Intro Math Problem Solving
October 5

Taylor Polynomials
Colors
Compute and Display Triangle Centroid
Design the letter "A"
Design a Flag
A Little More About Lists (“Vectors”)
Polygons and Triangulation
Homework #5

Taylor Polynomials

As you will learn in calculus, a function f(x) can often be represented by an infinite Taylor series, which has the mysterious form:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(0)}{n!} x^n$$

$$= f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \frac{f'''(0)}{3!}x^3 + \cdots$$

exp(x) and sin(x)

For exp(x), the series looks like:

exp(x) = 1 + x + x^2/2! + x^3/3! + ...

and for sin(x), it is:

sin(x) = x/1! - x^3/3^! + x^5/5! - x^7/7!...

By stopping the summing process early, we get a Taylor polynomial, which can approximate the function near x=0.

Series, Sequences, Stepping Stones

We have talked about infinite series and approximating them by stopping early.

There is a "stepping stone" relation for the sequence of successive terms. For example, for exp(x), if we call the i-th term a(i), then:

a(i-1) = x^(i-1)/(i-1)!

a(i) = x^i/i!

so a stepping stone rule is:

a(0) = 1;

a(i) = a(i-1) * x / i;

Plotting Taylor Polynomials

The codes taylor_exp.m and taylor_sine.m demonstrate how a Taylor polynomial can be computed and plotted and compared to the original function.

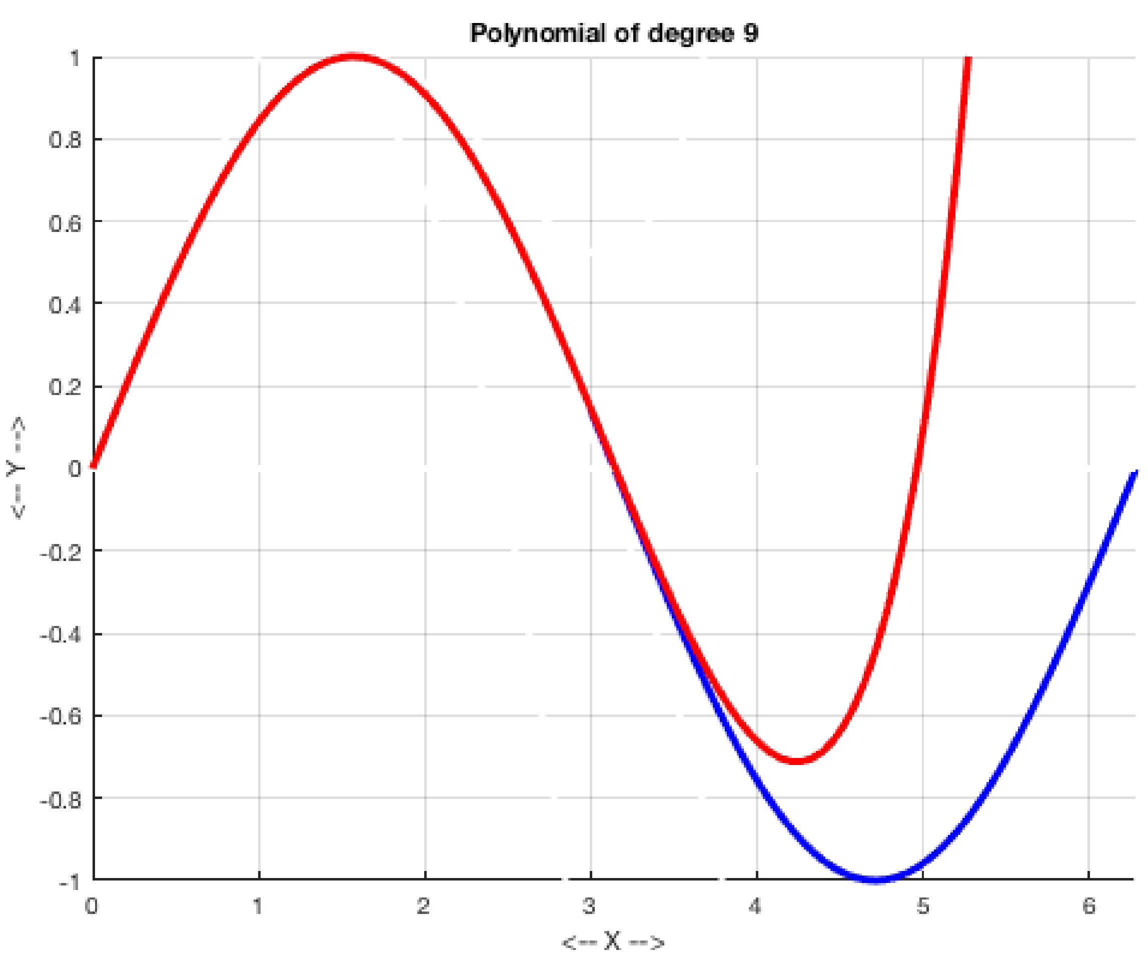
The big advantage is that polynomials are much easier to work with and to understand.

The codes contain complicated commands we haven't talked about, but they illustrate why Taylor polynomials are so useful, and inside the codes you can still spot a lot of MATLAB you can understand or guess at.

A piece of taylor_exp.m

```
for i = 0 : n

  if ( 0 < i )
    plot ( x, p, 'w-', 'linewidth', 3 );  %  White out previous polynomial
  end

  if ( i == 0 )
    term = ones ( 1, dots );  % Get a vector of 1's
  else
    term = term .* x / i;       % <-- x^i/i! = x^(i-1)/(i-1)! * ( x / i )
  end

  p = p + term;  % Add next term to Taylor polynomial

  plot ( x, p, 'r-', 'linewidth', 3 );  % Plot the updated polyomial.

end
```

9th degree Taylor Polynomial for Sin(x)



8 Easy Colors

Both the plot() and fill() commands allow us to choose a color from 8 choices:

'r', 'g', 'b': red/green/blue

'c','m', 'y': cyan/magenta/yellow

'k','w': black/white

```
plot ( xlist, ylist, 'g-' )  line in green
fill ( xlist, ylist, 'c' )   fill with cyan
```

RGB Colors

To specify more colors, we have to use MATLAB's RGB description:

```
mycolor = [ red, green, blue ];
```

where each value is between 0 and 1.

```
orange = [ 1.0, 0.4, 0.0 ];
maroon = [ 0.4, 0.0, 0.0 ];
```

```
plot ( xlist, ylist, 'LineColor', orange );
```

```
fill ( xlist, ylist, maroon );
```

Color is a another list or vector

Any color is a mix of red, green, and blue.

