Honors Differential Equations **NAME: ______________________________________**

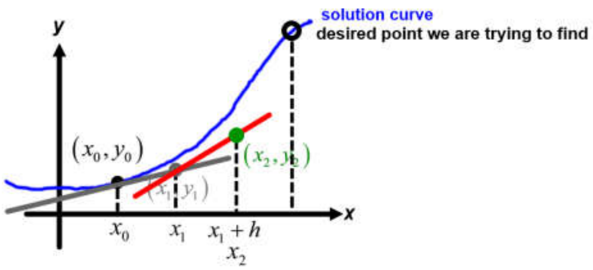
OCTAVE Notes/Programming Packet

We use numerical analysis to obtain solution curves when we don't have a method

Some differential equations can be solved using the methods we've learned in this class and we can obtain a solution function. But some differential equations we can't solve using our known method, so we resort to numerical analysis, using a computer. Remember this from earlier?...

Euler's Method

We then use this new point (x_1,y_1) to establish the slope for another estimate moving closer to the desired point by plugging this point into the original DE $f(x,y)$ function to establish the next slope:



This process continues iteratively until we are at the desired x value, and we then have an estimate of the (x,y) on the solution curve at the desired x value.

We use numerical analysis to obtain solution curves when we don't have a method

Use Euler's method to obtain a four-decimal approximation for $y(1.5)$

on the solution curve for $y' = 0.2xy \;\; with \;\; y(1) = 1$

Let's set h = 0.1, taking 5 iterations to reach from x=1 to x=1.5:

interval	$y_{n+1} = y_n + f'(x_n) \bullet \Delta x$
$(1,1)$	$y = 1 + [0.2(1)(1)] = 1.02$
$(1.1,1.02)$	$y = 1.02 + [0.2(1.1)(1.012)] = 1.04244$
$(1.2,1.042)$	$y = 1.04244 + [0.2(1.2)(1.04244)] = 1.0674...$
$(1.3,1.067)$	$y = 1.0674... + [0.2(1.3)(1.0674...)] = 1.0952...$
$(1.4,1.0952)$	$y = 1.0952... + [0.2(1.4)(1.0952...)] = 1.125878...$
$(1.5,1.1259)$	$so \;\; y(1.5) \approx 1.1259$

This is laborious to do by hand. Also, there are other methods which are more accurate and converge to the solution more quickly than Euler's method, but are even harder to do by hand, so we resort to using a computer.

OCTAVE - a free, open-source version of MATLAB

We will learn a little bit of MATLAB programming in order to be able to solve DEs we can't solve in other ways, however we will use a program called OCTAVE, which is an open-source version of MATLAB available for free to everyone. You can download and install this program on any computer, so learning this gives you a tool you can use in the future whenever needed.

The intent in this unit is to have you learn sufficient OCTAVE programming skills to be able to use the program to solve differential equation problems. Everyone needs to be able to participate by entering and running OCTAVE programs in the programming environment program installed on a computer.

We will start with an overview of the programming environment, then you can work individually or in pairs to complete the OCTAVE Programming Assignment Packet, which will walk you through a series of increasing complexity examples so you can learn and practice programming skills.

Throughout this packet there will be places for you to show your work. These are marked by boxes with cross-hatched edges like this:

This unit is graded and to get full credit, you need to complete each of the numbered steps, in order, and then to fill in each solution box like this throughout the packet with your solutions.

Step 1) Solve example #1 using manual methods (Solving a 1st-order single differential equation):

Let's start with a simple first-order linear differential equation that we do have methods to solve to see how OCTAVE programming can replicate what we already know how to do.

$$\frac{dy}{dx} = x - 2y \qquad y(0) = 3$$

We could solve this with an integrating factor:

$$\frac{dy}{dx} + 2y = x$$

$$IF = e^{\int 2dx} = e^{2x}$$

$$e^{2x}y = \int xe^{2x}dx \quad (by\ parts)$$

$$e^{2x}y = \frac{1}{2}xe^{2x} + \frac{1}{4}e^{2x} + C$$

$$y = \frac{1}{2}x + \frac{1}{4} + Ce^{-2x}$$

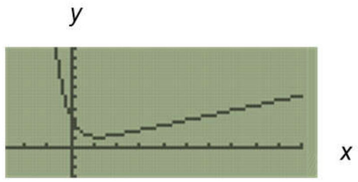
Using the initial condition:

$$y(0) = 3$$

$$3 = \frac{1}{2}(0) + \frac{1}{4} + Ce^{-2(0)}$$

$$C = \frac{13}{4}$$

$$\boxed{y = \frac{1}{2}x + \frac{1}{4} + \frac{13}{4}e^{-2x}}$$



Example #1 - now using MATLAB/OCTAVE

Now we will write files that MATLAB/OCTAVE (I'll just call it OCTAVE from here) can use to solve and graph this solution curve by numerical approximation.

Whenever we want OCTAVE to run a numerical analysis on a DE, we need to create two files. They both have the file extension .m:

A 'script' file which is main executable file for OCTAVE and gives it all the directions needed to produce the graphed output.

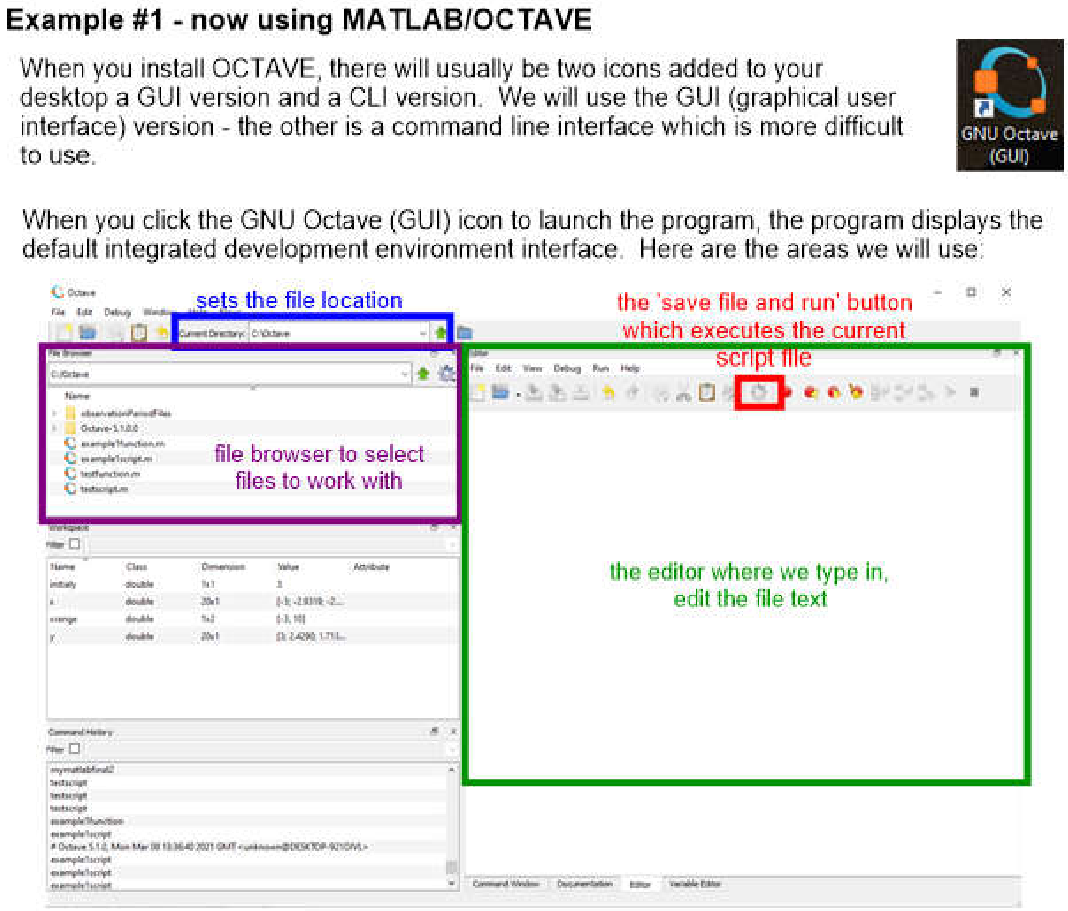
A 'function' file which is called by the script file and contains the definition of the differential equation we are solving.

