MATH 308 - SECTIONS 503/504 - PROJECT 2

INSTRUCTOR: DR. ADAM LARIOS

Due date: Monday, 5 Nov. 2012

Instructions. While there are no specific neatness requirements, your work should look professional, and you should submit it in a form that would be appropriate for submitting to a boss at a job you care about. Credit will be given for full, complete, and thoughtful analysis; however, a good report is not necessarily a long report. Do not submit fluff or filler, but only quality work that you are proud of.

You are free to choose your partner, so long as it is a different partner than Project 1. Submit it with all group member's names clearly labeled on the front. Make sure graphs are well labeled and referenced, and that they are easy to read. Please note that each person in a group is responsible for 100% of the work, so it is your responsibility to keep all group members on task, or to finish the work yourself. Submit all of your code, along with any relevant graphs, mathematical calculations, and explanations.

By turning in this project, you agree to follow the Aggie Honor Code. In particular, please be sure any code you represent as your own is 100% written by you and your group members. You are free to discuss with other groups, but your code should be your group's own distinct code.

NOTICE: If a group member is not responding or falls out of contact, you need to let me know as soon as possible. If you are stuck on something in Matlab and are spinning your wheels, please email me early so you can get help when you need it, even if you don;t know what question to ask. This project is not demanding, but if you wait until the last week to start it, it is unlikely that you will do well. If you start it early, it should be very manageable.

The goals of this project are:

- To get exposure to dynamical systems, which occur frequently in applications and have many similarities with differential equations. Dynamical systems can also display in a simple way some of the complex behavior which only occurs in much more complicated differential equations than those we have studied so far, so they can also provide nice analogues of difficult differential equations.
- To see a first glimpse of chaotic behavior, which underlies a very wide variety of physical processes modeled by differential equations, such as turbulence, economic forecasting, weather forecasting, and even some seemingly simple mechanical systems.
- To create and explore a fractal set. The word "fractal" comes from the fact that these sets have fractional dimension. The behavior of many differential equations and dynamical systems is often governed by an underlying fractal set (called the "attractor"). Here, we will get some idea of what fractals look like, and how they can be generated.

Part 1: Visions of Chaos

> "Chaos is the score upon which reality is written."
> *-Henry Miller*
>
> "Chaos is a friend of mine"
> *-Bob Dylan*

Discrete Dynamical Systems

Differential equations are "continuous" systems, but there are also "discrete" systems, which are called "discrete dynamical systems." Consider the familiar exponential growth population model with growth rate r. If we let P_n denote the population at the n^{th} time step, then P_{n+1}, the population at the $(n+1)^{\text{st}}$ time step is given by

$$P_{n+1} = rP_n$$

Warm up

Suppose $r = 0.17$, and the population starts at the "seed value" given by $P_0 = 3$. In Matlab, compute P_{10} as follows.

```
P = 3;
for n = 1:10
  P = 0.17*P;
end
P
```

Of course, this will not store any of the values, so if we want to store them (so we can use them to plot, for example), we can do this:

```
P(1) = 3;
for n = 1:10
  P(n+1) = 0.17*P(n);
end
plot(P);
```

(If we only pass one vector `P` to `plot`, then Matlab uses just the usual counting numbers for the x-axis, `[1 2 3 ... length(P)]`). Notice that, since Matlab can't start the index at 0, we have to start it at 1. Also, it is better to preallocate `P` by setting `P=zeros(1,10)` before the loop. This makes Matlab run faster, since it sets aside the space it needs beforehand.

To get used to `if` statements, try the following code.

```
a = 4;
b = 3;
if (a<b)
    j = -1;
elseif (a>b) & (a~=0) & (b==3)
    j = 2;
else
    j = 3;
end
j
```

Try varying `a` and `b` in the above example to get an idea of how the if statement behaves. There are many examples of these tools online as well, if you need more examples.

Recall that we learned about the logistic equation in class in Section 2.5:

$$\frac{dP}{dt} = r \cdot P \cdot \left(1 - \frac{P}{K}\right)$$

where r is the effective growth rate, and K is the carrying capacity. We can consider a discrete version of this equation (setting $K = 1$ for simplicity), namely

$$P_{n+1} = r \cdot P_n \cdot (1 - P_n)$$

This seemingly simple dynamical system exhibits chaotic behavior, in the sense that tiny changes in the parameters can lead to very different long-term outcomes. This means that, if we want to predict the behavior, even if our measurements of these parameters is very precise, but not perfect, the long-term behavior is essentially unpredictable. We will explore this in the exercises below.

Exercises

(1) To get started, choose the seed value $P_0 = 0.5$, and set $r = 1.61$, and repeat this process, say, 250 times to find P_{250}. Remember to only output P_{250}, since you don't want a mess showing up. (There is nothing special about the 250 here, we just need a large number.)

(2) Next, let's try varying r. Try the above exercise plugging in $r = 2.6$, $r = 2.61$, $r = 2.611$, $r = 2.6111$, $r = 2.61111$, $r = 2.611111$, outputting only P_{250} each time. Pay attention to (and record) the values as you go. We are only slightly varying r, and the output results are not very surprising.

(3) Try the above exercise again, but this time with $r = 3.6$, $r = 3.61$, $r = 3.611$, $r = 3.6111$, $r = 3.61111$, $r = 3.611111$. What the heck just happened?

(4) OK, that was weird. Let's try to get a better picture of things by looping over a wide range of r values. Calculate P_{250} for 1000 different r values, ranging between $3 \leq r \leq 4$ (`linspace` would be a good tool to use here), and save them as you go in a big vector. Plot your r values against your P_{250} values. What do you see? How are the points changing as r increases?

