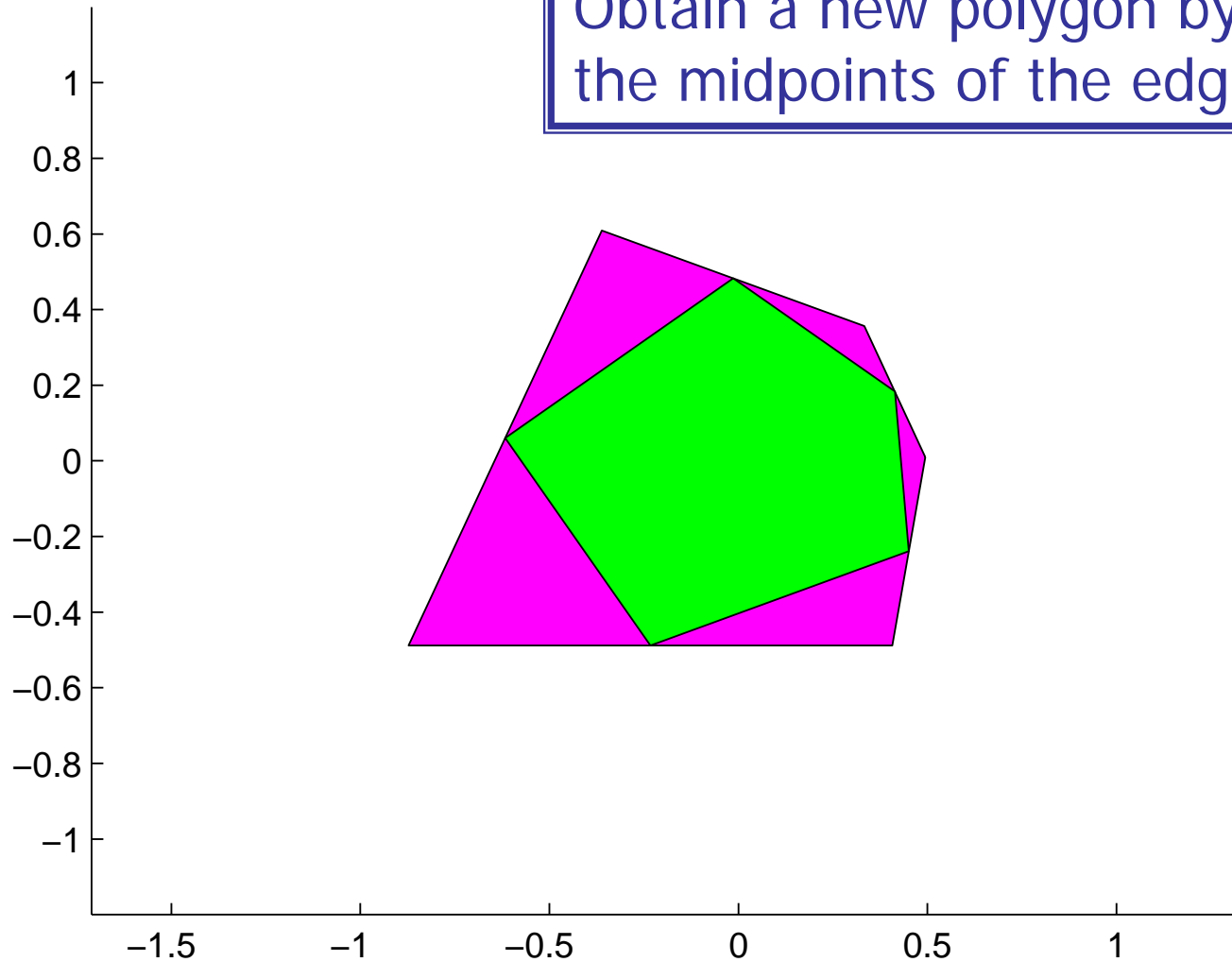


Vectorized ops

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



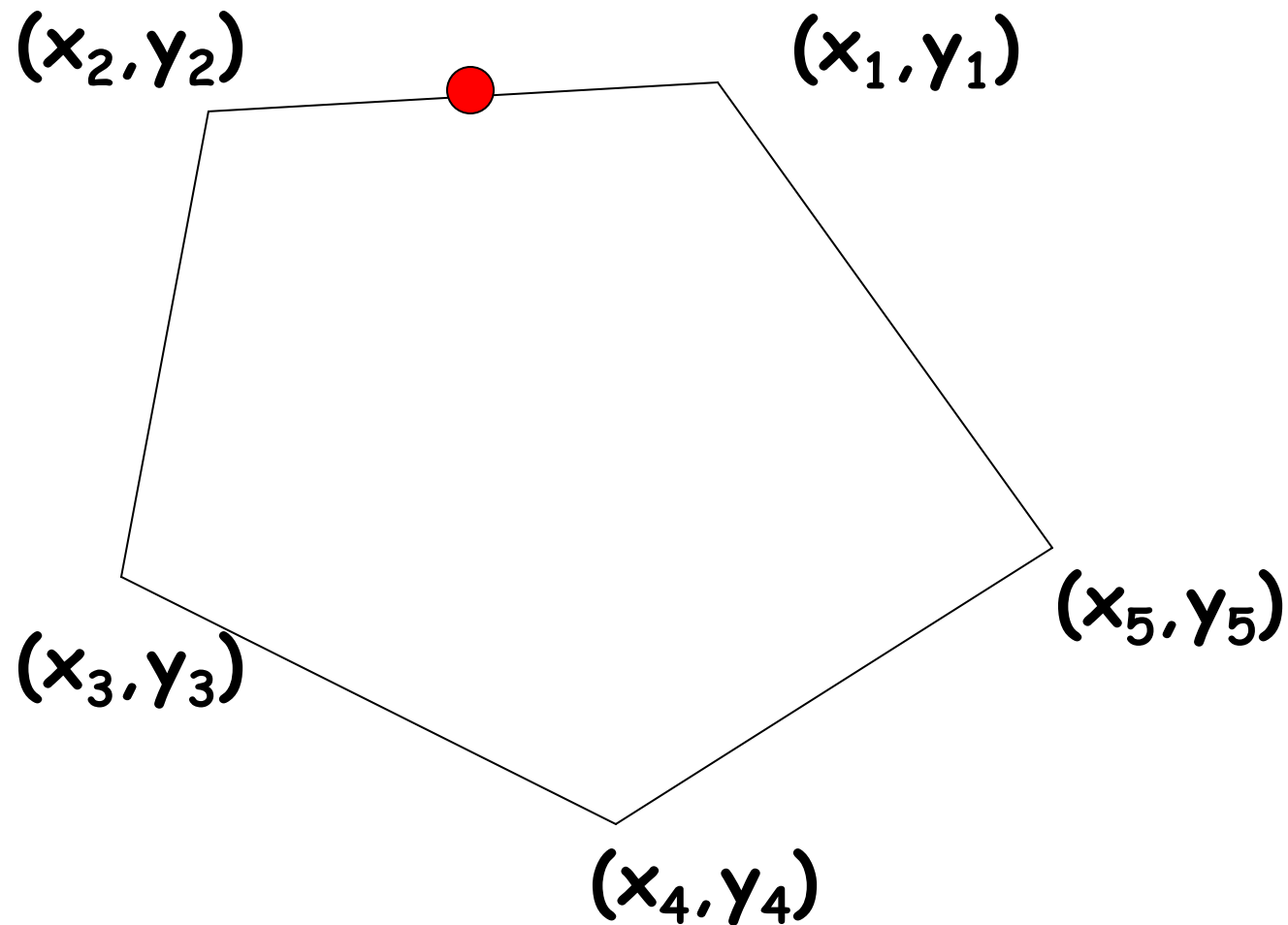
```
function [xNew,yNew] = Smooth(x,y)
% Smooth polygon defined by vectors x,y
% by connecting the midpoints of
% adjacent edges

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
```

