MATH 442: Mathematical Modeling

Lecturer: Dr. Jean Marie Linhart
`http://www.math.tamu.edu/~jmlinhart/m442`

Assignment 2 – due Thursday 1/31/2013

You are going upload a zip file containing a MATLAB script file and a MATLAB function files as described below.

If you are finding this difficult, either start looking at the getting started with MATLAB file linked with this homework, look at the MATLAB resources I have on the web, go to the library and find a MATLAB reference to help (or go buy one).

Try to develop some **knowledge** and **autonomy** so that MATLAB is **your tool to use**.

You can also ask your classmates, the TAs and me for help. That said, don't copy, and if you receive extensive help from someone, you should probably acknowledge it in the comments of your code.

1. Type the following code into MATLAB (verbatim).

```
clear all; %clears Matlab's memory.
close all; %closes all of the currently open figures.
t = linspace(0,3*pi, 100);
x = cos(t);
y = sin(t);
z = zeros(size(t));
figure(1)
plot(t, x, 'k--', 'LineWidth', 2)
hold on;
plot(t, y, 'k', 'LineWidth', 2)
plot(t, z, ':k')
title('Sine and Cosine','FontSize', 24)
xlabel('x', 'FontSize', 20)
ylabel('y', 'FontSize', 20)
legend('y=cos(x)', 'y=sin(x)')
xlim([0 3*pi])
```

The % character is a comment character. I made comments after the first two lines telling you what they did.

You are to provide brief comments for the remaining lines explaining what they do. You can, for example, comment out an entire line of code by starting it with %. You can figure out what the `xlim([0 3*pi])` command does by comparing the graphs with the command commented out or included.

2. Make t go from 0 to 4π with 150 steps in between. Open figure 2 and make a graph of the functions $t\sin(t)$ and $t\cos(t)$ overlaid on the same set of axes using LineWidth 2. Use the `axis` command to make the t axis go from $[0, 4\pi]$, and the y axis from $[-4\pi, 4\pi]$. The title of your graph should be "tsin(t) and tcos(t) from 0 to 4\pi" with FontSize 24. The xlabel is "t", the ylabel is "tsin(t) and tcos(t)", and have the labels display with FontSize 18. Include a legend with labels 'tsin(t)' and 'tcos(t)'. (Hint: You need `.*` here.)

3. Open figure 3 you will graph $t\sin(t)$ and $t\cos(t)$ as above, but you will stack the two plots instead of overlaying them. You will use the command `subplot()`, which takes three arguments, the number of rows, the number of columns and which plot you are on. `subplot(2,1,1)` says that on this figure I will have 2 rows, 1 column, and the plot to follow is the first one. Since the two graphs are not overlaid, you do not need a legend, but modify titles, axis labels, etc. appropriately.

4. The Bim-Bam challenge. Use a `for` loop and `if` statements to have MATLAB count from 1 to 100, replacing multiples of 3 with Bim and multiples of 5 with Bam. If the number is a multiple of both 3 and 5, replace it with BimBam. Otherwise output the digit in question. The `mod()` function (modular arithmetic) might be helpful here. I created a script file called BimBam for my code, and I used `fprintf('Bim,')` and similar commands to get the output you see below.

```
>> BimBam
1, 2, Bim, 4, Bam, Bim, 7, 8, Bim, Bam,
11, Bim, 13, 14, BimBam, 16,
>>
```

5. Now I want you to use MATLAB 's `ode45()` command, which is MATLAB's standard numerical code for solving a differential equation. It assumes the differential equation is written in the form

$$\mathbf{x}' = \mathbf{f}(t, \mathbf{x})$$

where $\mathbf{x}$ and $\mathbf{x}'$ are column vectors, and t represents the independent variable. We are going to solve the Lotka-Volterra predator-prey equations

$$\begin{aligned} \frac{dx}{dt} &= a_1 x - b_1 xy \\ \frac{dy}{dt} &= -a_2 y + b_2 xy \end{aligned}$$

The first equation is for the prey, the second equation is for the predators. These are autonomous ODEs, meaning there is no t dependence. Take a look at Section 1.4 in *A Concrete Approach to Mathematical Modeling* for more information; we are going to reproduce the graphs on page 10.

Here is the syntax for `ode45()`: `ode45(`*odefun, tspan, y0, options*`)`. The first argument, *odefun* is the differential equation function, which must be in the form $\mathbf{x}' = \mathbf{f}(t, \mathbf{x})$. The second argument *tspan* gives the time span. If you give the time span as two numbers indicating the beginning time and ending time, MATLAB will determine where to evaluate the $\mathbf{x}(t)$ in between these. You can also give a vector of numbers, and in this case MATLAB will evaluate $\mathbf{x}(t)$ where you asked it to. The third argument *y0* is the initial condition; if the first time value you give is *t0* then $\mathbf{x}(t_0) = y_0$. The last argument, *options* allows you to customize how `ode45` works. You can, for example, change the tolerance or error criterion that `ode45` uses to tell when the solution is good enough.

First write the differential equation. Open a new function file. Give your function a descriptive (but not too long) name that tells you what it is; I used `LotVolt`. To fit the general ODE form, your function will accept as input variables `t, x` and the output variable will be `xp`, which stands for x-prime ($\mathbf{x}'$). Program in the Lotka-Volterra ODEs with constants $a_1 = 3$, $a_2 = 5/2$, $b_1 = 2$, $b_2 = 1$. Save your file; the file name must be of the form `filename.m` where **the file name agrees with your function name**. My file as saved as `LotVolt.m`.