(5) Try changing your seed value $P_0 = 0.5$ a little. Do you see different behavior? If not, try $P_0 = 0.7$ and $P_0 = 0.9$. What do you notice?

(6) OK, it's time to sort everything out. Let's make a big loop that runs over everything, looping from seed P_0 from 0.1 to 0.9 using, about 100 values or so. Put this all on the same plot by declaring "`hold on`" somewhere near the top of your document. If you want to watch it draw each plot, put `pause(0.1)` in your outer loop. **Don't forget to go full screen and zoom in!**

Matlab automatically connects the dots between plotted points. Try plotting with points, to see things easier, like this:

```
plot(r_vals,P_vals,'.');
```

What you just built is called "The orbit diagram for the logistic family." It illustrates that the long-term behavior of this system is independent of initial conditions, and highly dependent on the parameters.

Part 2. Dreams of Order

> "'Chaos was the law of nature; Order was the dream of man."
> *-Henry Adams*
>
> "In the space between chaos and shape there was another chance."
> *-Jeanette Winterson*

Consider the discrete dynamical system given by

$$x_0 = 0$$
$$x_{n+1} = x_n^2 + c$$

Here, we will always have the same seed value $x_0 = 0$, but consider different values of c.

(1) Notice that for most values of c, $|x_n|$ gets very large after about 20 iterations or so. Try this out for 5 different values of c.

(2) For *some* values of c, x_n stays bounded forever, such as when $c = -0.5$. Find the values of c which make x_n stay bounded by looking at several thousand values of c. There are many ways to approach this, try to figure out a good way to do it! (Hint: Narrow your search to only values of c between -3 and 2.) You can assume that x_n will become unbounded once $|x_n|$ gets larger than 2 after 20 iterations, and that x_n will stay bounded otherwise. Find the left and right endpoints of the bounded region out to 2 decimal places.

(3) Next, consider the case where the numbers are complex. We have the same setup:

$$z_0 = 0$$
$$z_{n+1} = z_n^2 + c$$

Try to find the values of c which make z_n stay bounded when you allow c to be complex. To see if z_n is becoming unbounded, you need to see how large it is. You can find the "size" of a complex number $z = a + bi$ by computing its "modulus" or absolute-value $|z| = \sqrt{a^2 + b^2}$, which is written as

just `abs(z)` in Matlab . Again, you might want to narrow your search to look near the origin. You may need two nested loops now, one for the real part, and one for the imaginary part. Keeping track of $|z_n|$, you can again assume that z_n will become unbounded once $|z_n|$ gets larger than 2 after 20 iterations. This region is *much* more complicated, so you don't need to describe it very exactly. Just to find 5 points in the region, and 5 points outside of it (use non-real numbers for this part, since you already did this for real numbers in the last exercise). **Note: Matlab understands "i" as an imaginary number. Make sure you DO NOT use `i` as a variable or an index anywhere in your code, or you will have errors!**

While there are many ways to do the above exercise, one way is to start with a big matrix `A` of zeros. You can then use an if statement to check if `abs(z)` is larger than 2 after 20 iterations. If it is, you can set the corresponding matrix value to 1. Otherwise, leave it as zero. You can then see what this matrix looks like by using `spy(A)`, which shows the non-zero values of your matrix. If you do it this way and include the resulting plot (using at least a resolution of 100×100, you do not need to report the numerical values.

(4) Let's get a better picture of the situation. Instead of marking which c-values are unbounded and which one do not leave, let's instead look at $|z_n|$, and just color it different colors based on its size. We can also simplify our work by putting points in the plane into a big matrix, and operating on them all at once. To do this, use the following Matlab code:

```
x = linspace(x_center - x_length,x_center + x_length,resolution);
y = linspace(y_center - y_length,y_center + y_length,resolution);
[X,Y]=meshgrid(x,y);
c = X + i*Y;
z = zeros(size(c));
```

You have to put it numbers for the `y_length`, `x_center`, `resolution`, and so on. This will store all of your grid-points in the big matrix `c`. Now, run the iteration about 20 times, making sure to square things component-wise, like this: `c.^2`, instead of like this: `c^2` (`c^2` means matrix multiplication, which we don't want here). (Hint: The iteration is very simple, and will likely only take you three lines of coding.)

To plot this, we'll use the `pcolor` function. The values of `z` are too far apart for things to look nice, so we'll compress them using a decaying exponential function.

```
    w = exp(-abs(z)); % This is just to make the colors look nice.
    pcolor(w);        % This plots the colors.
    shading flat;
    axis('square','equal','off');
```

Make sure your resolution is high! You want at least 100×100 resolution, but Matlab can easily handle 1000×1000, which looks much nicer.

(5) Notice the extreme amount of complexity created from such a simple iteration. To see more of this complexity, change the values of `x_center`, `x_length`, `y_center`, and `y_length` to "zoom in" on particular parts of the picture. Note that this is not the same thing as clicking on the "zoom in" button, since you are recalculating the values. Find an interesting region of the picture, and include it in your project (this portion will only be graded based on whether or not you zoomed in on something, there is no need to worry about not finding something interesting enough).

(6) Fun stuff (do one or more of these):

(a) Try putting `colormap autumn(256);` before `pcolor`. You can google "matlab colormap" to find many neat-looking colors.

(b) Try putting `pause(0.2)` into your loop just before or after the `pcolor` command to watch as Matlab calculates each iteration. .

(c) What happens if you change the function in the iteration? What new shapes can you create?