Represent a color with a length-3 vector and an “rgb convention”.

c = [0.25 0.63 0.00]

red value between 0 and 1

green value between 0 and 1

blue value between 0 and 1

Our 8 Easy Colors:

White	[1 1 1]	
Blue	[0 0 1]	
Green	[0 1 0]	
Cyan	[0 1 1]	
Red	[1 0 0]	
Magenta	[1 0 1]	
Yellow	[1 1 0]	
Black	[0 0 0]	

DEMO: Exploring [r,g,b] colors

The "colors_random.m" script will show you some random [r,g,b] colors.

The "colors_rgb.m" script will let you specify the [r,g,b] values and show you the corresponding color.

So colors can be specified with the 8 shortcut abbreviations, or an RGB list.

Using colors with plot()

The plot command accepts one letter colors in commands like:

```
plot ( xlist, ylist, 'r-' )
plot ( xlist, ylist, 'bo-' )
```

To use an RGB color:

```
color = [ 1.0, 0.4, 0.0 ];
plot ( xlist, ylist, 'LineColor', color );
```

or

```
plot ( xlist, ylist, 'LineColor', [ 0.1, 0.5, 0.3] );
```

The fill(xlist,ylist,color) command will accept one-letter colors ('r', 'c', 'k') OR color vectors such as color = [1.0, 0.4, 0.0].

Using colors with fill()

The fill command accepts one letter colors and RGB colors in the same way:

```
fill ( xlist, ylist, 'r' )
```

or

```
color = [ 0.9, 0.5, 0.5 ];
fill ( xlist, ylist, color );
```

or

```
fill ( xlist, ylist, [ 0.12, 0.37, 0.83] );
```

The Centroid Problem

Suppose we have a triangle T with corners A, B, and C.

The centroid of T is the "balance point" of the triangle. Any line through the centroid will split it into two equal areas.

The centroid can be computed as the average of the points A, B, and C.

Our task is to:

Draw a triangle T.

Compute the centroid and display it.

Draw dashed lines from each corner to the centroid.

User Picks Points with ginput()

The function ginput() lets us click on selected points on the screen.

We can use this to choose an arbitrary triangle to work on.

[xlist,ylist] = ginput(); ← Pick til RETURN;

[xlist,ylist] = ginput(3); ← Pick list of 3.

[x, y] = ginput(1); ← Pick one point;

Get triangle corners A, B, C

```
[ ax, ay ] = ginput(1);
[ bx, by ] = ginput(1);
[ cx, cy ] = ginput(1);

xlist = [ ax, bx, cx ];  ← collect points into a list
ylist = [ ay, by, cy ];

fill ( xlist, ylist, 'y' );                ← Fill triangle with yellow;
hold on;                                    ← More graphics coming!
plot ( [ xlist, ax ], [ylist,ay], 'b-' ); ← Blue outline.
```

Compute Centroid

Centroid is average of vertices:

```
dx = ( ax + bx + cx ) / 3.0;
dy = ( ay + by + cy ) / 3.0;
```

OR:

```
dx = ( xlist(1) + xlist(2) + xlist(3) ) / 3.0;
dy = ( ylist(1) + ylist(2) + ylist(3) ) / 3.0;
```

Since xlist = [ax, bx, cx], we can find any of the three values simply by specifying its location within the list.

The location in parentheses (1), (2), or (3), is called a **subscript** or **index**. We will come back to this topic soon!

Mark the Centroid

One version of the plot command doesn't draw lines, but instead puts markers at the given locations. To mark with blue asterisks:

```
plot ( xlist, ylist, 'b*' );
```

Choices include 'r.' or 'go' or 'bs' or 'cp'.

```
plot ( dx, dy, 'k.', 'MarkerSize', 50 );
```

'MarkerSize' makes the symbols bigger.

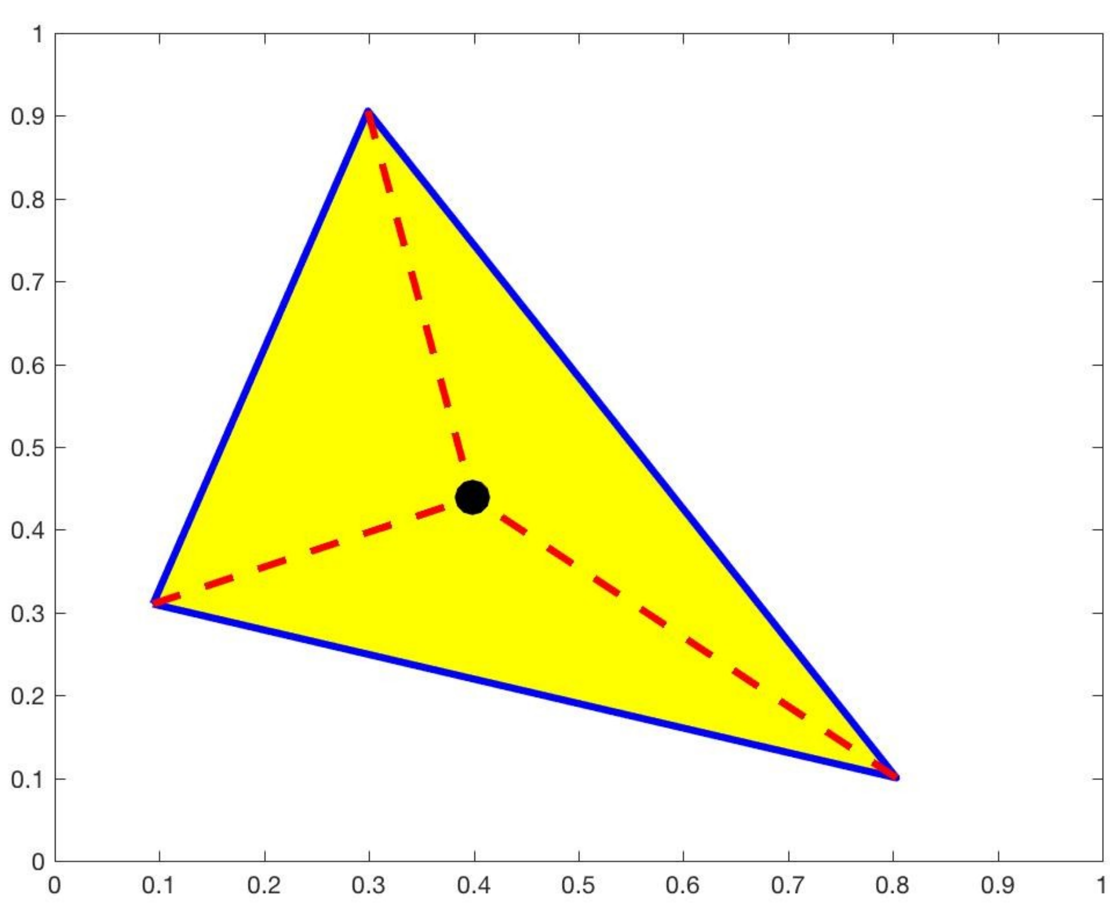
Connect Corners to Centroid

Just to make the plot fancy, we connect each corner to the centroid with a dashed line. We could issue three plot() commands to do this, but instead we use a **for** loop:

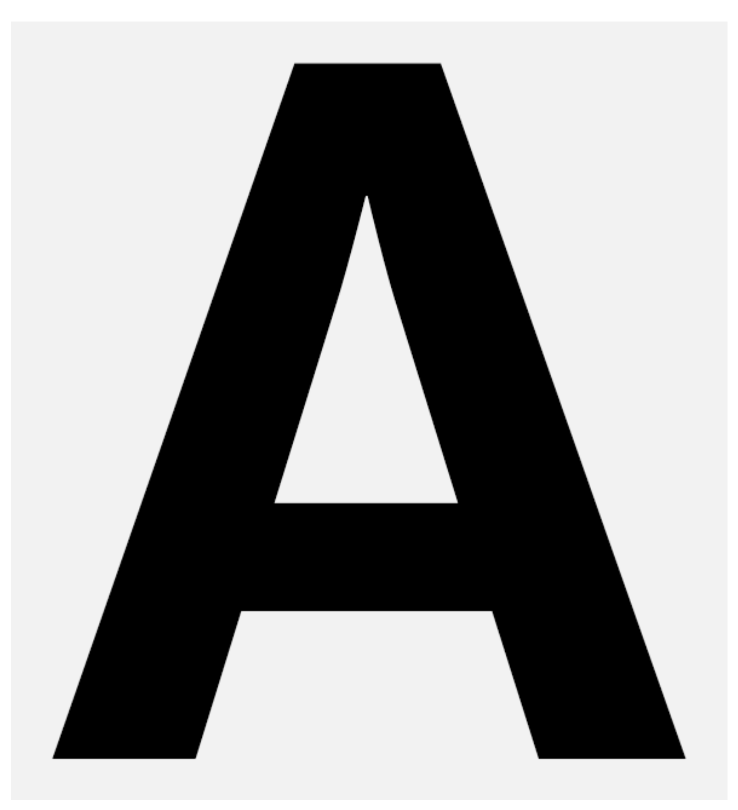
```
for i = 1 : 3
  plot ( [ xlist(i), dx ], [ ylist(i), dy ], 'r--', 'LineWidth', 3 );
end
```

By using subscripted values in the list, we are able to write the command **one time**, but have the for loop execute three versions of it, which takes care of all the drawing.

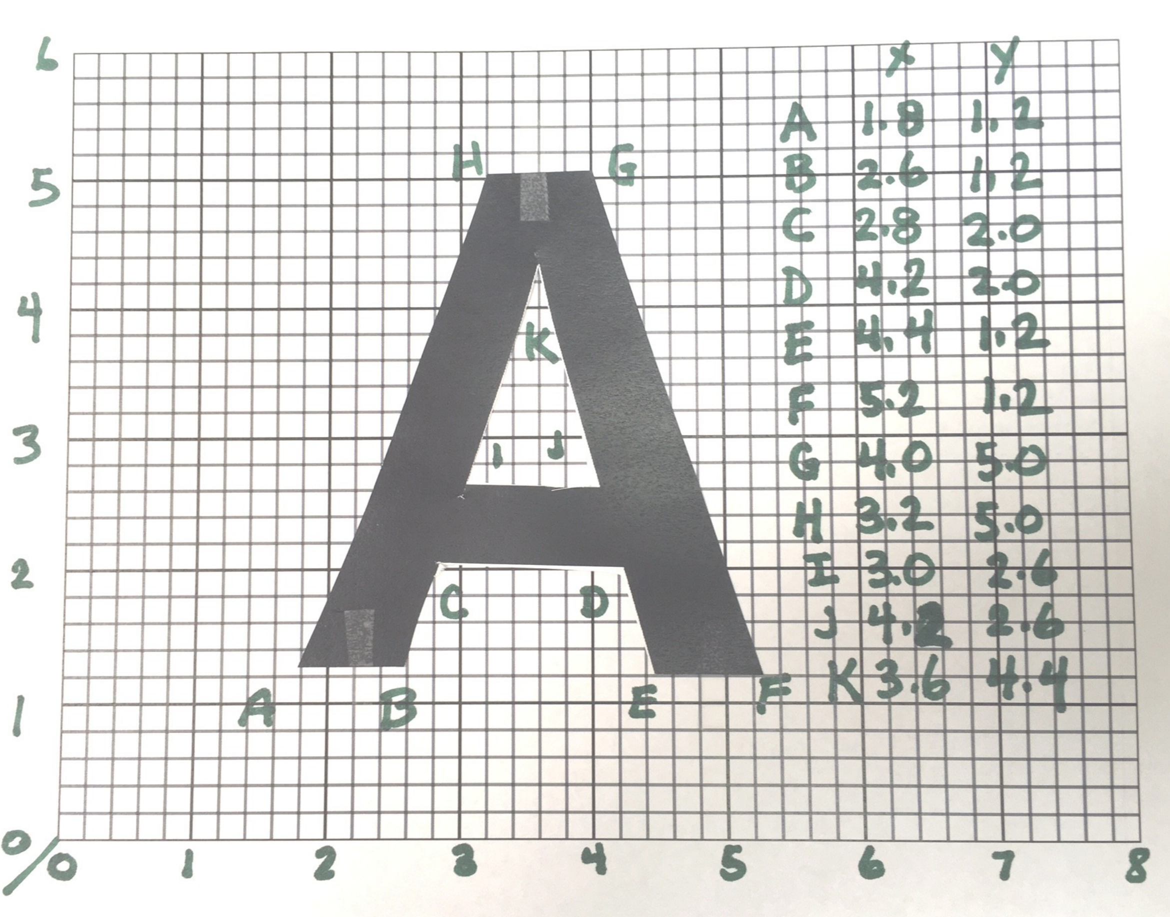
A Sample Centroid Plot



Suppose we want to plot a big "A"



