CS1115 Fall 2013 Project 3 Due Thursday October 10 at 11pm

You must work either on your own or with one partner. You may discuss background issues and general solution strategies with others, but the project you submit must be the work of just you (and your partner). If you work with a partner, you and your partner must first register as a group in CMS and then submit your work as a group. Each problem is worth 5 points. One point may be deducted for poor style. No partners are allowed for the Challenge problem.

Objectives

Completing this project will help you learn about one-dimensional arrays and how to implement and use functions. More on MATLAB graphics GUI design.

1 Intersecting Line Segments

Let L_1 be the line segment that connects the points (a_1, b_1) and (a_2, b_2). Let L_2 be the line segment that connects the points (a_3, b_3) and (a_4, b_4). Assume that the four points are distinct and that L_1 and L_2 are not parallel. If

$$\lambda \;=\; \frac{(a_3 - a_1)(b_3 - b_4) \;-\; (b_3 - b_1)(a_3 - a_4)}{(a_2 - a_1)(b_3 - b_4) \;-\; (b_2 - b_1)(a_3 - a_4)}$$

and

$$\mu \;=\; \frac{(a_2 - a_1)(b_3 - b_1) - (b_2 - b_1)(a_3 - a_1)}{(a_2 - a_1)(b_3 - b_4) \;-\; (b_2 - b_1)(a_3 - a_4)}$$

satisfy $0 \le \lambda \le 1$ and $0 \le \mu \le 1$, then L_1 and L_2 intersect at at the point

$$P \;=\; (a_1 + \lambda(a_2 - a_1), b_1 + \lambda(b_2 - b_1)).$$

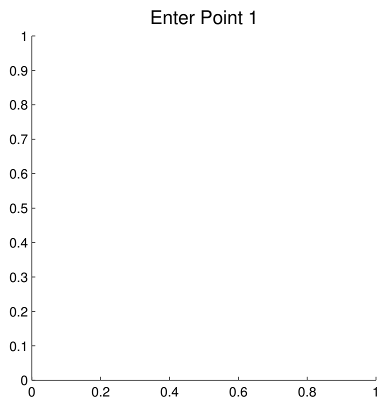
Otherwise, L_1 and L_2 fail to intersect. Complete the following function so that it performs as specified.

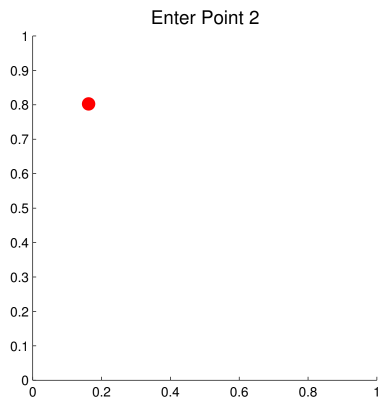
```
  function z = Intersect(a,b)
% a and b are length-4 vectors that collectively name four points.
% If the line segment that connects (a(1),b(1)) and (a(2),b(2)) intersects
% the line segment that connects (a(3),b(3)) and (a(4),b(4)), then
% (z(1),z(2)) is the point of intersection.  If the two line segments
% fail to intersect, then z is the empty vector. Assume that the four points
% are distinct and that the two line segments are not parallel.
```

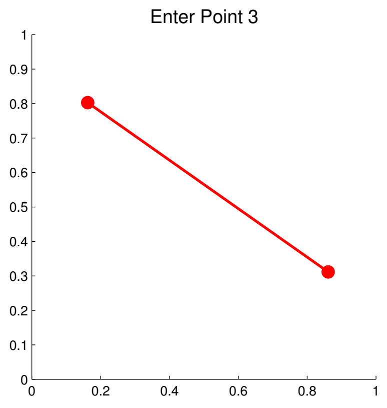
You must also write a mouse-clicking test script `P1` that can be used to check out your implementation of `Intersect`. It should be structured as follows

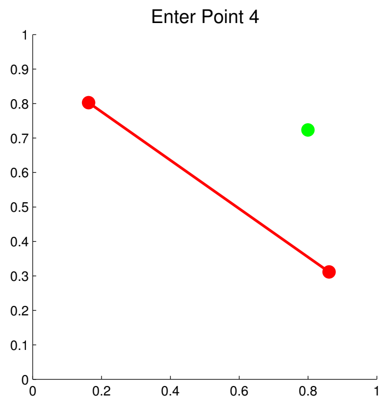
```
close all
figure
axis([0 1 0 1])
axis equal manual
hold on
%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Code that processes four mouseclicks and displays the results, i.e.,
% displays the two line segments (with highlighted endpoints) and the intersection
% point (if it exists)
%
%%%%%%%%%%%%%%%%%%%%%%%%%
hold off
```