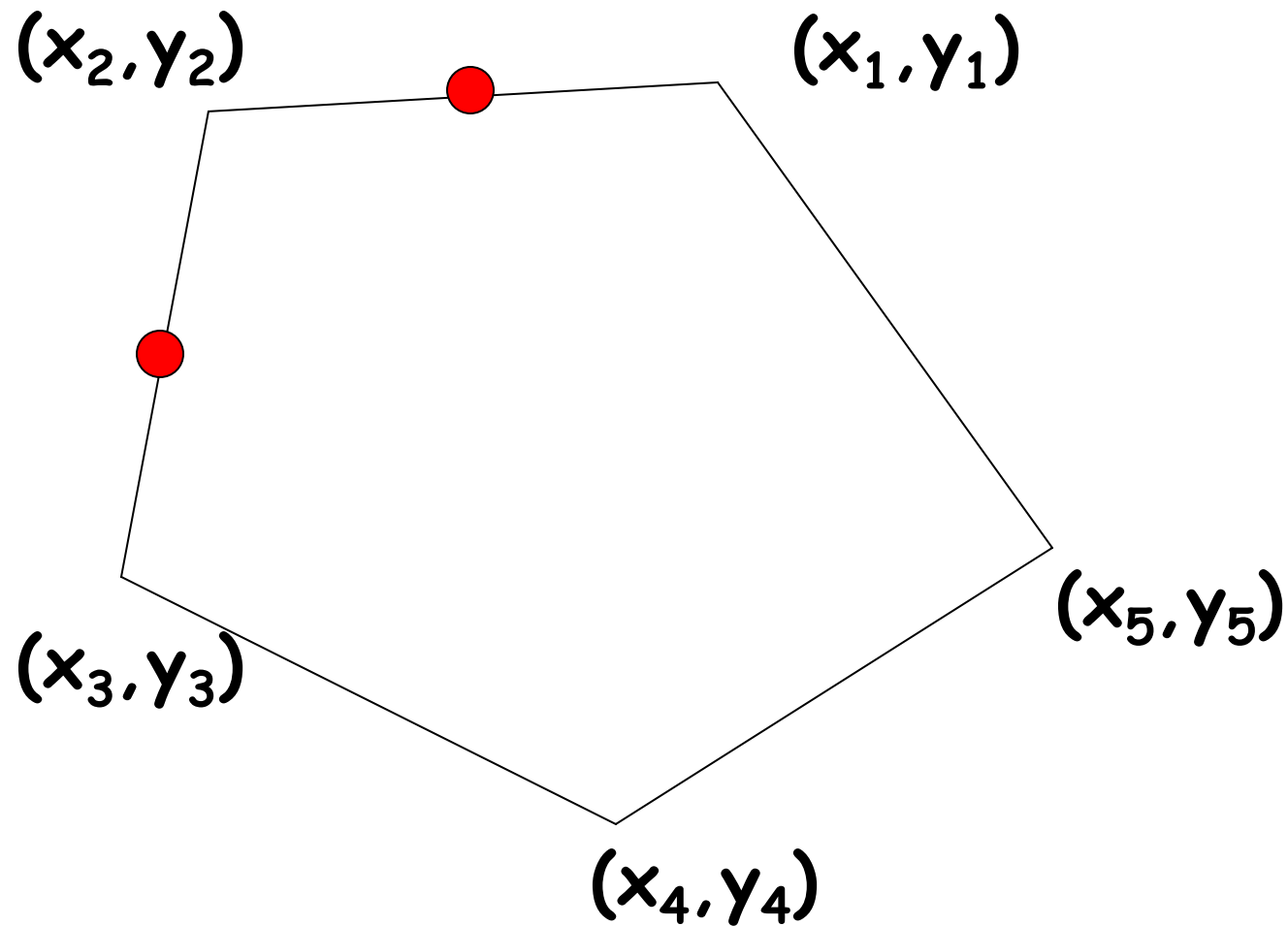
$$x_{\text{New}}(1) = (x(1) + x(2)) / 2$$

