Lab 1: MATLAB and Numerical ODE Solving

Laboratory Outline:

-MATLAB Refresher

-Write your own differential equation solver

-Solve given equation analytically and with solver

-Use built-in MATLAB solver to do the same

1. For the majority of you, it has been nearly six months since you last used MATLAB. The first part of this lab is a quick refresher of essential MATLAB commands and syntax. We will do this as a group. Please include your code and plot for this section.
    a. Given the function :

    $$y(t) = e^{-at}\cos(bt)$$

    Plot it between t=0 and t=2 at 0.01 increments for a=2 and b=10.

    Code help with syntax required for the above assignment:

    Creating variables and writing simple equations

    ```
    var = [number]; ←assigns a number to a letter variable
    var = [first number]:[increment]:[last number] ←creates an array
                                of evenly spaced numbers between a min and max
    ```

    Mathematical Operators

    ```
    [array].*[number] ←modifies each value in an array
    exp([number]) ←e^[number]
    ```

    Simple plots and axes labeling

    ```
    figure([figure number]) ←initializes a new figure
    plot([t1 vector],[y1 vector],[t2 vector],[y2 vector]) ←plot
                                multiple sets of variables on the same graph
    ```

    alternatively:

    ```
    plot([t1 vector],[y1 vector])
    hold on ←allows plotting to the same graph
    plot([t2 vector],[y2 vector])
    xlabel('[x variable name]');
    ylabel('[y variable name]');
    title('[plot title]');
    ```

b. **Function writing, calling m-files, good programming style**

We ask that you use one of two methods to structure your m-files:

One method for m-file writing is to have a main m-file that is not a function, but simply a script that contains code. You run your script by writing the m-file name (minus the .m of course) on the command line. In this case, if you want to define your own functions and call them in this main code, they must be provided in a *separate* m-file in which the name of the function is the same as that of the file.

The other method is to have your main code itself be a function. You still run your program by writing the function name (which is the same as the m-file name) on the command. The first line of the code must be "`function lab1`" (or whatever the m-file name is), followed by the rest of the commands. In this method, you *can* include your other functions in the *same* m-file following the first one, in any order (or you can place them in their own files). If your first function is ended with a line that says "end", all must be. If the first is simply ended by defining another function, no functions should be ended with "end". To run the entire script, merely type the name of the first function in the command window.

We also recommend the following good coding practices, which become essential as you write more complex code and as you create code that will be used in multiple sessions.

- Comment your code
- Start with the command "clear all" at the beginning of your main code or anywhere you are basically starting over. This clears all variables so that you have to be defining them each time and helps immensely with debugging. It prevents the situation where you forgot to define a variable within the code, the code runs because you happened to define this variable earlier in this session, and the result is not predicted by the code itself but rather by something else that you may have forgotten you did or you may forget to repeat.
- Any time you plot, assign a figure number, so you know what figure is plotting which data. Use the command "close all" at the start of your main code, to close all windows so you know that everything in your figure is from this run not some previous and now forgotten step.

c. **Difference between description and examples in documentation**

Go to "Help>Product Help" and search for "ode45"

The search results give you, for most functions and items, "Syntax", "Arguments", "Descriptions", "Options", and "Examples"

Syntax is useful for quick reference to how to call the function, but the examples show exactly how to complete an operation.

2. Next, we will learn how to write a simple code that will solve a differential equation. Consider a differential equation in one variable. This will have the general form $dy(t)/dt = f(y(t))$. If you are at position y_1 at time t_1, solving a differential equation numerically means you have to calculate your position y_2 at time t_2, where $t_2 = t_1+\Delta t$; that is, t_2 is a small increment in time forward from t_1.

   a. Remember from calculus that the definition of a derivative is:

   $dy/dt = \lim_{\Delta t \to 0} \frac{y(t+\Delta t) - y(t)}{\Delta t}$. Thus, a "first-order" estimation of the derivative is

   $dy/dt \approx \frac{y(t+\Delta t) - y(t)}{\Delta t}$, with the estimation becoming for more accurate as Δt approaches zero. Rearrange this to give a first-order estimation of $y(t+\Delta t)$ given $y(t)$. Now rewrite this estimation using the expression of the derivative presented in the differential equation: $dy/dt = f(y(t))$. It may help you to sketch a graph of the problem with some hypothetical or real function.