The particular software routine we are using is called ode45 which is a predefined programmed subroutine that implements the numerical approximation steps. We just need to call this function in the right way, connect to it appropriately using inputs and use the output to build a graph of the resulting solution curve.

Step 2) Install OCTAVE and verify it is working properly

Step 2a) If you haven't already installed OCTAVE, obtain the installer program for your operating system from the GNU OCTAVE project open-source website and install it on your computer.
- If you are using your own computer: Search online for 'GNU Octave' to find the open-source project's page, click 'download' and download an installer program for your computer (probably the latest Windows-64 installer).
- If you are using a school laptop: talk to your teacher about how to get access to OCTAVE.

Step 2b) Open OCTAVE and locate your file directory.



Example #1 - now using MATLAB/OCTAVE

When you install OCTAVE, there will usually be two icons added to your desktop a GUI version and a CLI version. We will use the GUI (graphical user interface) version - the other is a command line interface which is more difficult to use.

When you click the GNU Octave (GUI) icon to launch the program, the program displays the default integrated development environment interface. Here are the areas we will use:

sets the file location

the 'save file and run' button which executes the current script file

file browser to select files to work with

the editor where we type in, edit the file text

Step 2c) Download two starting 'test' files from the mrfelling.com website to your computer's OCTAVE file directory and use them to test your OCTAVE installation.
- Locate your local computer OCTAVE file directory by looking at the file location indicated at the top of the OCTAVE program window.
- Open a web browser and browse to: mrfelling.com/octave
- Right-click on each of the two test files (testfunction.m and testscript.m) and download them into your local computer's OCTAVE file directory.

Step 2d) Test your OCTAVE installation to make sure it is set up properly and functioning.
- In the OCTAVE File Browser window, double-click on 'testfunction.m'. This should open it in your editor window.
- In the OCTAVE File Browser window, double-click on 'testscript.m'. This should add it to your editor window and display this file.
- With the testscript.m file displaying in your editor window, click the 'save file and run' button (looks like a gear).
- A new window should popup which eventually displays a spiral. If this displays, you are good to go!
If this doesn't work, ask your teacher for help before proceeding.

Step 3) Make new copies of the test files and edit them to use OCTAVE to solve the differential equation in Example #1

Step 3a) Make new copies of the test files for this problem:
- Click the tabs at the top (if needed) to display the ‘testscript.m’ file, then click ‘File’ > ‘Save File As…’ to create a copy called ‘example1script.m’ in your file directory.
- Click the tabs at the top of the Editor window to switch to the ‘testfunction.m’ file, then click ‘File’ > ‘Save File As…’ to create a copy called ‘example1function.m’ in your file directory.

You should now be editing the two new files in your Editor.

Step 3b) Edit the example1function.m file to make it represent the differential equation in Example #1.
- Replace what is there with the following:

```
function returnValue = example1function (x, y)
    returnValue=x-2*y;
 endfunction
```

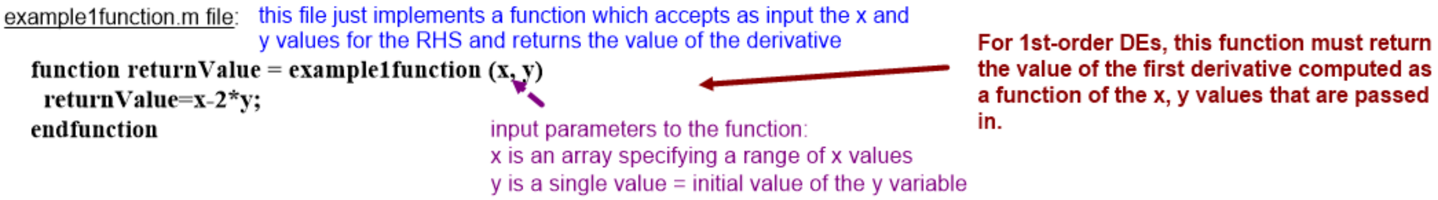
- Save this file by clicking ‘File’ then ‘Save File’ (Note: the name of the function in the first executable line of this file must match the name of the file).

Step 3c) Edit the example1script.m file to make it generate a function curve plot for Example #1.
- Replace what is there with the following:

```
clear all
clc
xrange=[0,10];
initialy=3;
[x,y]=ode45(@example1function,xrange,initialy);
plot(x,y)
ylabel('Y')
xlabel('X')
```

- Save this file by clicking ‘File’ then ‘Save File’.