$$y_{\text{New}}(1) = (y(1) + y(2)) / 2$$



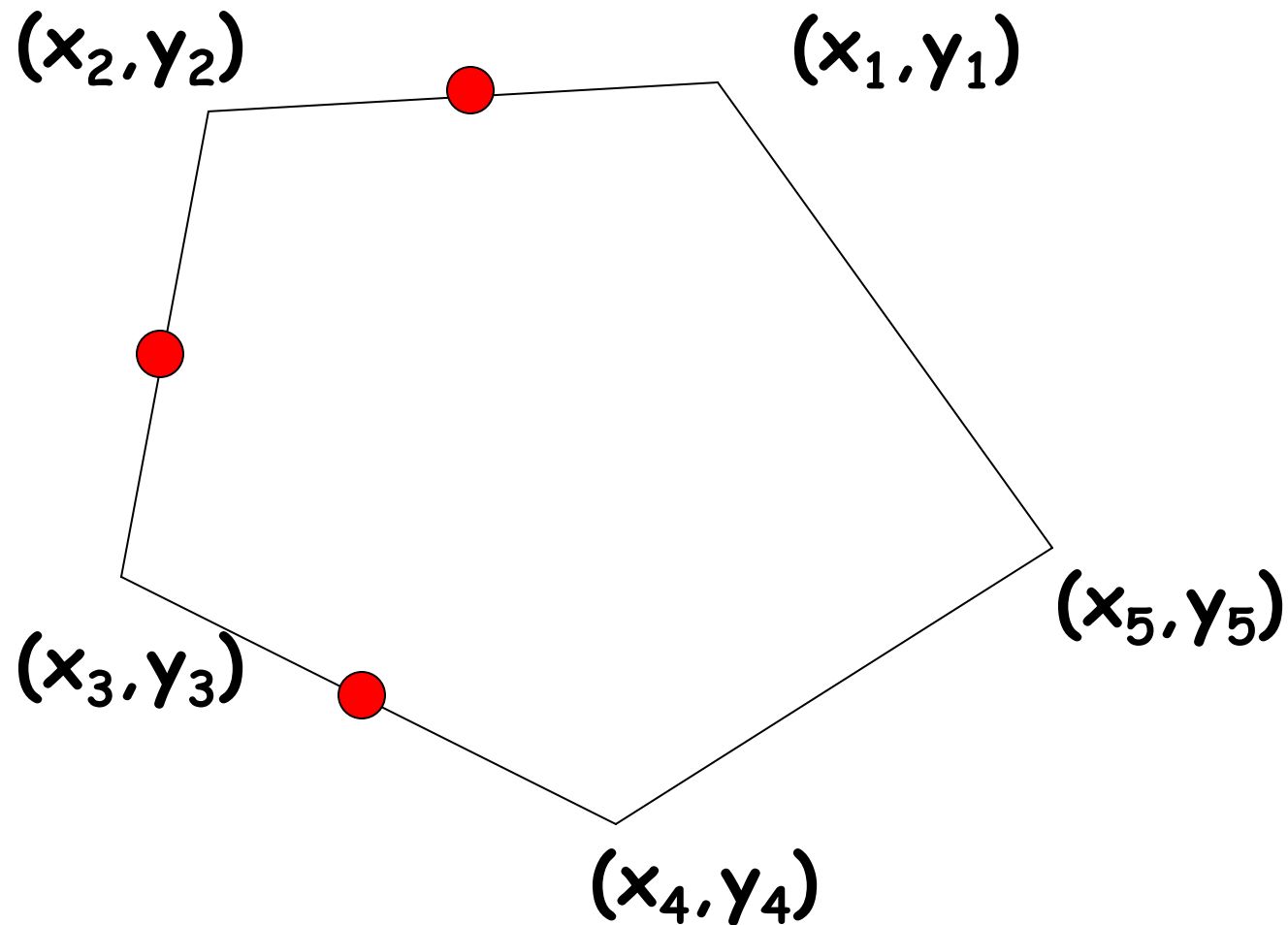
$$x_{\text{New}}(2) = (x(2) + x(3)) / 2$$

