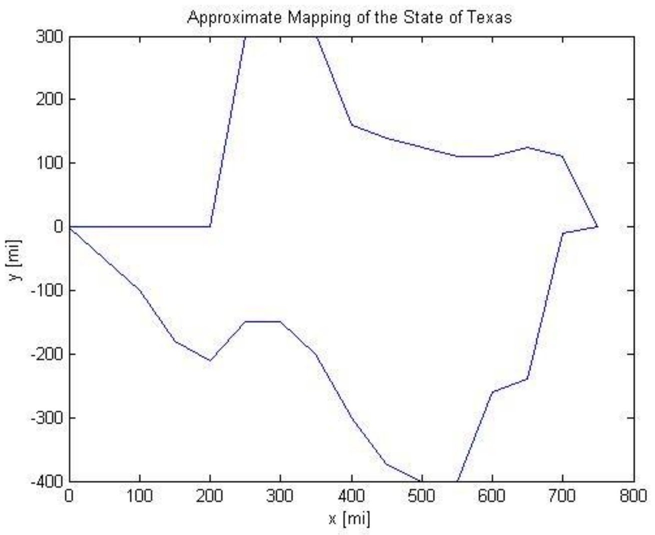
```
% Will Kram-linger
% CE 3101
% Gonella
% 4/10/14
% HW 6

% PROBLEM 1

% Please see stapled sheet.
% Commentary on Problem 1: Implementation of Numerical methods is very
%                          tedious by hand.

% PROBLEM 2

x = (0:50:750);
y_up = [0 0 0 0 0 300 300 300 160 140 125 110 110 125 110 0];
y_down = -[0 50 100 180 210 150 150 200 300 375 400 400 260 240 10 0];

hold on; plot(x,y_down);
plot(x,y_up);
```



```
% The coordinate-based approximation of Texas seems reasonable.
% If I was working for the state of Texas, I would've been a bit more
% careful with my plotting.  I also have met some unsavory characters from
% Texas.

edit
```

```
function I = comp_trap(x,y)
% Will Kramlinger; 4/6/14
% The function takes x and y arrays, representing a single variable
% function, and calculates an integral using the composite trapezoidal
% method.
% Input Variables:
% x = an array representing the independent variable
% y = an array representing the dependent variable
% Output Variables:
```

```
% I = the resulting integral (scalar value)

if length(x) ~= length(y)
    error('x and y must the same length, ya goon!')
else
    n = length(x) - 1; % n = number of intervals
end

table = zeros(n,1);
for k = 1:n
    table(k) = .5 * (y(k) + y(k+1))*(x(k+1) - x(k));
end
I = sum(table);
end
```

```
I = comp_trap(x,y_up) % Calculates the top integral

I =

       89000

I = comp_trap(x,y_down) % Calculates the bottom integral

I =

     -151250

% For finding area, we take the absolute value of the latest answer.
% The total area found by the composite trapezoidal method is 89000 +
% 151250 = 240250 sq. mi

x(17) = 800; y_down(17) = 0; y_up(17) = 0;

% For Simpson's 1/3 Method we need an even number of subintervals, so I
% added one data point such that x is equally spaced, and both y's = 0
% as to not affect the final answer.

edit
```

```
function I = simpsons_third(x,y)
% Will Kramlinger; 4/6/14
% The function takes x and y arrays, representing a single variable
% function, and calculates an integral using the composite Simpson's 1/3
% Method.
% Input Variables:
% x = an array representing the independent variable
% y = an array representing the dependent variable
% Output Variables:
% I = the resulting integral (scalar value)

if length(x) ~= length(y)
    error('x and y must the same length, ya goon!')
else
    n = length(x) - 1; % n = number of intervals
end
```

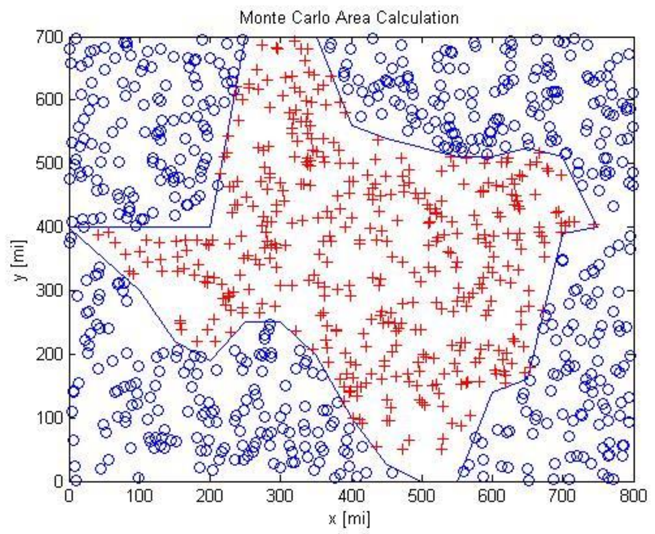
```
if mod(n,2) ~= 0
    error('You do not have an even number of equally spaced subintervals,
         ya idiot!')