Step 4) Understand what these new files are doing…



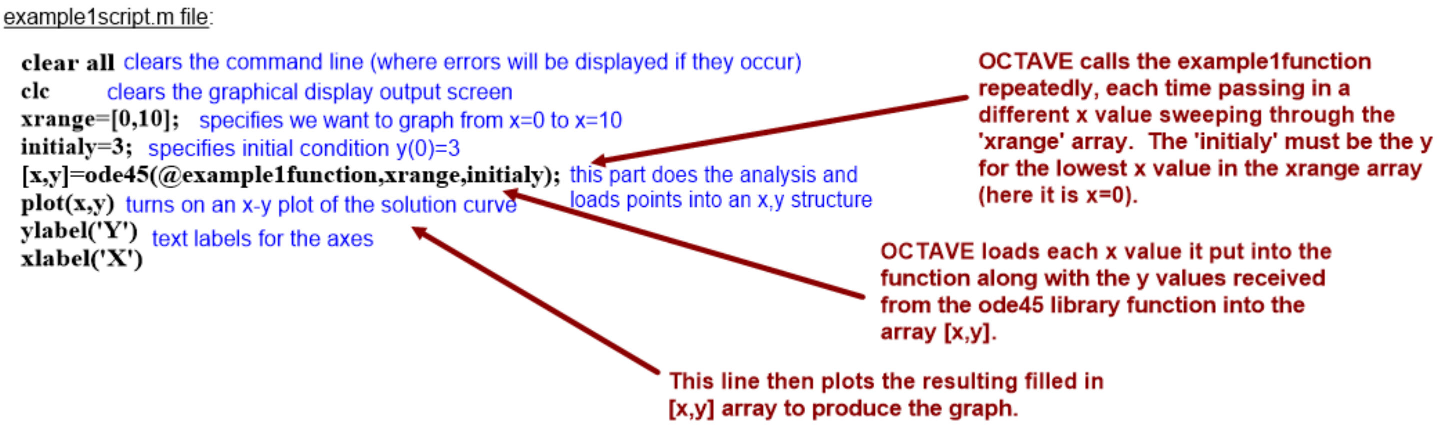
example1function.m file: this file just implements a function which accepts as input the x and y values for the RHS and returns the value of the derivative

```
function returnValue = example1function (x, y)
  returnValue=x-2*y;
endfunction
```

For 1st-order DEs, this function must return the value of the first derivative computed as a function of the x, y values that are passed in.

input parameters to the function:
x is an array specifying a range of x values
y is a single value = initial value of the y variable

example1script.m file:



```
clear all  clears the command line (where errors will be displayed if they occur)
clc        clears the graphical display output screen
xrange=[0,10];  specifies we want to graph from x=0 to x=10
initialy=3;  specifies initial condition y(0)=3
[x,y]=ode45(@example1function,xrange,initialy);  this part does the analysis and
                                                  loads points into an x,y structure
plot(x,y)  turns on an x-y plot of the solution curve
ylabel('Y')  text labels for the axes
xlabel('X')
```

OCTAVE calls the example1function repeatedly, each time passing in a different x value sweeping through the 'xrange' array. The 'initialy' must be the y for the lowest x value in the xrange array (here it is x=0).

OCTAVE loads each x value it put into the function along with the y values received from the ode45 library function into the array [x,y].

This line then plots the resulting filled in [x,y] array to produce the graph.

Step 5) Run the OCTAVE program

Step 5a) Now we'll execute the analysis by clicking the 'Save File and Run' button (in the Editor window, it looks like a gear). You must have the 'script' file displaying in the Editor window when you click the run button. The result should be a plot window displaying the solution curve (it is a separate window which if already open might be covered up by the OCTAVE programming environment window).

If your program did not display the solution function curve, debug your program. In the Editor window, at the bottom, there are tabs to switch between the Editor and Command Window:

Switch to the Command Window. If the program is not running, it will display error messages here which can help you pinpoint where the error might be located.

If you are unable to debug your program, ask your teacher for help.

Once your program runs successfully and displays a graph of the solution function curve, sketch the graph produced by OCTAVE below. It should roughly match the curve you graphed when you solved Example #1 by hand earlier.

Step 6) Example #2: Now use OCTAVE to solve a new 2^{nd}-order homogeneous differential equation

Next, let's analyze one of the mass-spring problems from our ch5 homework:

A mass weighing 8 pounds is attached to a spring. Because there is no damping, when set in motion, the spring/mass system exhibits simple harmonic motion. Determine the equation of motion (graphical appoximation) if the spring constant is 1 lb/ft and the mass is initially released from a point 6 inches below the equilibrium position with a downward velocity of 1.5 ft/s.

Find the mass: $weight = mg$

$$8 = m(32)$$

$$m = \frac{8}{32} = \frac{1}{4} slug$$

$$mx'' + \beta x' + kx = f(t)$$

Here: $m = \frac{1}{4},\quad \beta = 0,\ k = 1,\quad f(t) = 0$

$$x(0) = \frac{6}{12} = 0.5,\quad x'(0) = 1.5$$

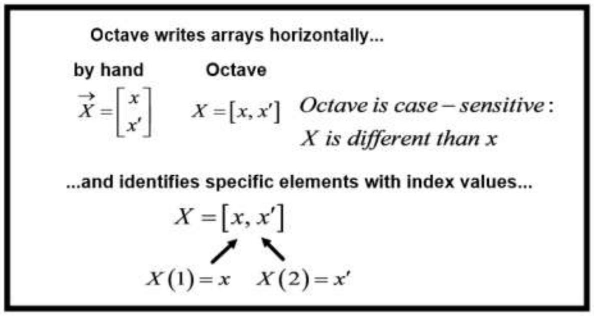
So our DE is: $\frac{1}{4}x'' + x = 0$

OCTAVE ode45 can only handle first-order DEs so we need to express this as a system of first-order DEs first:

Step 6a) Write the 2^{nd}-order DE written as a system:

$$\begin{cases} x' = \quad 0x + 1x' + 0 \\ x'' = -4x + 0x' + 0 \end{cases}$$

Step 6b) Understand how to format the system as an array and build the returnValue statement:



Octave writes arrays horizontally...

by hand $\vec{X} = \begin{bmatrix} x \\ x' \end{bmatrix}$

Octave $X = [x, x']$ *Octave is case − sensitive : X is different than x*

...and identifies specific elements with index values...

$$X = [x, x']$$

$X(1) = x \quad X(2) = x'$

We need to represent our system using OCTAVE's array format:

$$\vec{X}' = \begin{bmatrix} 0 & 1 \\ -4 & 0 \end{bmatrix} \vec{X} \quad initial\ condition : \vec{X}(0) = \begin{bmatrix} 0.5 \\ 1.5 \end{bmatrix}$$

The function.m file needs to always return the derivative of its input:

For a single DE

$\underline{input} \rightarrow function\ file \rightarrow \underline{output}$

$$(x, y) \quad \rightarrow \quad \frac{dy}{dx}$$

$returnValue = x - 2 * y;$

For a system, we must reference array elements

$\underline{input} \rightarrow function\ file \rightarrow \underline{output}$

$X = [x, x'] \quad \rightarrow \quad X' = [x', x'']$

$output\ x' = input\ x'$

$output\ x'' = -4x$

using array elements, becomes...