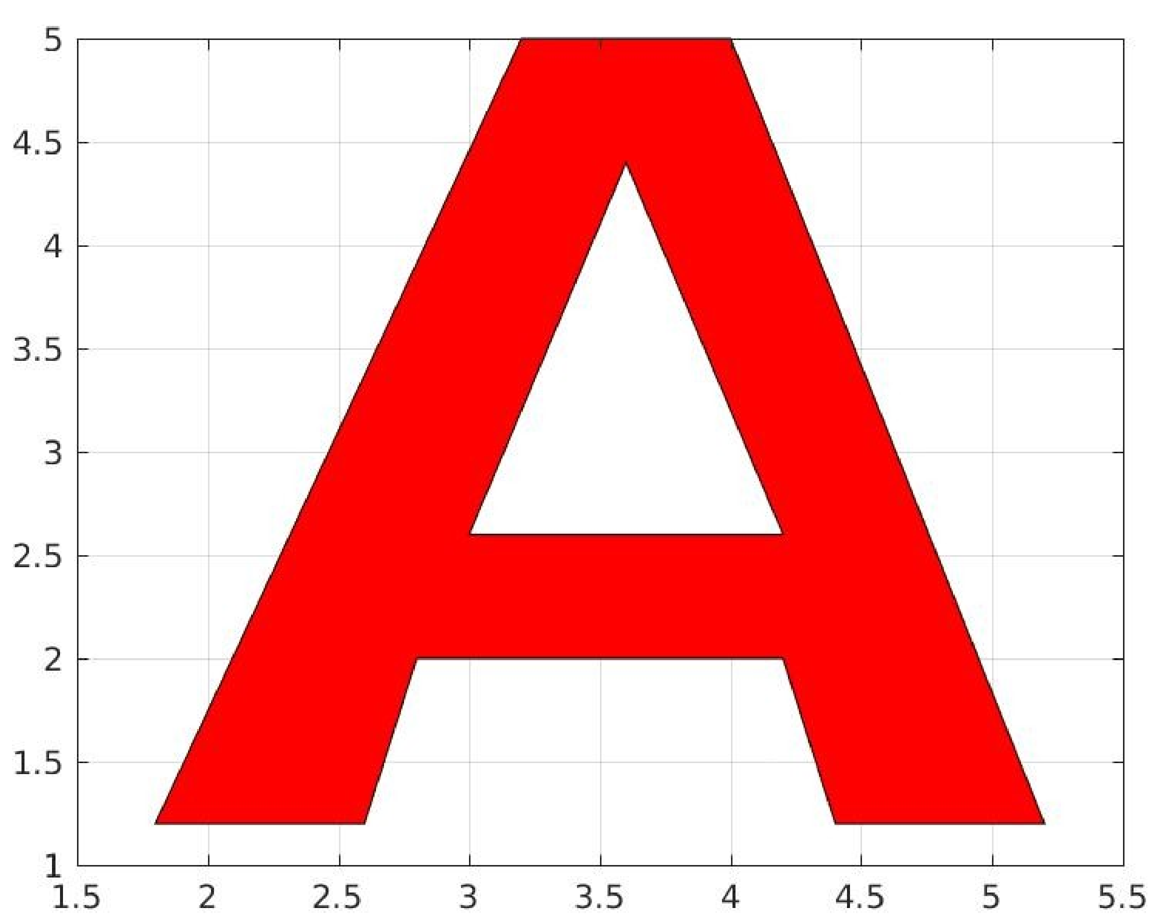
How to Make an "A" in this Class



big_a.m

```
% big_a.m
% Draw a big letter A.
%
xlist1 = [ 1.8, 2.6, 2.8, 4.2, 4.4, 5.2, 4.0, 3.2 ];
ylist1 = [ 1.2, 1.2, 2.0, 2.0, 1.2, 1.2, 5.0, 5.0 ];

xlist2 = [ 3.0, 4.2, 3.6 ];
ylist2 = [ 2.6, 2.6, 4.4 ];

fill ( xlist1, ylist1, 'r' );
hold on
fill ( xlist2, ylist2, 'w' );
%
grid on
print ( '-djpeg', 'big_a.jpg' );
```

big_a.jpg



Design a Flag

For the new country of Vatechia, we want to plot a flag. It should be 6 units wide and 4 units tall. This makes 24 little squares, and we want a yellow star in every other square, and a blue background:

```
*   *   *
  *   *   *
*   *   *
 *   *   *
```

Fill the Background First

To make the blue background, we need to set the vertices of a rectangle of width 6 and height 4. (By the way, we need to list the vertices in some order, not just scrambled!)

```
fill ( [ 0, 6, 6, 0 ], [ 0, 0, 4, 4 ], 'b' );
```

We do the background first. That way, the stars appear in front of the background.

The "star()" function

I wrote a function to draw stars:

```
star ( xc, yc, r, c );
```

To use it, specify:

- xc, yc: coordinates of the star center;
- r: the radius (size) of the star;
- c: the color ('r', 'g', 'b', 'c', ...);

A function makes it easy to draw lots of stars in a systematic way.

We will spend a whole class, later, on how to create and use our own functions.

The "guts" of star.m

```
function star ( xc, yc, r, color )

  a = pi / 2 + linspace ( 0, 2 * pi, 11 );  ← Get 11 (actually 10) equal angles.

  ca = cos ( a );
  sa = sin ( a );

  xlist = r * ca;                            ← Locate 10 points on circle of radius r.
  ylist = r * sa;

  xlist(2:2:10) = xlist(2:2:10) / (2*(1+sin(pi/10)));  ← Shrink 5 of the points.
  ylist(2:2:10) = ylist(2:2:10) / (2*(1+sin(pi/10)));

  xlist = xc + xlist;                        ← move points to center (xc,yc)
  ylist = yc + ylist;

  fill ( xlist, ylist, color );              ← Draw the shape.
```

Where do we want stars?

A pair of for loops move row I (1 to 4) and column J (1 to 6).