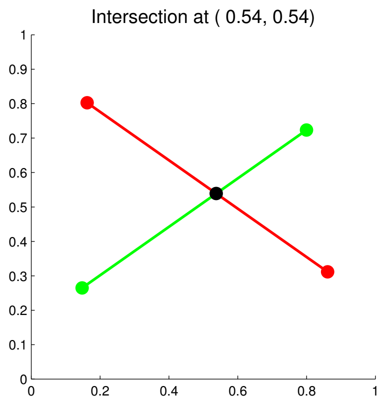
Here are figure window snapshots of what the user of your code should see (more or less) as the points are clicked in:











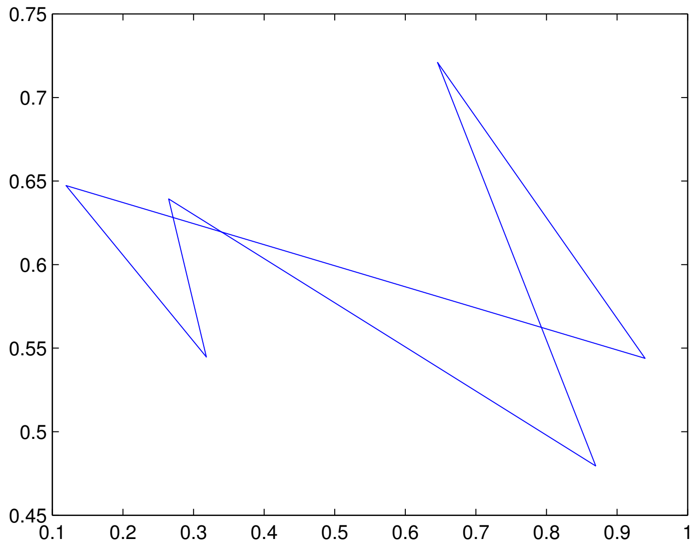
Use `title` for prompts and also for displaying the coordinates of the intersection point if it exists. If the line segments do not intersect, then use `title` to indicate that fact. Color the line segments and intersection point nicely. Set the properties `'Fontsize'`, `'Markersize'`, and `'Linewidth'` to values that make other aspects of the display attractive. Make use of the built-in function `isempty`. You do not have to check to see if the line segments are parallel or that the four points are distinct. You do not have to "force" the mouse clicks to be inside the plot window. Submit `P1` and `Intersect` to CMS.

2 Polygons with Crossing Edges

Assume that `n` is initialized and houses an integer that satisfies $n \geq 4$. The following fragment displays a random polygon with n edges:

```
x = rand(1,n);
y = rand(1,n);
plot([x x(1)],[y y(1)])
```

In general, the polygon will have edges that cross, e.g.,



Formally, we say that a polygon has crossing edges if it has an edge that intersects another edge that is not one of its two "neighbors,". The above hexagon has three crossing points. Complete the following function so that it performs as specified:

```
  function [u,v] = PolygonCrossings(x,y)
% x and y are column n-vectors and n>=4.
% If the polygon with vertices (x(1),y(1)),...,(x(n),y(n)) has no crossing edges, then
% u and v are each empty vectors.
% If the polygon has k crossing points, then u and v are length-k column vectors that
% house their coordinates, i.e., the crossing points are (u(1),v(1)),...(u(k),v(k).
```

Submit your implementation of `PolygonCrossings` to CMS. It should make effective use of the function `Intersect` that you developed in the previous problem. You will have to write a test script–it will not be provided.

3 A GUI for Polygon Untangling

The following fragment generates and displays a random "normalized" polygon with n vertices:

```
x = rand(n,1); x = x - mean(x); x = x/norm(x);
y = rand(n,1); y = y - mean(y); y = y/norm(y);
plot(x,y)
```

To understand `mean` and `norm`, if `z = [3;5;10]`, then `mean(x)` is 6 and `norm(z)` is $\sqrt{3^2+5^2+10^2} = \sqrt{134}$. Suppose we connect the midpoints of the polygon and then normalize:

```
x = (x + [x;x(1)])/2; x = x-mean(x); x = x/norm(x);
y = (y + [y;y(1)])/2; y = y-mean(y); y = y/norm(y);
```

If this process is repeated, then the original polygon with its criss-crossy edges untangles itself and its vertices head towards an ellipse with a 45-degree tilt! In this problem you build a GUI that can be used to illustrate this phenomena.