$output\ x' = X(2)$

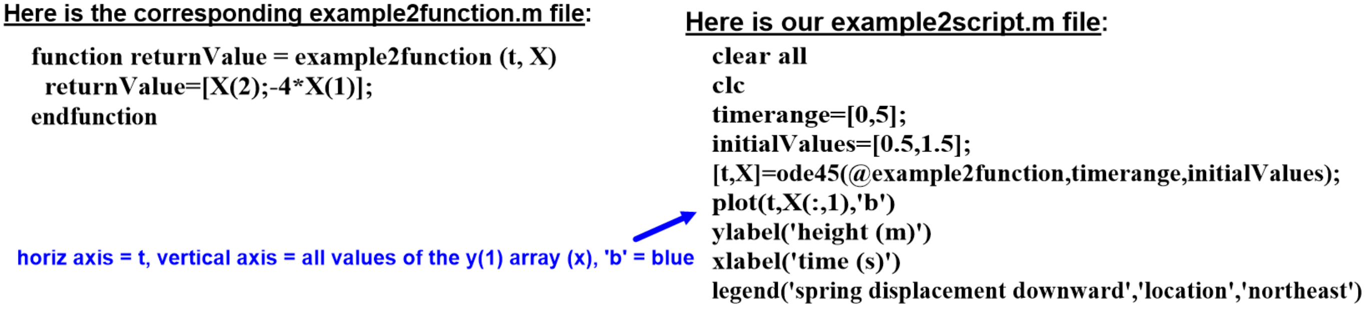
$output\ x'' = -4 * X(1)$

$returnValue = [an\ expression\ for\ x'; an\ expression\ for\ x'']$

$returnValue = [X(2); -4 * X(1)]$

Step 6c) Save new copies of the test files named ‘example2function.m’ and ‘example2script.m’ and change the function names to match. (Repeat what you did in Step 3a to save new copies of the function and script files to edit).

Step 6d) Change our example2function.m and example2script.m files to match this:



Here is the corresponding example2function.m file:

```
function returnValue = example2function (t, X)
  returnValue=[X(2);-4*X(1)];
endfunction
```

Here is our example2script.m file:

```
clear all
clc
timerange=[0,5];
initialValues=[0.5,1.5];
[t,X]=ode45(@example2function,timerange,initialValues);
plot(t,X(:,1),'b')
ylabel('height (m)')
xlabel('time (s)')
legend('spring displacement downward','location','northeast')
```

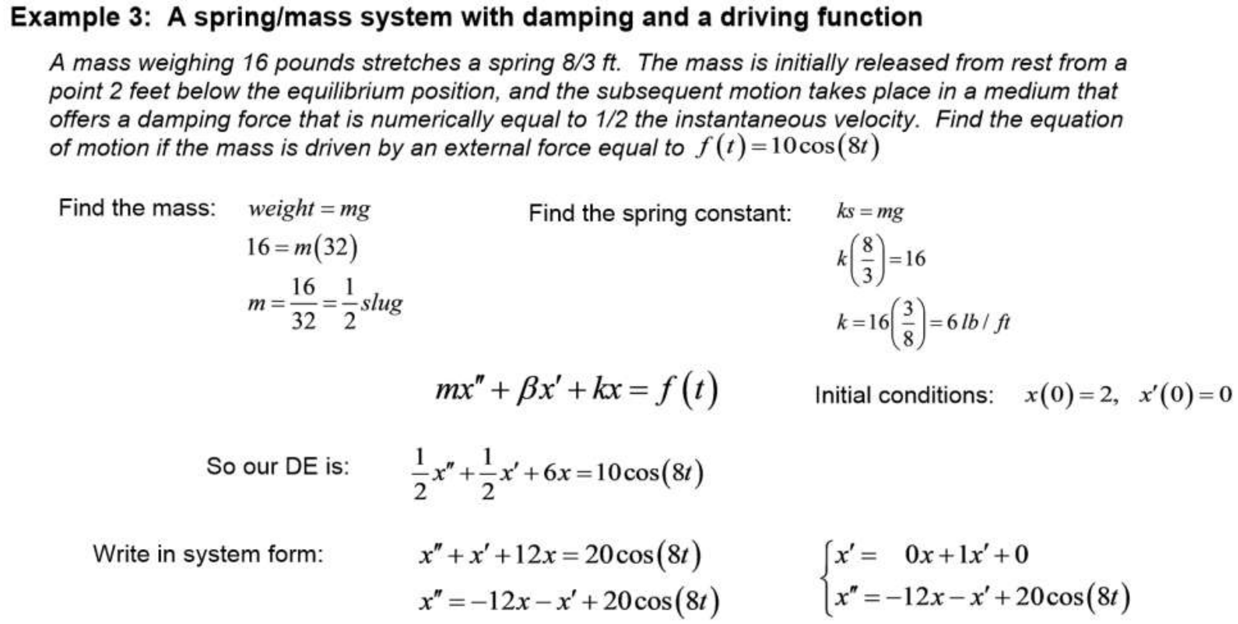
horiz axis = t, vertical axis = all values of the y(1) array (x), 'b' = blue

Try to figure out what each of these lines in the two files is doing and why they are written this way ☺

Step 6e) Save and Run these files (gear button). If needed debug your program. Once it is running successfully, copy the solution curve graph below:

Step 7) Example #3: Now use OCTAVE to change things in Example 2 and quickly see effects

One great thing about OCTAVE is that it doesn't take much effort at all to add in things like a damping coefficient or a right-hand side driving function and quickly see the effects on the solution curve…



Example 3: A spring/mass system with damping and a driving function

A mass weighing 16 pounds stretches a spring 8/3 ft. The mass is initially released from rest from a point 2 feet below the equilibrium position, and the subsequent motion takes place in a medium that offers a damping force that is numerically equal to 1/2 the instantaneous velocity. Find the equation of motion if the mass is driven by an external force equal to $f(t)=10\cos(8t)$

Find the mass:

$$weight = mg$$
$$16 = m(32)$$
$$m=\frac{16}{32}=\frac{1}{2}\,slug$$

Find the spring constant:

$$ks = mg$$
$$k\left(\frac{8}{3}\right)=16$$
$$k=16\left(\frac{3}{8}\right)=6\,lb/ft$$

$$mx''+\beta x'+kx=f(t)$$

Initial conditions: $x(0)=2,\quad x'(0)=0$

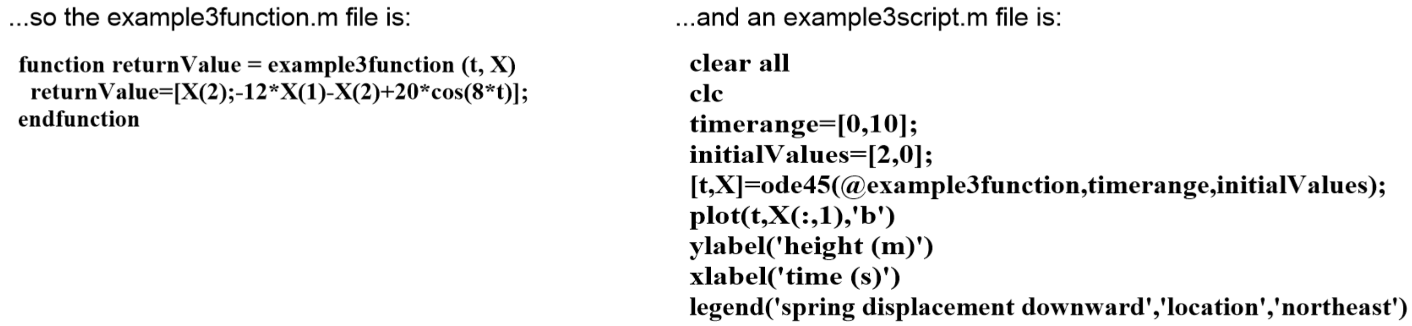
So our DE is: $\frac{1}{2}x''+\frac{1}{2}x'+6x=10\cos(8t)$

