Mathematica Project 1

This is the first of three Mathematica projects you will be doing in the class. You must turn in the assignment portion in class on **Wednesday March 21st**. Only knowledge of material from the course up through February 28th is assumed, although if you wish to incorporate things learned since then, feel free. Mathematica commands will usually be denoted by this typeface.

Getting Started: The first thing you need to do is open up Mathematica. It is preinstalled on some campus computers already. If you are using your own computer, please see the installation instructions on OnCourse.

When you first open Mathematica, the first thing you should should do is play around a little bit in the notebook. There is a tutorial for beginners (`http://reference.wolfram.com/language/tutorial/GettingStartedOverview.html`), but most of this you can pick up as you go along. A few mathematica specific things to keep in mind are the following:

- The workstation of Mathemtica is called a *notebook*. Lines in the notebook are called *cells*, which are numbered, and further subdivided into input (what you type) and output (what you get when you evaluate). You can save your current notebook as a *.nb* file. When opening a *.nb* file, Mathematica will display outputs of cells evaluated at last save, but they will need to be evaluated again if you defined things in them.
- Mathematica treats the two Enter keys on a keyboard differently. The 'Text Enter' key (the one in the middle of they keyboard sometimes called 'Return') produces line breaks. The 'Numeric Enter' key (the one by the number pad on the right if you have one) evaluates a cell immediately. To evaluate a cell using the Text Enter key, hold down Shift+Enter.
- Mathematica will ignore line breaks and blank spaces, so you can organize inputs with those if you so please.
- Mathematica uses all the usual inputs +, -, /, *, ^ for elemtary operations and uses parenthesis for grouping terms. Square brackets are used for functions, and curly brackets are used for lists.
- Mathematica will turn expressions blue to denote that what you have entered has not been defined.
  Ex. if you haven't defined the variable X, the first time you write it will display X.
- Mathematica is **very** picky about capital letters. You will sometimes find if you wrote something with incorrect capitalization that it will be in blue, as it is not defined. Most predefined functions and values begin with capitals. Ex. If you wish to enter π and type pi (all lowercase), it will display as pi, which should clue you in to the fact that it should instead be entered as Pi. Similarly entering for e^x you must type E^x and not e^x.
- Functions are defined by square brackets around the inputs, ex. Sin[Pi/2].
- Adding a semicolon ; at the end of the line will have it execute without displaying the output. You should use this **extensively** to clean up your code before turning it in to me. Don't send me 100 page packets.
- The letter N is reserved, and is the function that outputs a decimal approximation. Ex. N[Sin[1]].
- You can use equals = to set values for previously undefined variables (they don't need to be declared). Ex. X = 1.
- You can use colon equals := to define functions. You need to specify the inputs of the functions with an underscore _ and separate them by commas , inside of a pair of brackets. If done correctly, the new function name will be displayed in blue and the variables will be in *green italics*. Ex. Entering Combine[X_, Y_] := X + Y defines a function called *Combine* that adds together two inputs and can be called at will. Ex. Combine[1,2].
- You can clear previously defined variables and functions. Ex. Clear[Combine] deletes the function defined above.
- You can add comments to your code by writing them in between star and parenthesis. The text will appear greyed out and will do nothing. Ex. (* This is an example of a comment *). You should strive for well commented code.
- Save often.

If you get stuck, Mathematica has very robust documentation (`reference.wolfram.com/`). and is very google-able.

Mathematica Commands and Structures Useful for This Assignment (.nb file included)

Lists (documentation)

Ordered lists in mathematica are created using curly braces { and } with entries separated by commas.
For example, we can create a two-element list $\{1, -1\}$ as follows:

Enter: `L = {1, -1}`

We can get individual list elements via double brackets, with index starting at 1. For example,to return -1,
Enter: `L[[2]]`

To add elements to the end of a list, we can use `Append[ list , element ]`, which takes two arguments
For example, if we would like to add 2 to our list L,

Enter: `L = Append[L, 2]`

Mathematica views most mathematical structures (such as sets, points, vectors, and matrices) as lists.
If we wanted to make a set of points in Mathematica, we would to enter it as a list of lists.
For example, we can build the set $S = \{(0, 1), (1, 2)\}$,

Enter: `S = { {0,1} , {1, 2} }`

And hence, to add in the point $(2, 7)$ into it, we can use append.