In row I=1, we want stars in columns J=2, 4, 6;
In row I=2, we want stars in columns J=1, 3, 5;
In row I=3, we want stars in columns J=2, 4, 6;
In row I=4, we want stars in columns J=1, 3, 5;

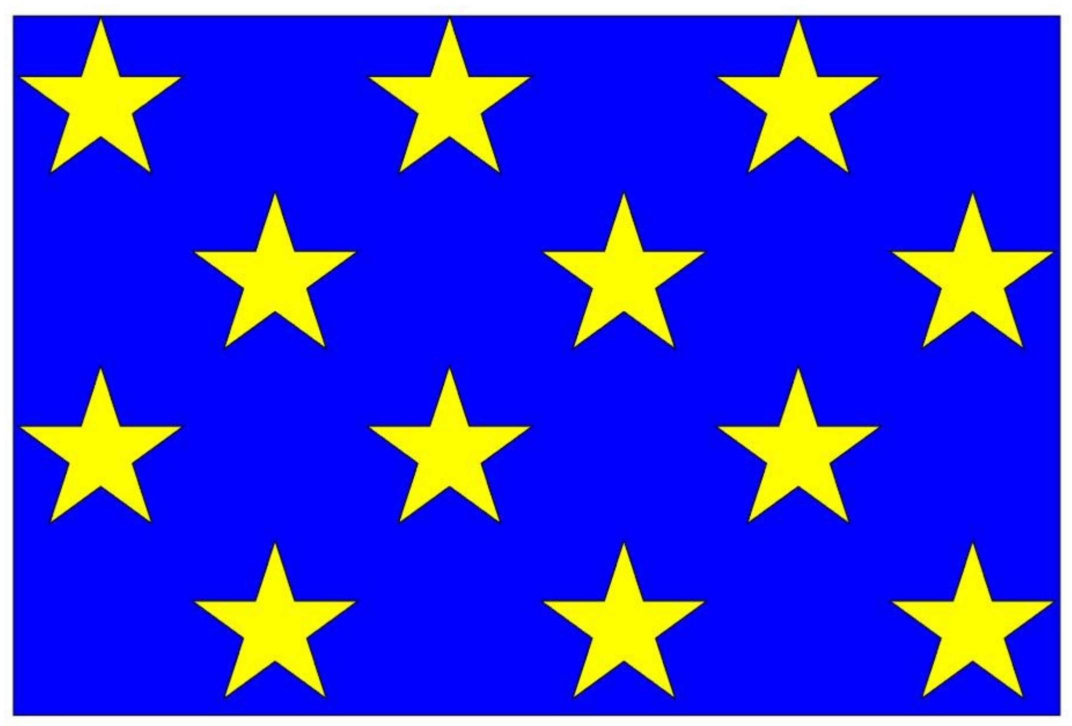
When I is odd, we want the even J's.
When I is even, we want the odd J's.

So when I + J is odd, we want to draw a star.

A Field of Stars

```
hold on
for i = 1 : 4                  ← Make 4 rows
  for j = 1 : 6                ← Each row has 6 columns
    if ( mod ( i + j, 2 ) == 1 )  ← Only draw some
      star ( j - 1/2, i - 1/2, 0.5, 'y' );
    end
  end
end
hold off
```

Flag of Vatechia



Ways to Create a List

We know several ways to create a list:

```
xlist = []; xlist = [ xlist, new_value ];
xlist = [ 1, 2, 3, 4 ];
xlist = linspace ( 15, 20, 51 );
```

New ways to make a list:

- xlist = zeros (1, 10); ← list of 10 0's.
- xlist = ones (1, 10); ← list of 10 1's.
- xlist = rand (1, 10); ← list of 10 random values between 0 and 1.
- xlist = randn (1, 10); ← list of 10 "normal" random values, with average 0.

The (1,10) means "1 row, 10 columns".

If we used (10,1) we'd get a column vector, and if we used (5,2) we'd get a 5x2 matrix, neither of which we want right now!

(But try xlist = ones(5,2) and see what happens!)

Functions of Lists

We have seen that we can apply functions to a list and get a list of results. Some useful functions include:

big = max (xlist);

small = min (xlist);

ave = mean (list); ← average them

total = sum (xlist); ← add 'em up

n = length (xlist); ← how long is this list?

ylist = sort (xlist); ← sort the list.

Example: 100 random numbers
list_std.m

```
x = rand ( 1, 100 );
x(1:5);  ← print first five values;
y = sort ( x );
y(1:5);  ← print first five values.
n = length ( y );
yave = sum ( y ) / n;
yave = mean ( y );  ← same as previous line
ystd = sqrt ( sum ( ( y - yave ).^2 ) / n );
ystd = sqrt ( mean ( ( y - yave ).^2 ) );
```

Picking a List Entry

If xlist contains 5 values, then they each have an "address" or "index" or "subscript" based on their position.

Using that index, you can copy, modify or completely replace any single value.

This also means that a **for** loop can be used to define the entries of a list.

Indexing a List

Set xlist to [101, 202, 303, 404, 505];

xlist(2) ← will print "202"

xlist(4) = xlist(4) + 12

xlist is now [101, 202, 303, 416, 505];

xlist(1) = xlist(2) + xlist(5);

xlist is now [707, 202, 303, 416, 505];

max (xlist) ← will print "707"

Define a List with "for"

Compute and store the first 100 Fibonacci numbers

```
n = 100;
f = zeros ( 1, n );  ← set up space for the list.

for i = 1 : n
  if ( i == 1 )
    f(i) = 1;
  elseif ( i == 2 )
    f(i) = 1;
  else
    f(i) = f(i-1) + f(i-2);
  end
end
```

Define a list with "for"

Evaluate the Legendre polynomials at x = 0.4;

```
for i = 1 : n
  if ( i == 1 )
    p(i) = 1.0;
  elseif ( i == 2 )
    p(i) = x;
  else
    p(i) = ( ( 2*i-1) * x * p(i-1) - ( i-1 ) * p(i-2) ) / i;
  end

end
```

Using Colon Notation

Remember in for loops how we could specify a range of values as i=ilo:ihi, or even i=ilo:iskip:ihi?

If xlist = [11, 22, 33, 44, 55, 66, 77, 88, 99]

then xlist(2:4) is the values 22, 33, 44.

xlist(1:2:9) is the values 11,33,55,77,99.

xlist(3:5) = xlist(3:5) + 2

will result in

xlist = [11, 22, 35, 46, 57, 66, 77, 88, 99]

Polygons

A polygon is any shape with straight line boundaries.