Write in system form:

$$x''+x'+12x=20\cos(8t)$$
$$x''=-12x-x'+20\cos(8t)$$

$$\begin{cases} x' = \;\; 0x+1x'+0 \\ x''=-12x-x'+20\cos(8t)\end{cases}$$

<u>Step 7a</u>) Make new copies of the function and script files and update as follows:



...so the example3function.m file is:

```
function returnValue = example3function (t, X)
  returnValue=[X(2);-12*X(1)-X(2)+20*cos(8*t)];
endfunction
```

...and an example3script.m file is:

```
clear all
clc
timerange=[0,10];
initialValues=[2,0];
[t,X]=ode45(@example3function,timerange,initialValues);
plot(t,X(:,1),'b')
ylabel('height (m)')
xlabel('time (s)')
legend('spring displacement downward','location','northeast')
```

<u>Step 7b</u>) Sketch your solution curve:

Your solution curve for Step 7b should be complicated at first because there is a mix of variation due to the natural resonance of the system along with the driving function, but the natural resonance eventually dies out, leaving only the driving function's effect.

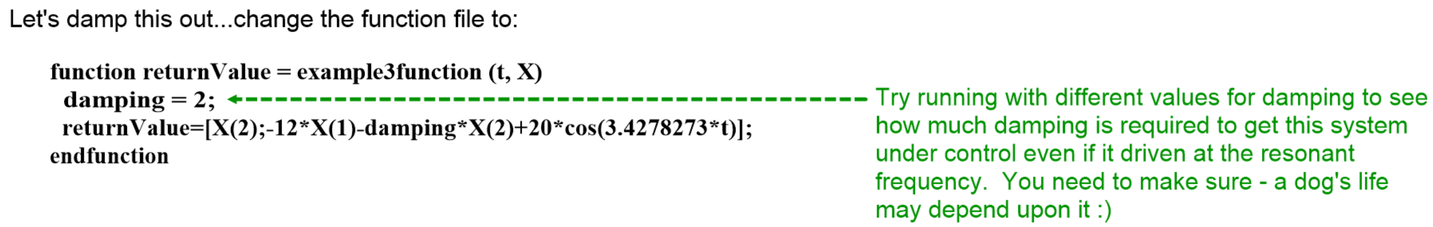
Now let's play a bit…

Step 7c) What happens if we remove the driving function and just let the mass oscillate at it natural resonant frequency once perturbed? Figure out what changes you need to make to the function file (you can keep reusing the same example3function.m and example3script.m files) to model this and sketch your new solution curve:

Step 7d) Now take out the damping entirely but add the driving function back in. Sketch the resulting solution curve:

Step 7e) The natural resonant angular frequency of this system is 3.4278273. Try changing the driving function to cos(3.4278273t) with no damping and sketch the resulting solution curve:

This solution curve for Step 7e should be an oscillation which is growing without bound – destructive resonance!



Let's damp this out...change the function file to:

```
function returnValue = example3function (t, X)
  damping = 2;
  returnValue=[X(2);-12*X(1)-damping*X(2)+20*cos(3.4278273*t)];
endfunction
```

Try running with different values for damping to see how much damping is required to get this system under control even if it driven at the resonant frequency. You need to make sure - a dog's life may depend upon it :)

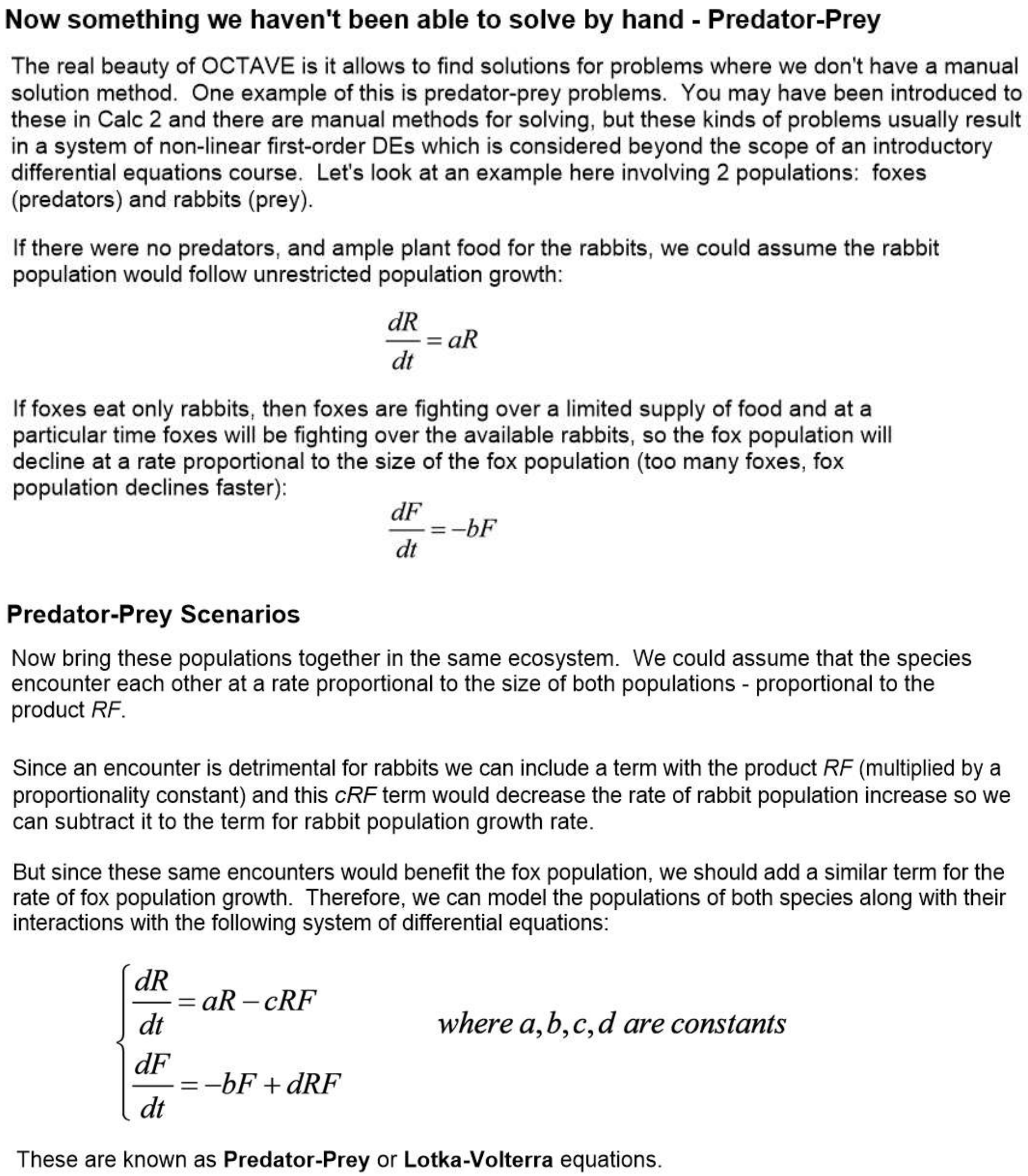
Step 7f) How much damping do you believe we need (there is no 'correct' answer, just how high do you think we should go until the oscillations are under control?