Start by downloading `ShowAveraging`. Play with it and understand the four subfunctions `GenEG`, `Show` , `AverageOnce`, and `Simulate`.

Next, download `UntangleGUI.fig` and `UntangleGUI.m`. It has limited functionality. In particular, it has only a slider that sets the value of n and a push button that can be used to generate and display a new example. Study their callbacks and then carefully proceed to incorporate these features:

1. Add a push button that triggers one averaging and displays the result. (However, it shouldn't do anything if the displayed polygon has is the result of `itMax` averagings.) It will make use of `AverageOnce` and `Show`. For now, display the limiting ellipse.

2. Add a push button that triggers a simulation. It will make use of `Simulate`. For now, display the limiting ellipse and simulate at the fast rate.

3. Add a check box to control whether or not the limiting ellipse is displayed. Its value should be used by the callbacks of the two push buttons that you just developed. And when the box is checked or unchecked, the current display should reflect the change.

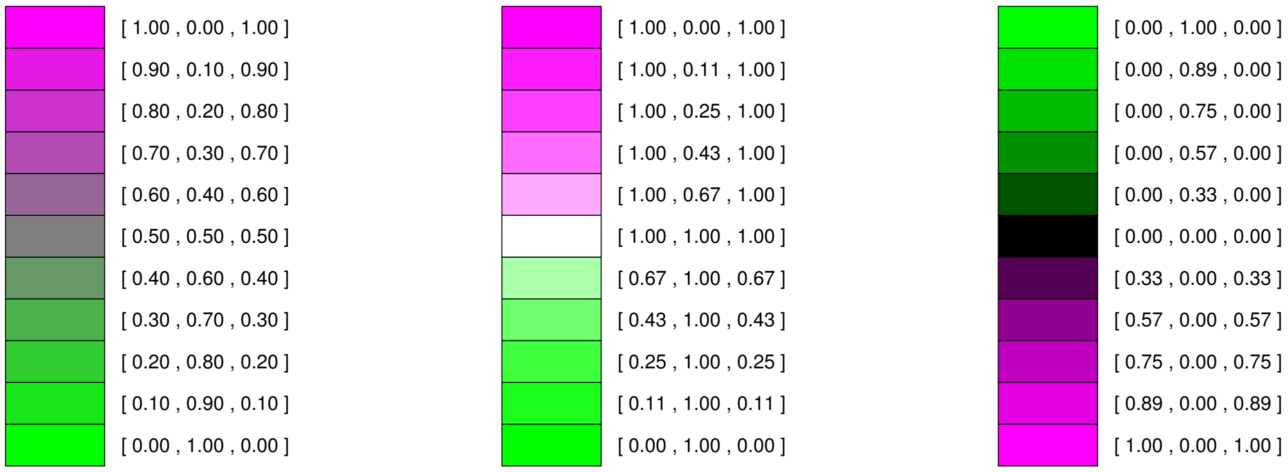
4. Add a button group and two radio buttons that can be used to control whether or not the simulation is to be fast or slow. The callback for the simulate button needs to be modified so that it can sense and make use of the radio button settings.

It is recommended that you proceed with these "improvements" in the order that they are given. After you have developed a new component and it works, make a copy of the GUI files before moving on to the next feature that you want to incorporate.

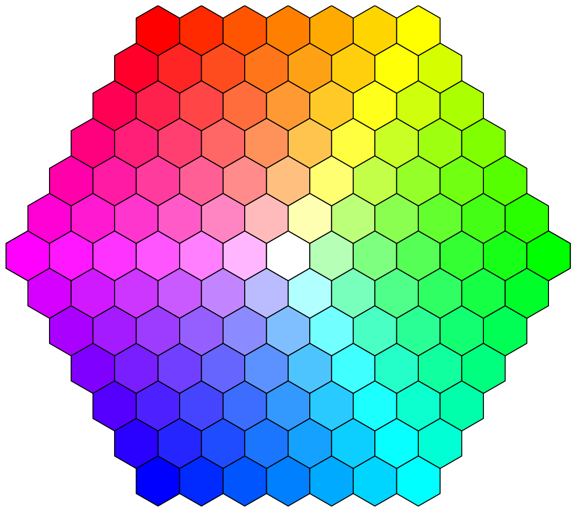
Other comments. Make sure that the opening function sets up all the components nicely. As you add components, pay attention to the "handle variables" that are initialized in the opening function. Various callbacks will have to interact with these variables. And you may have to create some new ones. For guidance about radio buttons, push buttons, and check boxes, play with the working GUIs `ShowRadioButtons` and `ShowButtons` that available on the course website. Submit your final `UntangleGUI.fig` and `UntangleGUI.m` to CMS.