Enter: `S = Append[S, {2, 7} ]`

Notice now that in order to output the value 7, we need to grab the 2nd entry of the 3rd entry in our list.
Enter: `S[[3]][[2]]`

Loops (documentation)

To run iterations we will need to use loops. Mathematica can do loops in several ways, but I find it easiest to use a for loop, which has the structure `For[ start, test, increment, body ]`.

- *start* is an initial state of the index, usually entered as something such as `i = 1`.
- *test* is an expression including `i`. The loop stops when the expression is false. Usually something like `i < 11`.
- *increment* is by how much `i` increases each time the loop finishes. Use `i++` to increase `i` by one each time.
- `body` is the code you actually would like run repeatedly.

For example, if we wanted to make the set $\{ (0, 0), (1, 1), (2, 4), (3, 9), \ \ldots \ , (10, 100) \}$, we should use a loop.
We can start with a list of just $(0, 0)$, then use a loop to append the rest as follows,

Enter:
```
NewList = { {0,0} }
For[ i = 1,  i < 11,  i++ ,
    NewList = Append[  NewList , { i, i^2 } ]
]
```

Now if you query `NewList`, you should have the set we wanted.

Be **VERY** careful with loops. they are computationally cumbersome and can easily cause major issues.
A great way to crash Mathematica is to run a loop on a slightly incorrect expression.
Before you evaluate a cell with a loop with a test like `i < 100`, try it with `i < 3` and make sure there are no errors.

Again, save often.

Continuous Graphs (documentation)

There are many ways to graph things in Mathematica. We will only use a few types, and they all have similar structure.

For a function, you can use Plot [*f[t]* , {*t*, t_{min}, t_{max}} *]* , *options*]. If done correctly, the parameter should be tinted *teal*.

For example, to plot $y = e^x$ on $[0, \pi]$,

Enter: Plot[Sin[E^*x*, {*x*, 0, Pi}]

To adjust the picture, we can use a few options. They are entered as *option* $->$ *setting*. Some useful ones are:

- AxesOrigin, which is often best set to { 0, 0 }. If your graph seems incorrectly centered, try using this.
- PlotRange, which can be used to set maximum and minimum horizontal and vertical variables displayed in the graph, usually as a list of of lists. For example using PlotRange $->$ { {1,2 }, {3, 4} } will make the graph display horizontally from 1 to 2 and vertically from 3 to 4.
- PlotStyle, which has a ton of options, but I find it is most helpful for assigning color to your graph.

Discrete Graphs (documentation)

We will wish to graph a list of points on the plane. For such a list we can use: ListPlot[*listname* , *options*] .

For example, to graph our list of points and their squares from earlier,
Enter: ListPlot[NewList]

ListPlot[ ] has the same options as Plot[ ] above as well as some specific to discrete lists.
The most important one is Joined, which will connect your dots using line segments if set to True. for example,

Enter: ListPlot[NewList , Joined $->$ True]

Slope/Vector Fields (documentation)

We will like to graph slope fields. Unfortunately, Mathematica prefers vector fields, so we have to do something a little bit annoying. If we have an ODE given by $\frac{dy}{dx} = F(x, y)$, we can graph its slope field by :

VectorPlot[{*1, F[x, y]*} , {*x*, x_{min}, x_{max}}, {*y*, y_{min}, y_{max}} , VectorStyle $->$"Segment" , *options*] .