damping =

Can you have too much damping? Yes, it might be very expensive or maybe impossible physically to get large amounts of damping. In the case of a bridge or the floor of a building you really don't want oscillations, so the less movement, the better. But adding damping to a physical system may increase the cost or the weight, so there is a limit.

There are other kinds of systems which need control too. Let's say you are designing the control system for the cruise control for an automobile's speed. The driving function could represent the new 'set speed' which you want the speed to match. If you have huge amounts of damping in the system, then it could take so long for the car to reach the equilibrium speed that the cruise control is useless. On the other hand, you need some damping or the speed could oscillate around the desired speed, with wider and wider variations, causing loss of control of the vehicle.

Step 8) Example #4: A Predator-Prey System



Now something we haven't been able to solve by hand - Predator-Prey

The real beauty of OCTAVE is it allows to find solutions for problems where we don't have a manual solution method. One example of this is predator-prey problems. You may have been introduced to these in Calc 2 and there are manual methods for solving, but these kinds of problems usually result in a system of non-linear first-order DEs which is considered beyond the scope of an introductory differential equations course. Let's look at an example here involving 2 populations: foxes (predators) and rabbits (prey).

If there were no predators, and ample plant food for the rabbits, we could assume the rabbit population would follow unrestricted population growth:

$$\frac{dR}{dt} = aR$$

If foxes eat only rabbits, then foxes are fighting over a limited supply of food and at a particular time foxes will be fighting over the available rabbits, so the fox population will decline at a rate proportional to the size of the fox population (too many foxes, fox population declines faster):

$$\frac{dF}{dt} = -bF$$

Predator-Prey Scenarios

Now bring these populations together in the same ecosystem. We could assume that the species encounter each other at a rate proportional to the size of both populations - proportional to the product *RF*.

Since an encounter is detrimental for rabbits we can include a term with the product *RF* (multiplied by a proportionality constant) and this *cRF* term would decrease the rate of rabbit population increase so we can subtract it to the term for rabbit population growth rate.

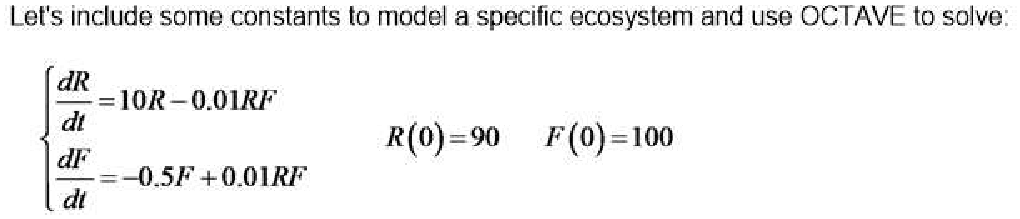
But since these same encounters would benefit the fox population, we should add a similar term for the rate of fox population growth. Therefore, we can model the populations of both species along with their interactions with the following system of differential equations:

$$\begin{cases} \dfrac{dR}{dt} = aR - cRF \\ \dfrac{dF}{dt} = -bF + dRF \end{cases} \qquad where\ a, b, c, d\ are\ constants$$

These are known as **Predator-Prey** or **Lotka-Volterra** equations.

<u>Step 8a</u>) Make new copies of the function and script file for example 4.

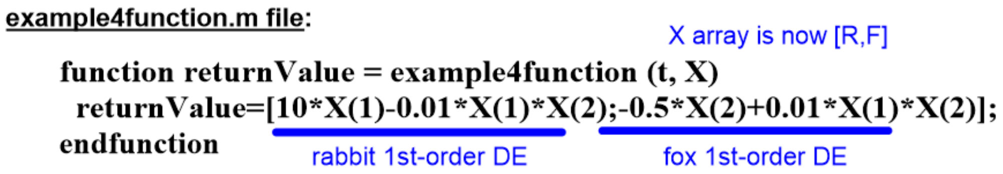
<u>Step 8b</u>) Select some values of the constants *a, b, c,* and *d* to model a particular ecosystem (in reality, if you were a biological researcher you could postulate initial values of these constants, run the simulation, and compare the resulting solution to the real world, adjusting these constants until they produced a model which matched the real-world data).



Let's include some constants to model a specific ecosystem and use OCTAVE to solve:

$$\begin{cases} \dfrac{dR}{dt} = 10R - 0.01RF \\ \dfrac{dF}{dt} = -0.5F + 0.01RF \end{cases} \qquad R(0) = 90 \qquad F(0) = 100$$

Step 8c) Update the example4function.m file:



example4function.m file:

X array is now [R,F]