4 Challenge 3: Hexagonal Color Wheels

The script `ColorInterp` illustrates three examples of color computation: interpolation, saturation, and complementation. The leftmost sequence shows how to generate 9 "equally spaced" colors in between green and magenta. The middle sequence shows what happens when these tiles are saturated. For example, $v = [0.80\ 0.20\ 0.80]$ becomes $\texttt{v/max(v)} = [1.00\ 0.25\ 1.00]$. The rightmost sequence displays the complement of each saturated tile. For example, $\texttt{v} = [1.00\ 0.25\ 1.00]$ becomes $[1.00\ 1.00\ 1.00] - v\ =\ [0.00\ 0.75\ 0.00]$.

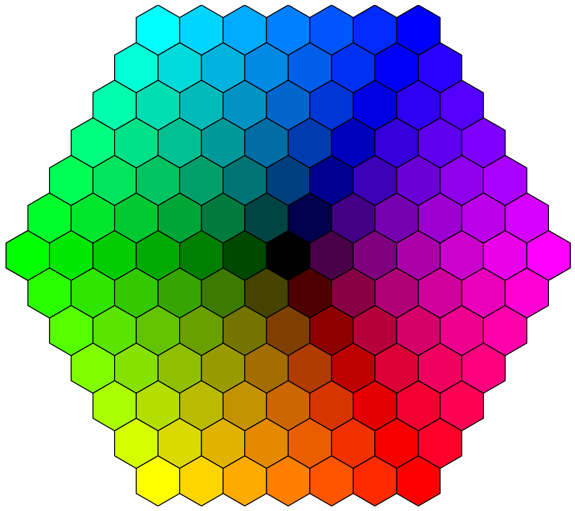


You are to implement a function `C3(N)` that produces two figures. In Figure 1 should be displayed a *saturated hex wheel* of size N. For example, if $N = 7$, then the following graphic should be produced:



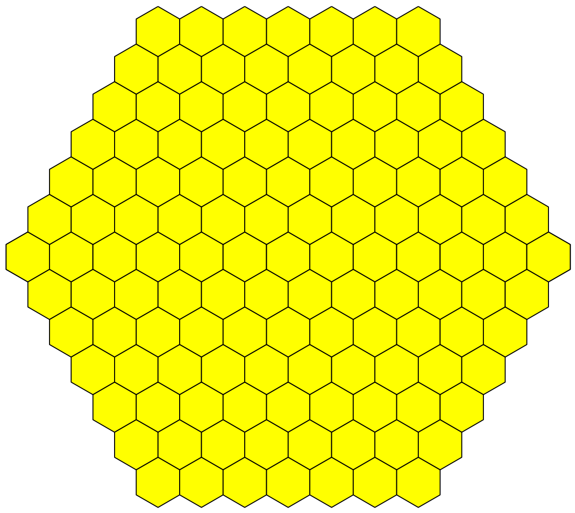
Notice that there are N hex tiles along each side and that the *vertex tiles* are green, yellow, red, magenta, blue and cyan (in that order). Along each edge, interpolation and saturation are used to produce the intermediate colors. Likewise, along each horizontal row, the interior colors are generated by (a) interpolating between the color of the row's leftmost tile and the color of the row's rightmost tile and then (b) saturating the result.

As mentioned, the function `C3(N)` must also produce a Figure 2. In this figure the color complement of the Figure 1 hex wheel is to be displayed:



The color of a given tile in Figure 2 is the complement of the color of the corresponding tile in Figure 1.

A partial implementation of `C3(N)` is available on the course website. It produces two figures, but in each case every displayed tile is yellow, e.g.,



You are free to modify the given subfunctions and to add others. Indeed, your score on this problem will depend on how well you do this. The interpolation-saturation operation is certainly something worth encapsulating as a function. The drawing of the two wheels is very similar, so this presents another opportunity to design a single subfunction that can be used generate both wheels. Submit your final implementation of `C3` to CMS. All the functions you write should be included in this file.