else
    h = (x(end) - x(1)) / n; % h = step size between consecutive x values
end
% Beginning of integral calculations
sum1 = zeros(length(x), 1);
for k = 2:2:n
   sum1(k) = y(k);
end
sum1 = 4*sum(sum1);

sum2 = zeros(length(x),1);
for l = 3:2:(n-1)
    sum2(l) = y(l);
end
sum2 = 2*sum(sum2);

I = h/3 * (y(1) + sum1 + sum2 + y(end));
end
```

```
I = simpsons_third(x,y_up)

I =

   9.1833e+04

I = simpsons_third(x,y_down)

I =

     -154000

% The approximate area of Texas found by the Composite Simpson's 1/3
% Method is 91833 + 154000 = 245833 sq. mi.

% Via a quick Google search, the actual area of Texas is 268820 sq. mi.
% Interestingly, the Simpsons 1/3 Method provided a closer answer, even
% though the extra points I needed to add to the end of the x,y arrays
% should not have affected the integrals whatsoever.

% The relative inaccuracy of these calculations can most likely be
% attributed to the very rough approximation made by using large intervals
% along the x-axis.  It can be visually discerned from the above figure
% that I have cut out part of the actual area because of this.

% PROBLEM 3

% Through some matrix manipulation, I created one polygon out of the
% tabulated data by making arrays which were twice as long
% After perusal of the MathWorks help files, I came up with this:
```

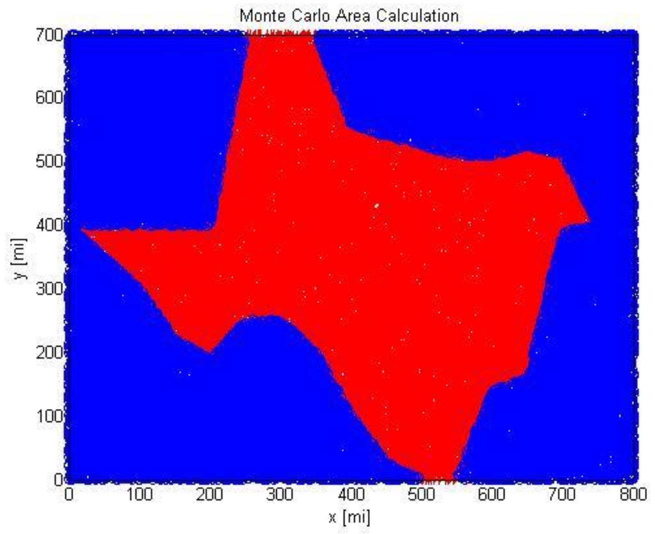
```
edit
```

```
function [r,area] = monte_carlo_area(xpoly,ypoly,xscale,yscale,n)
% Will Kramlinger; 4/6/14
% The function uses the Monte Carlo area approximation method on a given
% user-supplied polygon.
% Input Variables:
% xpoly = array of x values for the polygon
% ypoly = array of y values for the polygon
% xscale = scaling factor for the x-axis
% yscale = scaling factor for the y-axis
% n = the desired number of random points used to check in/outside the
%     area
% Output Variables:
% r = (number of points in polygon) / (total number of points)
% area = area of polygon [r * size of rectangle]

x = xscale .* rand(n,1);
y = yscale .* rand(n,1);
in = inpolygon(x,y,xpoly,ypoly);
plot(xpoly,ypoly,x(in),y(in),'r+',x(~in),y(~in),'bo');
xlabel('x [mi]'); ylabel('y [mi]');
title('Monte Carlo Area Calculation');

r = sum(in) / n;
area = r * xscale * yscale;

end
```

```
% Being dumb/lazy, I did not scale the polygon down to fit inside a 1 x 1
% box. As what I feel to be a reasonable compromise, I used the inputs
% xscale and yscale to provide a multiplier such that the range of random
% test points would match the range of the axes.

% I added 400 to the ymonte array to put the polygon in the 1st quadrant,
% facilitating placement of the random data points.

[r, area] = monte_carlo_area(xmonte,ymonte, 800, 700,1000)

r =

    0.4320


area =

      241920
```



```
[r, area] = monte_carlo_area(xmonte,ymonte, 800, 700, 50000)

r =

    0.4279


area =

    2.3963e+05
```



```
% After multiple executions of the function, the Monte Carlo method
% provided an area that centers around ~240000 sq. mi.


A = polyarea(xmonte,ymonte)

A =
      240250

% The polyarea function gives the same area as that found using the
% Composite Trapezoidal Method used in Problem 2, an interesting
% coincidence.
```