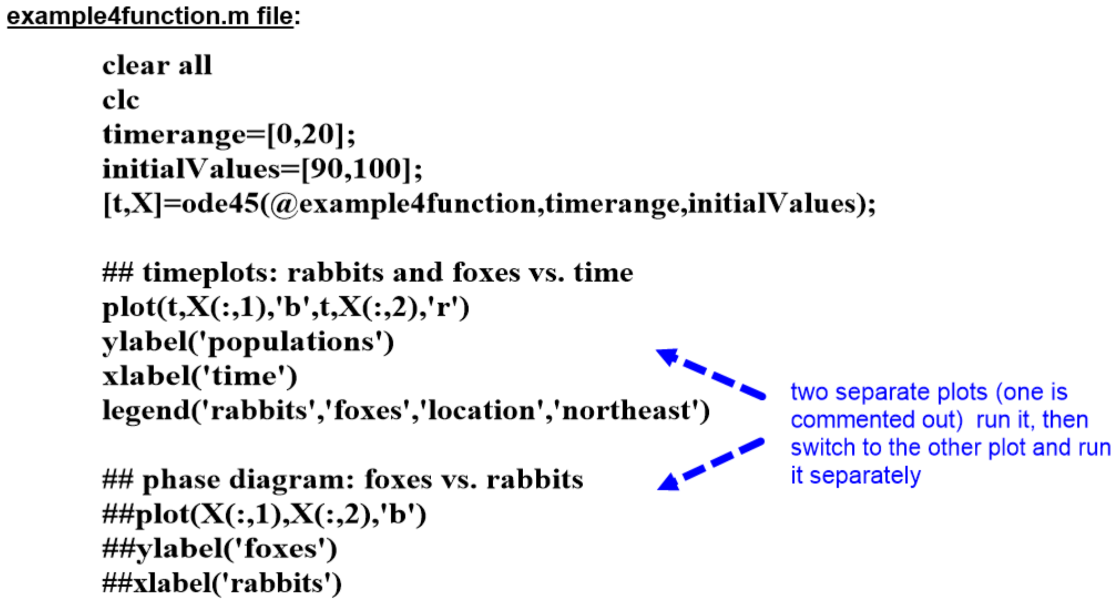
```
function returnValue = example4function (t, X)
  returnValue=[10*X(1)-0.01*X(1)*X(2);-0.5*X(2)+0.01*X(1)*X(2)];
endfunction
```

rabbit 1st-order DE

fox 1st-order DE

This is now a true system of two different differential equations (not a system which is representing a single higher-order differential equation), so in the return array, each of components is the full expression of the 1st-order differential equation for the rabbits and the foxes. The x array now has the two different independent values for the amount of rabbits and foxes, so x(1) is the first component of the x array which represents *R* (the number of rabbits) and x(2) is the second component of the x array which represents *F* (the number of foxes).

Step 8d) Update the example4script.m file:



example4function.m file:

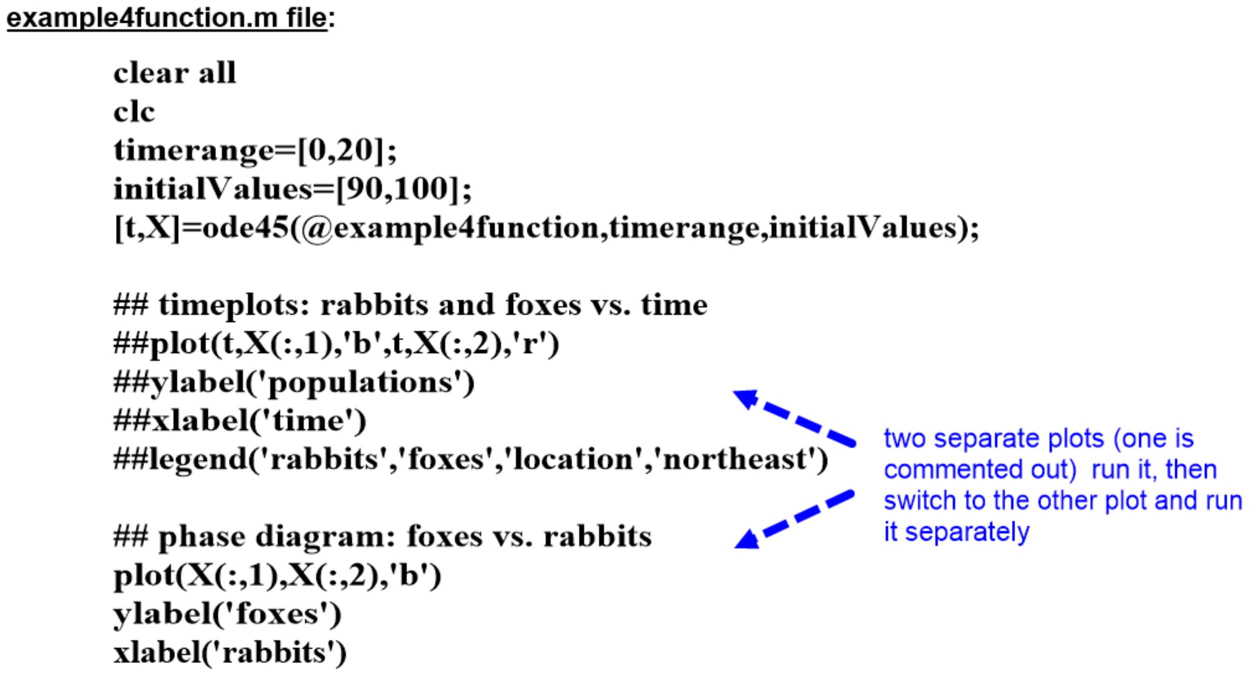
```
clear all
clc
timerange=[0,20];
initialValues=[90,100];
[t,X]=ode45(@example4function,timerange,initialValues);

## timeplots: rabbits and foxes vs. time
plot(t,X(:,1),'b',t,X(:,2),'r')
ylabel('populations')
xlabel('time')
legend('rabbits','foxes','location','northeast')

## phase diagram: foxes vs. rabbits
##plot(X(:,1),X(:,2),'b')
##ylabel('foxes')
##xlabel('rabbits')
```

two separate plots (one is commented out) run it, then switch to the other plot and run it separately

Step 8e) Sketch your population timeplot:

This plot of populations vs. time should show that the rabbit and fox populations are increasing, then decreasing in a cyclical way.

Step 8f) Another way to display the state of the ecosystem is with a Phase Diagram which plots the two dependent variables (foxes and rabbits) on the x and y axes. To change the plot to this, modify your example4script.m file as follows to comment out the plot you just produced, and enable a phase diagram plot:



example4function.m file:

```
clear all
clc
timerange=[0,20];
initialValues=[90,100];
[t,X]=ode45(@example4function,timerange,initialValues);

## timeplots: rabbits and foxes vs. time
##plot(t,X(:,1),'b',t,X(:,2),'r')
##ylabel('populations')
##xlabel('time')
##legend('rabbits','foxes','location','northeast')

## phase diagram: foxes vs. rabbits
plot(X(:,1),X(:,2),'b')
ylabel('foxes')
xlabel('rabbits')
```