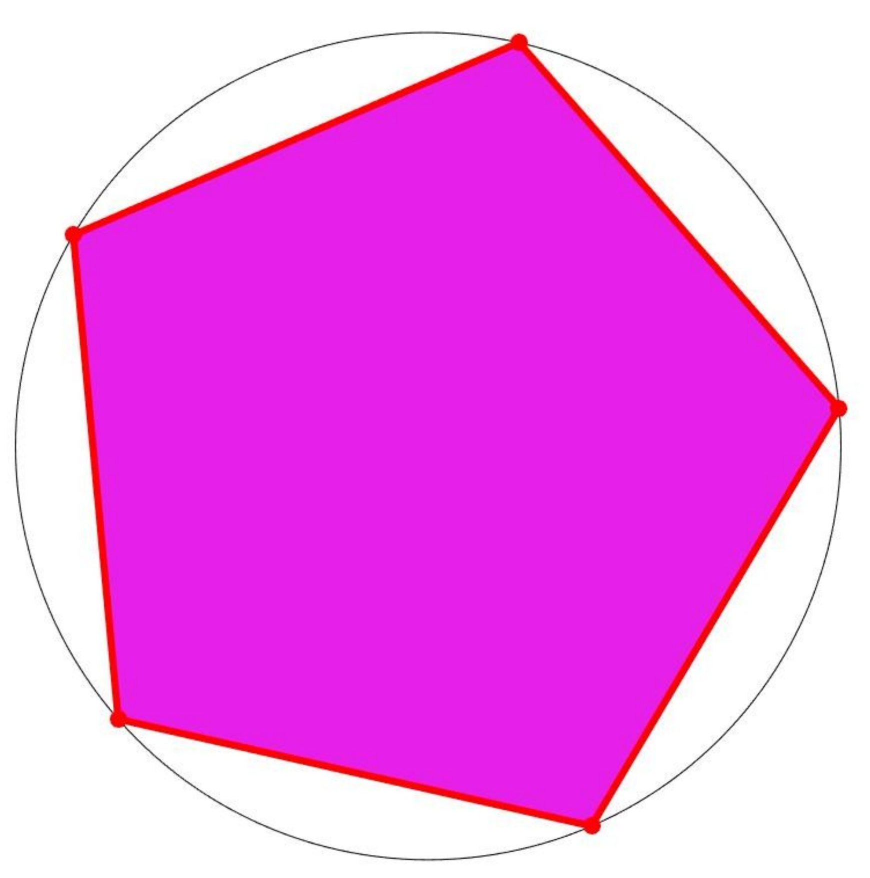
We'll assume the boundary never crosses itself.

A regular polygon has equal sides and equal angles. We can generate an N-sided regular polygon by connecting N points separated by equal angles.

For convenience, “x” and “y” might store the N coordinates of the points (or “corners” or “vertices”).

Demo: regular_polygon.m

A Regular Pentagon



Triangle: the Simplest Polygon

If we know the vertices of a triangle, we can compute its area:

```
area = 0.5 * ( ...
    x(1) * ( y(2) - y(3) ) ...
  + x(2) * ( y(3) - y(1) ) ...
  + x(3) * ( y(1) - y(2) ) );
```

And we know the centroid of the triangle is found by averaging x and y coordinates.

Triangle Centroid Using Lists

We know the centroid of the triangle is found by averaging x and y coordinates.

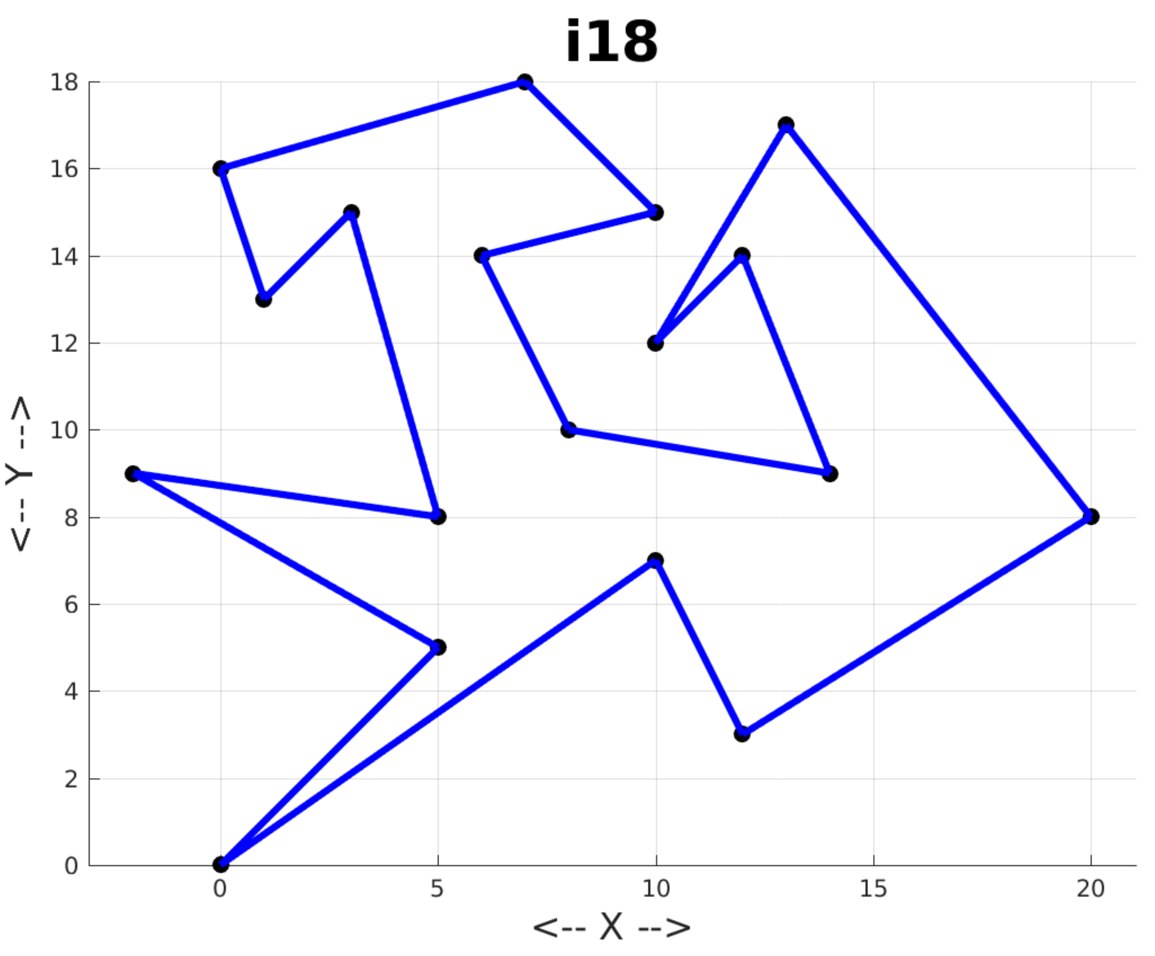
If we stored these coordinates in lists, the centroid computation simplifies:

```
xc = sum ( x ) / 3;
yc = sum ( y ) / 3;
```