$$y_{\text{New}}(2) = (y(2) + y(3)) / 2$$



$$x_{\text{New}}(3) = (x(3) + x(4)) / 2$$

$$y_{\text{New}}(3) = (y(3) + y(4)) / 2$$



# 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
```

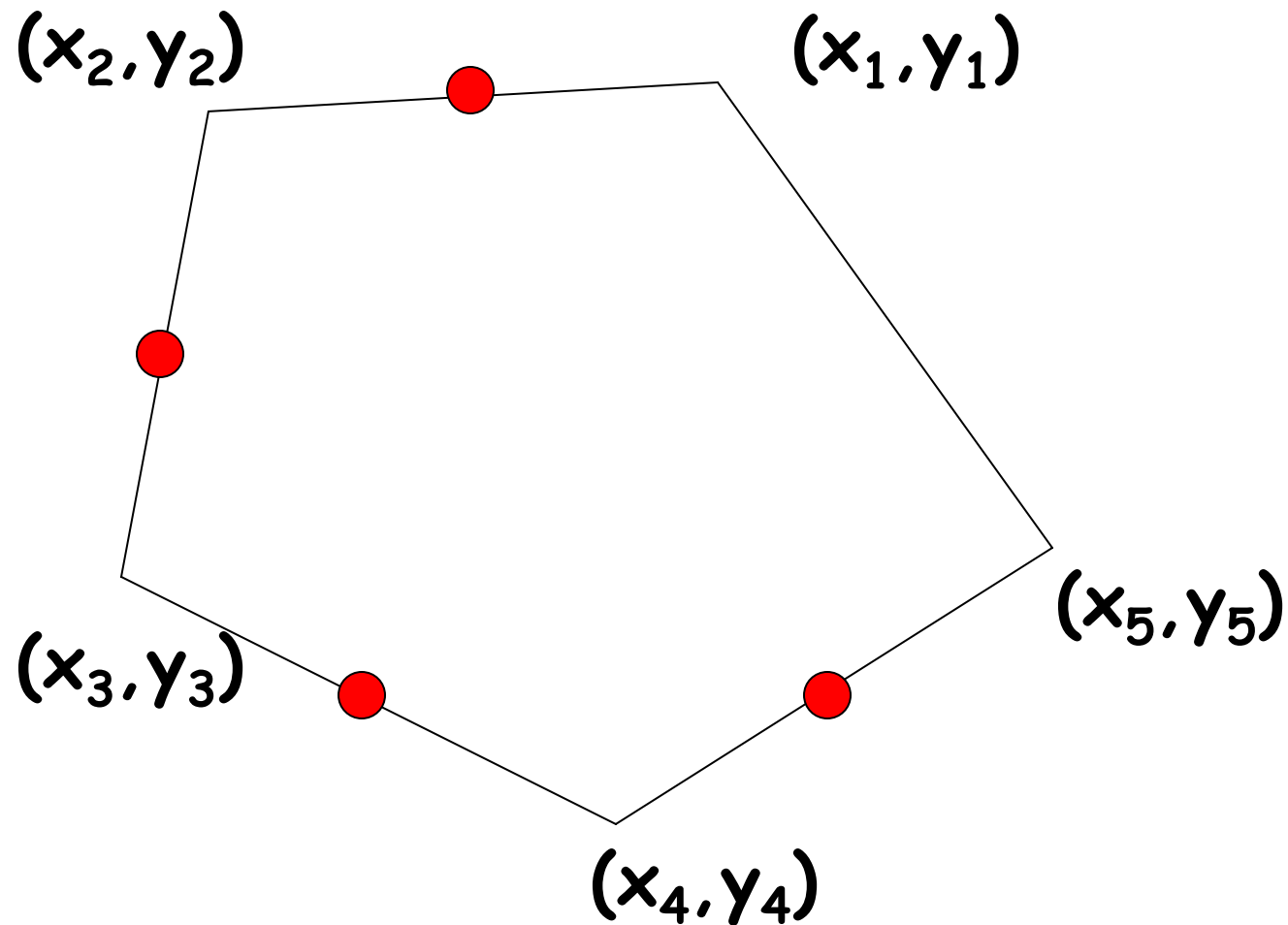
Does above fragment compute the new n-gon?

A: Yes

B: No

$$x_{\text{New}}(4) = (x(4) + x(5)) / 2$$