For example, for the ODE $\frac{dy}{dx} = y$ in the window from $[0, 2] \times [0, 5]$
Enter: VectorPlot[{*1,* y} , {*x*, 0, 2}, {*y*, 0, 5} , VectorStyle $->$"Segment"]

Combining Graphs (documentation)

We can display multiple graphs on top of each other using the Show[ ] command, sparating graphs with commas. Color coding helps.

```
Enter: Show[
         VectorPlot[{1, y} , {x, 0, 2}, {y, 0, 5} , VectorStyle -> "Segment"]
         ListPlot[ NewList , Joined -> True ]
         Plot[ Sin[E^x, {x, 0, Pi} ]
       ]
```

Application: Predator-Prey Modeling

For this project, we consider we will work with a model of a predator-prey system as in the Exam 1 bonus.
If you want to know more details, a good place to start is pgs. 108-109 in our book.

Consider Capybaras (world's largest rodents), which are hunted by Jaguars (big cats).
Let $x(t) \geq 0$ be the number of Capybaras in an ecosystem after t months, and $y(t) \geq 0$ be the number of Jaguars.

The so-called Lotka-Volterra Equations that model the equations are as follows:

$$\frac{dx}{dt} = \alpha x - \beta xy$$
$$\frac{dy}{dt} = \delta xy - \gamma y$$

They can be interpreted as follows:

- With no Jaguars, Capybaras reproduce proportionally to number of Capybaras, at a constant rate $\alpha > 0$.
- With no Capybaras, Jaguars starve to death proportionally to number of Jaguars, at a constant rate $\gamma > 0$.
- The rate at which Capybaras are hunted is proportional to the number of Jaguar-Capybara pairings (the product of the number of Capybaras and the number of Jaguars), at a constant rate $\beta > 0$.
- If the Jaguars have food to hunt, they will reproduce proportionally to the number of Jaguar-Capybara pairings (the product of the number of Capybaras and the number of Jaguars), at a constant rate $\delta > 0$.
- The critical points $(0, 0)$ and $\left(\frac{\gamma}{\delta}, \frac{\alpha}{\beta}\right)$ correspond to mutual extinction and equilibrium respectively. Their behavior is more complicated than our usual attractor/repeller/semi-stable critical points for autonomous ODEs.

This is a system of non-linear ODEs, and thus at the moment we can not solve them. Hence, we will *approximate* them. (the solutions to these can not be expressed in elementary functions, and requires knowledge far beyond our course).

To approximate, notice that t is the only independent variable, and both x and y are dependent variables.
In the case of using an Euler's method, we must increment t and take linear approximations of both x and y.

Your Assignment: We will consider an Euler's method approximation with the following parameters:

$$\alpha = 0.80 \qquad \beta = 0.05 \qquad \gamma = 0.85 \qquad \delta = 0.02 \qquad x_0 = 100 \qquad y_0 = 30 \qquad h = 0.01$$

- Write out a system of recursive expressions for x_{n+1} and y_{n+1} in terms of $\alpha, \beta, \gamma, \delta, x_n, y_n$ and h.
- Compute by hand (x_1, y_1), and (x_2, y_2).
- Enter your expression into Mathematica. Check that you get the same (x_1, y_1) and (x_2, y_2) as you did by hand.
- Use Mathematica to find data approximate at least two years of data.
- Create two lists of paired data. One for (t, x) and one for (t, y). Graph them on the same axis. This displays how the populations change over time. State any observations about the graphs.
- Create a list of paired points (x, y), and graph this list. This displays how the populations change relative to each other. Note the equilibrium is a point in the (x, y) plane. State any observations about the graphs.
- Give some sort of interpretation of what is going on with the Jaguars and Capybaras in our ecosystem.
- If you are feeling ambitious, try more accuracy or more iterates and compare to your previous approximations (Not required for full credit).

You must turn in a cleaned-up print out of your Mathematica notebook .nb file (be sure to supress large outputs before turning it in) and your hand calculations, observations, and interpretations. If you wish to write in these things into the comments of your code instead, that is acceptable as well.

This is due in class on **Wednesday March 21st**. I suggest you start early.

Do not seek my help unless you have attempts/code to show me.

Feel free to discuss with your classmates as well, but be sure that the work you turn in is your own.

As is the nature of numerical methods, an error in coding can easily crash Mathematica when iterating a large number of times. As such, **save frequently and expect crashes**. Don't lose everything over a misplaced symbol.