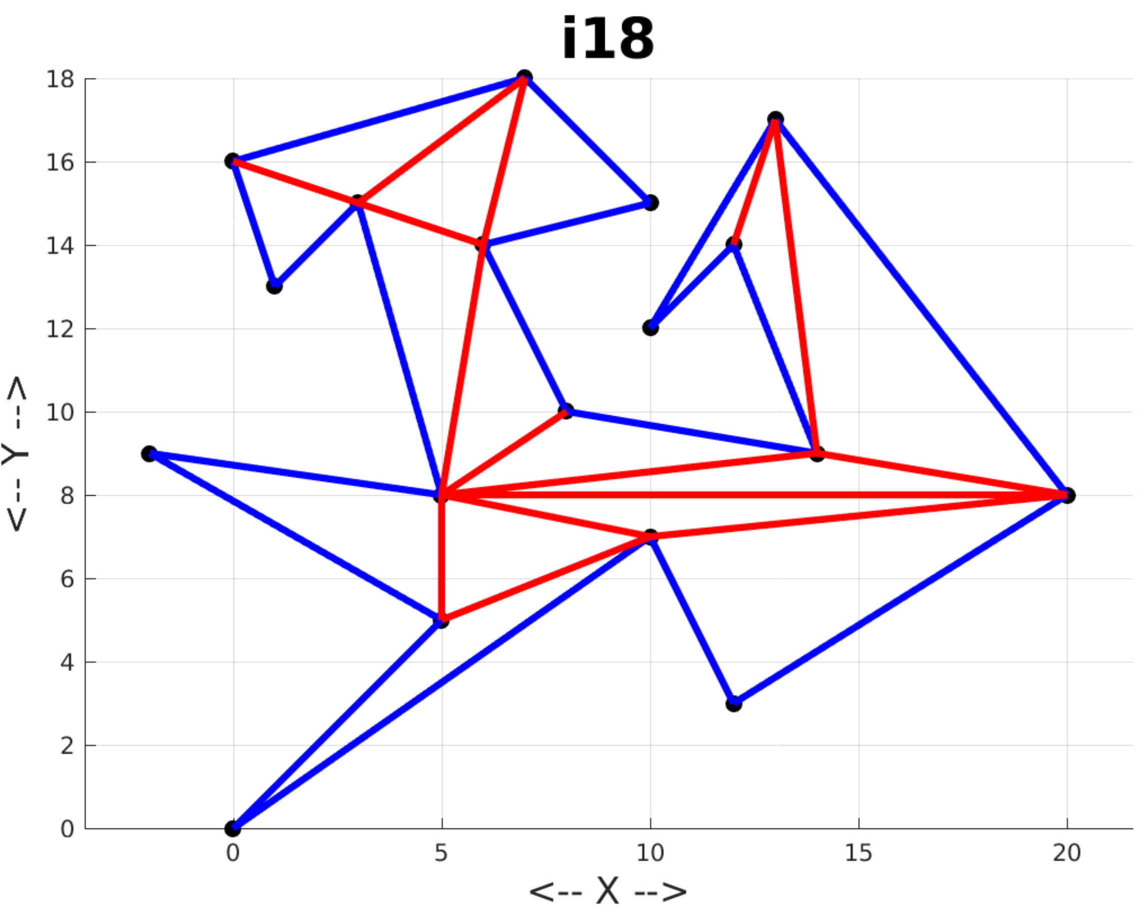
instead of

```
dx = ( ax + bx + cx ) / 3;
dy = ( ay + by + cy ) / 3;
```

Given any polygon...



...We can "triangulate" it



Analysis from Triangles

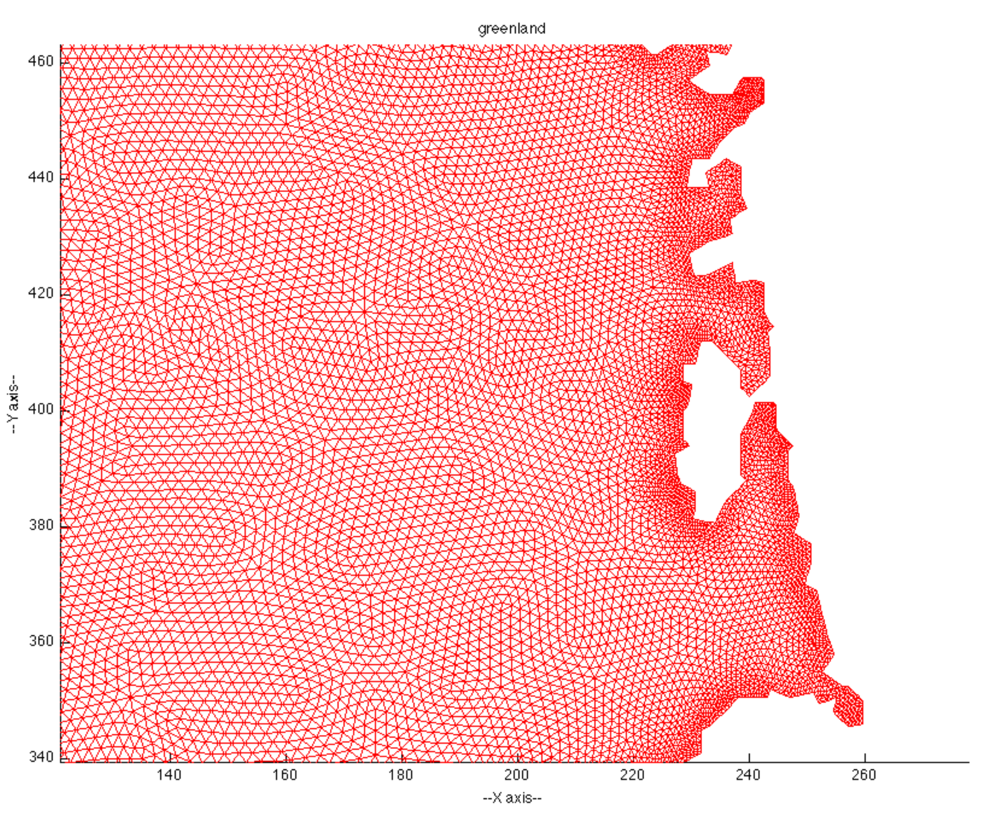
Once a polygon is triangulated, we can compute its area (sum the areas of the triangles);

The centroid of the polygon is found by multiplying each triangle centroid by its area, and then dividing by the total area.