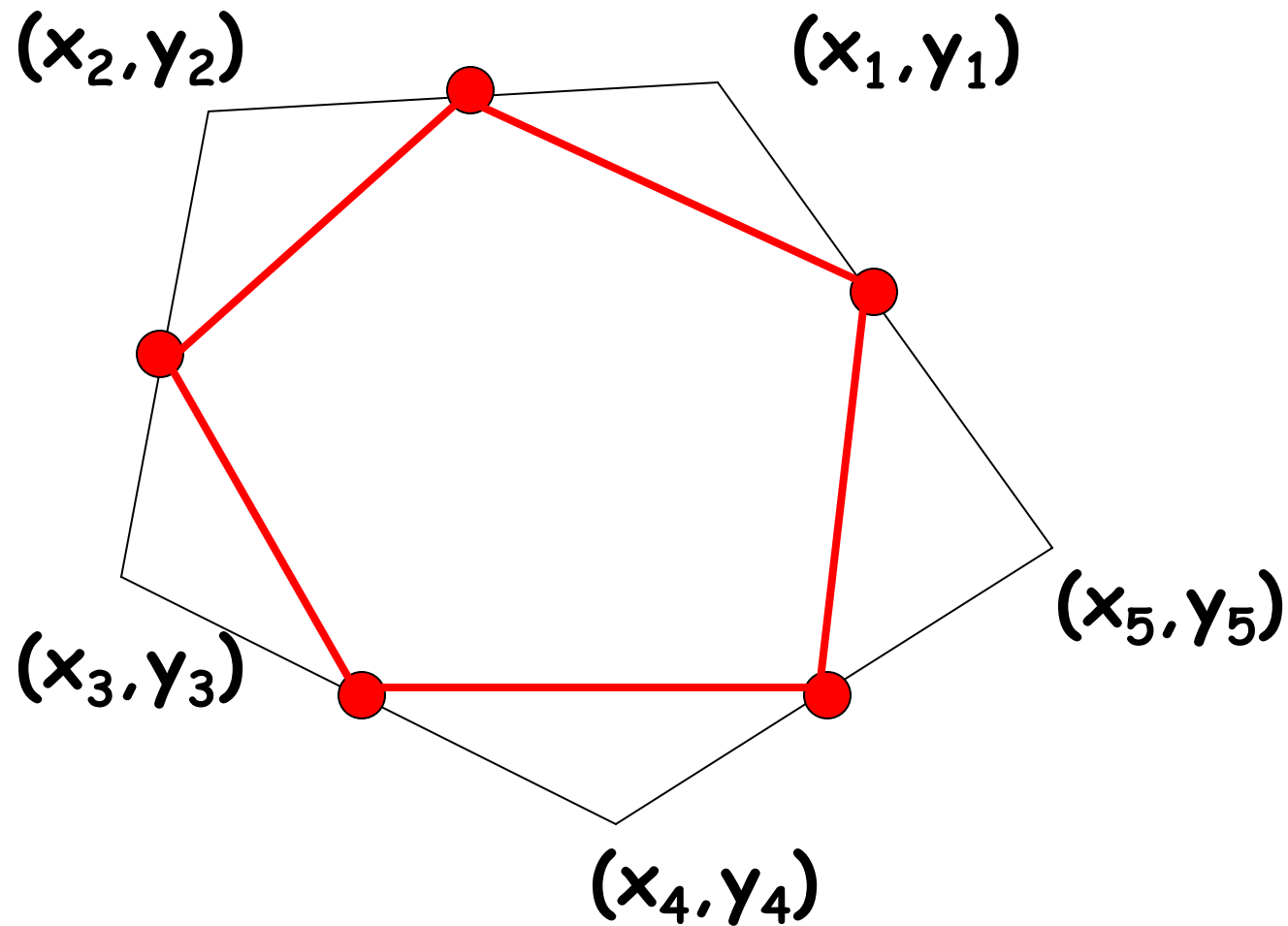
$$y_{\text{New}}(4) = (y(4) + y(5)) / 2$$





$$x_{\text{New}}(5) = (x(5) + x(1)) / 2$$

$$y_{\text{New}}(5) = (y(5) + y(1)) / 2$$



## 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

Start with 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
```