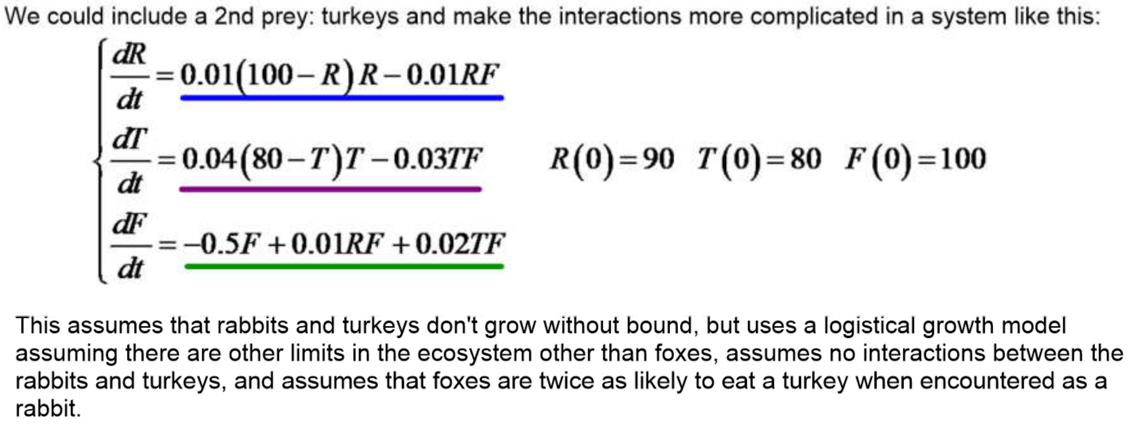
two separate plots (one is commented out) run it, then switch to the other plot and run it separately

Step 8g) Run OCTAVE and sketch the resulting 2D Phase Diagram for the rabbits/foxes ecosystem:

You should see the phase state oscillating, but never reaching a point of stability.

Step 8h) We could make the model more realistic by including two prey species and we could use logistic models for growth that account for food limitations in the environment for the prey…

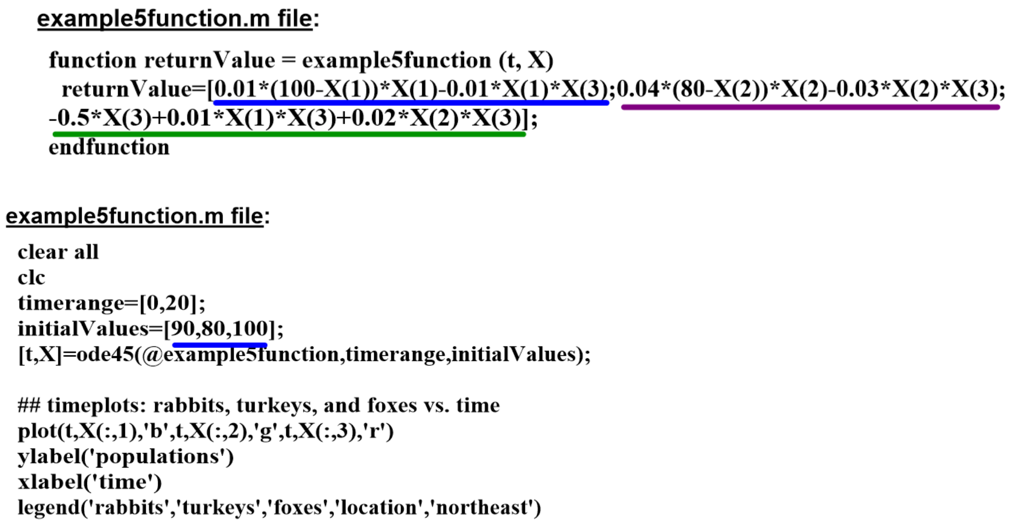
- Read through the following scenario: With 3 species and time, we could now make plots for species vs. time (timeplot) and 3 different 2D phase diagrams comparing species populations, and we can even use the 'plot3' function to have OCTAVE produce a 3D phase diagram.
- Save new copies of your function and script files (example 5) and see if you can make OCTAVE produce the following plots successfully:



We could include a 2nd prey: turkeys and make the interactions more complicated in a system like this:

$$\begin{cases} \dfrac{dR}{dt} = \underline{0.01(100-R)R - 0.01RF} \\ \dfrac{dT}{dt} = \underline{0.04(80-T)T - 0.03TF} \qquad R(0)=90 \quad T(0)=80 \quad F(0)=100 \\ \dfrac{dF}{dt} = \underline{-0.5F + 0.01RF + 0.02TF} \end{cases}$$

This assumes that rabbits and turkeys don't grow without bound, but uses a logistical growth model assuming there are other limits in the ecosystem other than foxes, assumes no interactions between the rabbits and turkeys, and assumes that foxes are twice as likely to eat a turkey when encountered as a rabbit.



example5function.m file:

```
function returnValue = example5function (t, X)
  returnValue=[0.01*(100-X(1))*X(1)-0.01*X(1)*X(3);0.04*(80-X(2))*X(2)-0.03*X(2)*X(3);
-0.5*X(3)+0.01*X(1)*X(3)+0.02*X(2)*X(3)];
endfunction
```

example5function.m file:

```
clear all
clc
timerange=[0,20];
initialValues=[90,80,100];
[t,X]=ode45(@example5function,timerange,initialValues);

## timeplots: rabbits, turkeys, and foxes vs. time
plot(t,X(:,1),'b',t,X(:,2),'g',t,X(:,3),'r')
ylabel('populations')
xlabel('time')
legend('rabbits','turkeys','foxes','location','northeast')
```

Step 8i) Run OCTAVE and sketch the resulting population timeplots for the ecosystem:

Step 8j) Modify the script file to plot a 3D Phase Diagram showing the interaction of all the species:

example5function.m file:

```
clear all
clc
timerange=[0,20];
initialValues=[90,80,100];
[t,X]=ode45(@example5function,timerange,initialValues);

## 3D phase diagram: foxes vs turkeys vs rabbits
plot3(X(:,1),X(:,2),X(:,3))
ylabel('turkeys')
xlabel('rabbits')
zlabel('foxes')
```

Step 8k) Run OCTAVE and sketch the resulting 3D Phase Diagram for the 3 species ecosystem:

ASSESSMENT ---

Now it's time to see if you can put all this together to solve a problem on your own. Use what you've learned to use OCTAVE to graph the solution curve for the following problem:

A 3-kg object is attached to a spring and hangs down, stretching the spring 0.392 meters. Assume there is a damping force with $\beta = 2$, and there is a forcing function of the form: $f(t) = 30\cos(3t)$. Initial conditions: $x(0) = 0.2,\ x'(0) = -0.1$

The differential equation which models this is: $x'' + \frac{2}{3}x' + 25x = 10\cos(3t)$

ASSESSMENT #1) Write the lines in your function.m file:

ASSESSMENT #2) Write the lines in your script.m file:

ASSESSMENT #3) Sketch the OCTAVE output solution curve:

Done!

If you’ve finished with time to spare, please ask for the follow-on ‘Murder at the Mayfair’ extra credit problem ☺