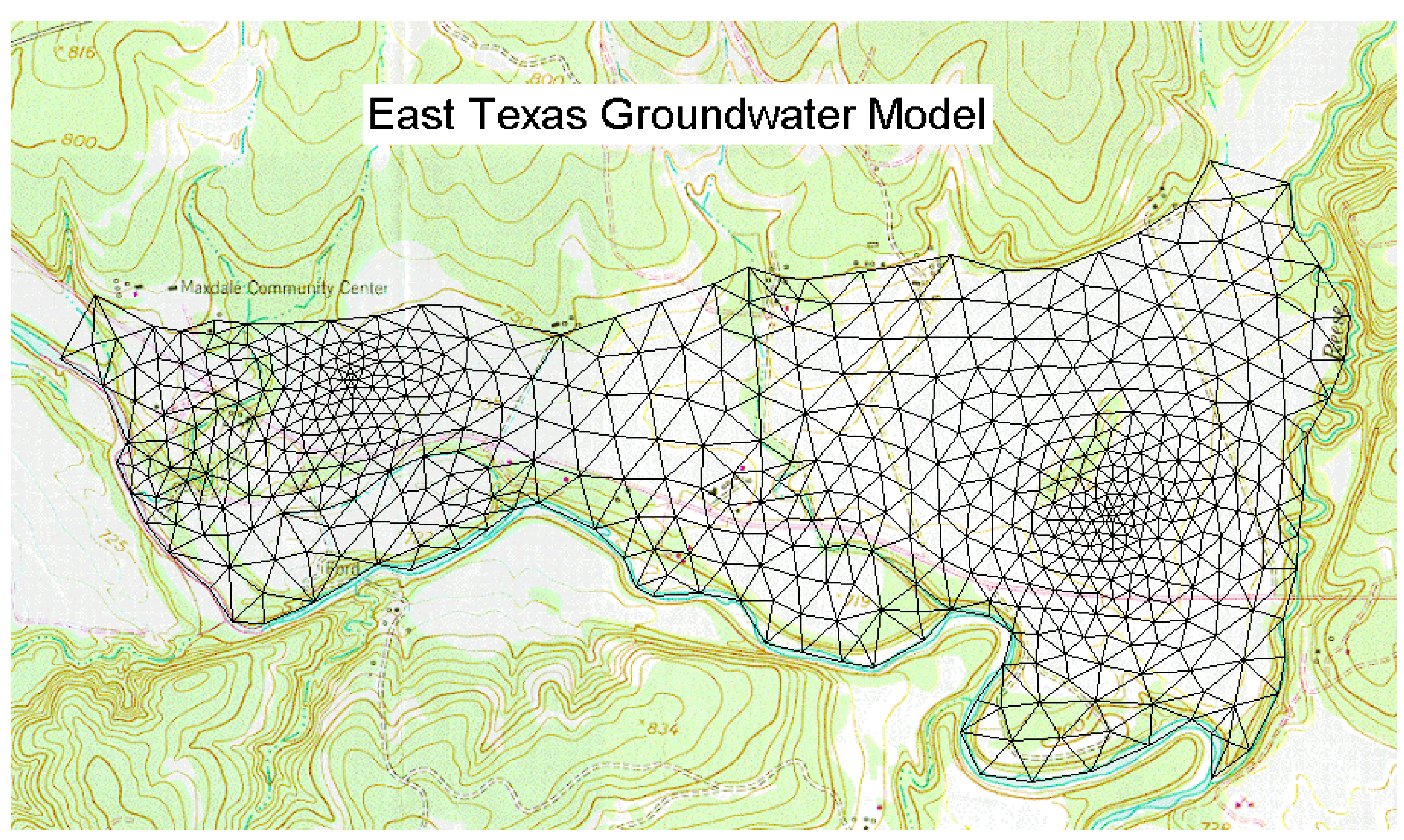
Triangulating a shape creates a mathematical model that can be analyzed.

Advanced versions of triangulation are used in mapping, and in modeling 3D surfaces.

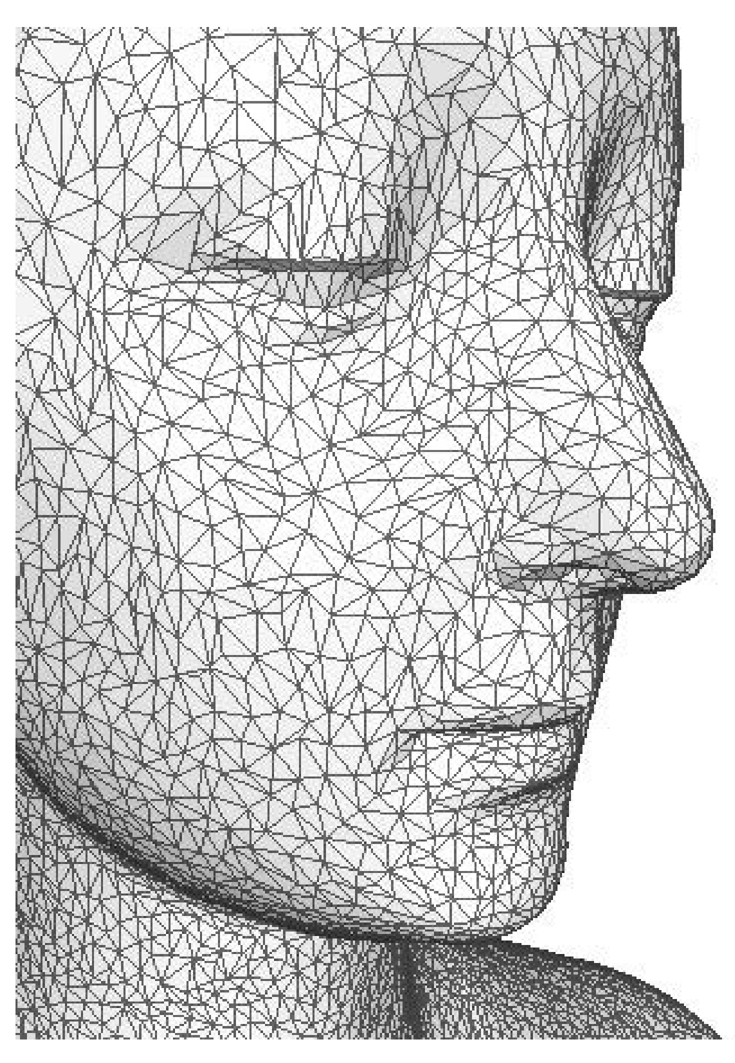
A Tiny Portion of Triangulated Greenland



A Triangulation for Groundwater



A Triangulated Face



Question Time

What is the output?

```
x = [10 20 30];
y = [3 1 2]
k = y(3)-1;
z = x(k+1)
```

A. 11 B. 20 C. 21 D. 30 E. 31

Homework #5

(Homework #4 due this Friday, midnight)

hw034: plot several Chebyshev polynomials of the second kind on one graph;

hw035: plot the letters "V" and "T" as filled polygons;

hw037: using a function I give you called "ellipse_fill()", draw a face using ellipses of different positions, shapes, and colors.