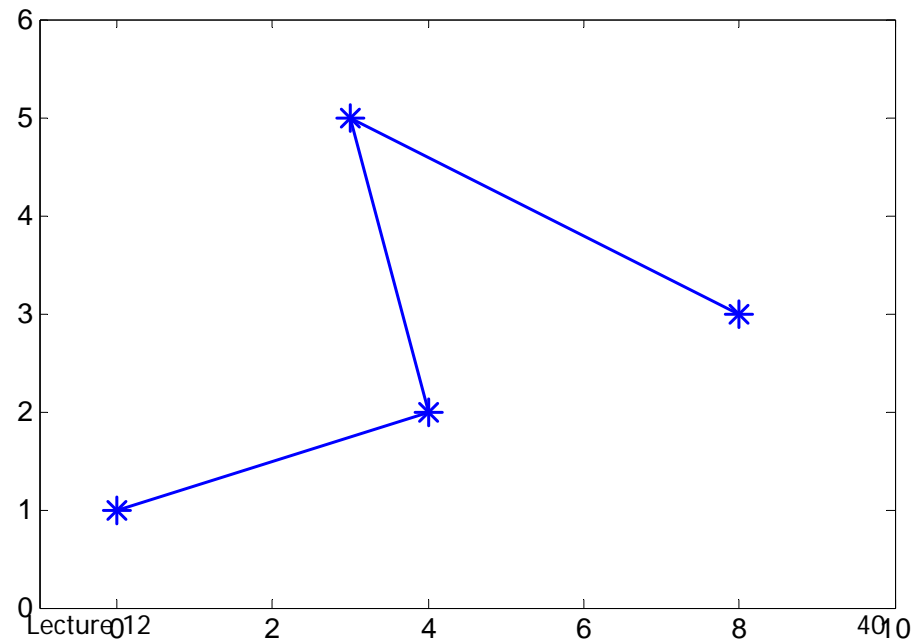
```
b= [1 2 5 3]; % y-coords
```

```
plot(a, b, '-*')
```

x-values  
(a vector)

y-values  
(a vector)

Line/marker  
format





## Drawing a polygon (multiple line segments)

```
% Draw a rectangle with the lower-left  
% corner at (a,b), width w, height h.  
x= [          ]; % x data  
y= [          ]; % y data  
plot(x, y)
```

Fill in the missing vector values!

## Drawing a polygon (multiple line segments)

```
% Draw a rectangle with the lower-left  
% corner at (a,b), width w, height h.  
x= [a    a+w    a+w    a        a ]; % x data  
y= [b    b        b+h    b+h    b ]; % y data  
plot(x, y)
```

## Coloring a polygon (fill)

```
% Draw a rectangle with the lower-left  
% corner at (a,b), width w, height h,  
% and fill it with a color named by c.  
x= [a  a+w  a+w  a  a];  % x data  
y= [b  b    b+h  b+h  b];  % y data  
fill(x, y, c)
```



A built-in function

## Coloring a polygon (fill)

```
% Draw a rectangle with the lower-left  
% corner at (a,b), width w, height h,  
% and fill it with a color named by c.
```

```
x= [           ]; % x data  
y= [           ]; % y data
```

```
fill(x, y, c)
```



A built-in function

## Coloring a polygon (fill)

```
% Draw a rectangle with the lower-left  
% corner at (a,b), width w, height h,  
% and fill it with a color named by c.  
x= [a  a+w  a+w  a  a];  % x data  
y= [b  b    b+h  b+h  b];  % y data  
fill(x, y, c)
```

Built-in function **fill** actually does the “wrap-around” automatically.