Now, assuming your function is named `LotVolt`, our time span is from $t = 0$ to $t = 6$, and we have initial conditions $x(0) = 1$ and $y(0) = 1$ you can invoke `ode45` with the following command

```
[t x] = ode45(@LotVolt, [0 6], [1 1]);
```

In `t` we will have a (column) vector of time values. In `x` we will have an array with 2 columns, the first column is $x(t)$ the prey population over time, the second is $y(t)$, the predator population over time.

In figure 4 and figure 5 create reasonable facsimiles of figure 1.2 and figure 1.3 on page 10 of *A Concrete Approach to Mathematical Modeling*. Label your graphs well.

6. **Numerical methods for ordinary differential equations.** We are going to program Euler's Method for solving an ODE.

   You know from the limit definition of the derivative that

$$\mathbf{x}'(t) = \lim_{h \to 0} \frac{\mathbf{x}(t+h) - \mathbf{x}(t)}{h}$$

   Assuming h is small,

$$\mathbf{x}'(t) \approx \frac{\mathbf{x}(t+h) - \mathbf{x}(t)}{h}$$

should be a good approximation. Rearrange to get

$$\mathbf{x}(t+h) \approx \mathbf{x}(t) + h\mathbf{x}'(t)$$

If ODE is in the form

$$\mathbf{x}' = \mathbf{f}(t,\mathbf{x}), \quad \mathbf{x}(t_0) = \mathbf{x}_0$$

we get

$$\mathbf{x}(t+h) = \mathbf{x}(t) + h\mathbf{f}(t,\mathbf{x}(t))$$

This gives us Euler's method

$$\mathbf{x}(t_{n+1}) = \mathbf{x}(t_n) + h\mathbf{f}(t_n,\mathbf{x}(t_n))$$

Open a new function file in Matlab, and save it as `euler.m`. Type the following code in exactly as written

```
function [t,y] = euler(func, T, y0)
% explicit Euler method for scalar ODEs of the form y'=func(t,y)
% Arguments:
% func(t,y) is the ODE which returns a *column* vector
% T is a vector with time steps
% y0 the initial condition at T(1) (row vector)


% N gives number of time steps
N=length(T);
M = length(y0);
%create a vectors for t
t=zeros(N,1);
%create a matrix for y since y0 may be a vector.
y=zeros(N,length(y0));

%first row is our initial condition
t(1)=T(1);
y(1,:)=y0;

for n=1:N-1
    t(n+1)=T(n+1);
    h = T(n+1) - T(n);
    %func returns a column vector, whereas y(n,:) is a row vector
    %the ' operation transposes a matrix.
    y(n+1,:)= y(n,:) + h*func(t(n),y(n,:))';
end

end
```

We will use this code to solve the Lotka-Volterra equations as in the previous problem.

Unlike `ode45()`, for our Euler code, we have to supply a complete vector of time values. Use

```
tspan = 0:0.05:6;
[t x1] = euler(@LotVolt, tspan, [1 1]);
[t x2] = ode45(@LotVolt, tspan, [1 1]);
```

In figure 6 you will use `subplot` to get 2 plots stacked vertically. For the top plot, plot the the prey population from the Euler code versus time, and overlay a plot of the prey population from `ode45()` versus time. For the bottom plot, do the same with the predator populations. Include appropriate titles, axis labels, and a legend clearly identifying which curves are from `ode45()` and which are from the Euler method.

7. Now that you've successfully programmed the Euler method, you will program the Runge-Kutta order 4 method, which is a lot more accurate than Euler. Runge-Kutta order 4 is derived from the Taylor series for $\mathbf{x}(t)$; here is the algorithm:

$$\begin{aligned}
h &= t_{n+1} - t_n \\
\mathbf{k}_1 &= h\mathbf{f}(t_n, \mathbf{y}_n) \\
\mathbf{k}_2 &= h\mathbf{f}\left(t_n + \frac{1}{2}h, \mathbf{y}_n + \frac{1}{2}\mathbf{k}_1\right) \\
\mathbf{k}_3 &= h\mathbf{f}\left(t_n + \frac{1}{2}h, \mathbf{y}_n + \frac{1}{2}\mathbf{k}_2\right) \\
\mathbf{k}_4 &= h\mathbf{f}\left(t_n + h, \mathbf{y}_n + \mathbf{k}_3\right) \\
\mathbf{y}_{n+1} &= \mathbf{y}_n + \frac{\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4}{6}
\end{aligned}$$

Rather than starting from scratch, open up your Euler method code, and **IMMEDIATELY** save it with file name `rk4.m`. *Do not inadvertently write over the Euler code you just created.* Rename the function to `rk4`. Add and modify what you must to get the above algorithm working. Don't forget to change the comments!

Once you have it working, use it to solve the Lotka-Volterra equations as you did with Euler above. Use `t = 0:0.05:6` again, and initial condition $x(0) = 1$, $y(0) = 1$.

In figure 7, you will use `subplot` to get 2 plots stacked vertically. The top will be the prey populations versus time from the Euler code, the `rk4` code, and the `ode45()`. The lower will be the predator populations from the three ODE solvers. Include appropriate titles, axis labels, and a legend clearly identifying which curves are from `rk4`, which are from `ode45()` and which are from `euler`.

The curve from `rk4()` should line up on top of the curve for `ode45()`. `ode45()` is an improvement on `rk4()`; both are much more accurate than Euler.