   b. Now write a MATLAB script that solves for y_n iteratively (using a loop of your choosing) for the following differential equation:

   $$dp_1(t)/dt = k_{01} - p_1(t)(k_{01} + k_{10})$$

   with $k_{01} = 50$ sec^{-1}, $k_{10} = 2$ sec^{-1}, and the initial value of $p_1 = 0$. Solve for the time period from 0 to 1 second with a time step $\Delta t = 0.001$ seconds. This is called numerical integration of a differential equation. As described below, this equation models the probability of a channel being open as a function of time, where the channel is originally closed but the conditions change at time 0. Plot the proportion of channels open (p_1) against time. Remember to label your axes.

   The following is meant to explain how the equation we are using is derived, model-building will not be covered until next lab. Feel free to skip to section c below.

The physiological event you are modeling is the opening of ion channels due to depolarization, which directly affects the rate at which channels open. Because the rate of channel opening is so much greater than closing, nearly all of the channels open in very little time after depolarization.

$dp_1(t)/dt = -p_1(t)k_{10} + p_0(t)k_{01}$ ←change is number turning on minus turning off

$dp_1(t)/dt = -p_1(t)k_{10} + (1-p_1(t))k_{01}$ ←proportion off is one minus proportion on

$dp_1(t)/dt = k_{01} - p_1(t)(k_{10} + k_{01})$

c. <u>You should quickly be able to solve this particular differential equation analytically since you've taken differential equations.</u> (What is the form of a first-order linear differential equation in one variable? Ie, if dy/dt = -kt, what is y(t)? How does the constant term affect this?). <u>Plot the analytic together with your numerical solution.</u> Numerical integration should yield a close but not exact solution. If your solution is way off, this either means you made an error in your code or that the time step is too large for the estimation to be accurate. Make sure you are getting close to the analytic solution, and include this plot in your lab write up.

d. Now consider the situation where the voltage changes part-way through the experiment. After 0.3 seconds, the parameters change from the above values to $k_{01} = 0.0001$ sec^{-1}, and $k_{10} = 20$ sec^{-1}. <u>Plot p_1 against time for 1 second.</u>

3. <u>Now use one of the MATLAB ode solvers to produce the plots again for parts (b) and (d) above.</u> (Use the documentation mentioned previously to know how they work). There are many different solvers, each using different algorithms and having different advantages and disadvantages. However, they all use essentially the same syntax.

Some hints on how to call them:

a. You will need to call a function to use their ode solvers. To do this, you will have to make your main file also be a function file. The first line should thus be: "function mfilename( )" . Then you will call this from the MATLAB command window with "mfilename". If you want to pass variables into your function, you will include variables within the parentheses and write in the command line: "mfilename(variable1, variable2)". You can include the function(s) you are calling within the same main file, using "end" at the end of each function and "function…" at the start of the next. Or, you can make each function be its own file.

b. You can re-define variables within the ODE function or declare variables to be global. But it is more elegant to pass variables into the ODE function: After the "options" in the input list of ode23 (or whichever ode solver you are using) you can pass variables: [T,Y] = ode23(@ODEname, timeframe, IC, options, variable1, variable2). You can then include the variables in the 'ODEname' function in this way: 'function dY = ODEname(t,Y,variable1, variable2)'

4. Given the following second order differential equation:

a * x''(t) + b * x'(t) + c * x(t) = 0

where x' and x'' are the first and derivatives of x, resp.

This is the generic (dimensionless) equation for a harmonic oscillator (e.g. pendulum, spring, RLC circuits)

a) Rewrite the above equation as a system of ODEs (no second derivatives) by introducing a new variable, **y(t)**. What is the differential equation for **y(t)** and the new differential equation for **x(t)**?

b) Use a = 1, b = 0.1, c = 1, and x(0) = y(0) = 1 Use one of the MATLAB ode solvers to generate a graph showing x(t) and y(t) from time = 0 to time = 50

c) Use the **hold on** command to plot multiple time series of x(t) showing what happens as you change the parameter **b**. Change **b** from 0.1 to 1 in increments of 0.1, and plot each of the time series plots on the same graph. Give a sentence description of what the parameter **b** doing?