Math 308 - Sections 503/504 - Project 1

Instructor: Dr. Adam Larios

Due date: Friday, 12 October 2012, 4:30 pm in Blocker 601
First draft due: Friday, 28 September 2012, Electronically

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. If you submit your project as a group, submit it with all group member's names clearly labeled on the front. 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[1], along with any relevant graphs, mathematical calculations, and explanations.

Your first draft is due in **electronic form only**. Matlab files and PDF files are allowable, but **Microsoft Word files will not be accepted**. If you use Word (or a similar program), be sure to print your file, and select "print to pdf" or "print to file" so that you are using a universal format which looks the same on different computers. For your final draft, please turn in a single **paper copy**.

Make sure any graphs are well labeled and well referenced, and that they are easy to read. Do not just print out graphs and code and expect me to be able to follow it. You need to explain what you are doing at each step. Someone who is familiar with differential equations, but who does not know about this project, should be able to read your report and understand it completely.

By returning 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.

[1]There is no need to submit code which was *not* written by your group.

Warm Up

Warm up part 0: Try some of the exercises in "Matlab Introduction" on the main class webpage.

Warm up part 1: Did you do part 0? If not, go back and do it. OK, now get started by trying the Matlab code below. It's easier to type this into an *.m file rather than to enter them one by one in the command line.

Pro tip: Typing code by hand will help you learn it much better than copy/paste.

```
num_points = 4;
for i =1:6
    t = linspace(0,2*pi,num_points);
    x = cos(t);
    y = sin(t);

    subplot(2,3,i);
    plot(x,y);
    axis('square');
    title(sprintf('Here are %d points',num_points-1));

    num_points = num_points + 1;
end
```

Remember: If you don't know how to use a command, use `help`. You will need to scroll up to see what it spits out. Try these:

```
help plot
help for
help axis
help help
```

Part I: Making your own ODE solver

Now that you are getting familiar with the basics of Matlab, let's try something a little more useful. Let's solve first-order ODE's (Ordinary Differential Equations) with initial values:

$$\begin{cases} \dfrac{d}{dt}y = f(t,y), \\ \quad y(t_0) = y_0. \end{cases}$$

In this section, you will program two O.D.E. solvers and test their accuracy. The first one is already done for you below. **Type this method out yourself**, and see if you can understand each line. When you are ready, try to run the code. Notice that this code is a function, so you can change the inputs and outputs easily. Here, we take a time interval $t_0 \le t \le T$, where with think of t_0 as the initial time, and T as the final time. Let N be the number of points, and calculate the step size h by $h = (T - t_0)/N$.

```
%% FILE forward_euler.m %%

function [Y t]= forward_euler(f,t0,T,y0,N)
% Solve dy/dt = f(t,y), y(t0)=y0
% for t0 <= t <= T, with N time steps.
% Sample run:
% [Y t]= euler(@(t,y) sin(t*y), 0, 5, 0.2, 10);
% plot(t,Y,'-o');
close all;

h = (T - t0)/N; % Calulate and store the step-size
Y = zeros(1,N); % Initialize the Y vector.

t = linspace(t0,T,N); % A vector to store the time values.
Y(1) = y0; % Start Y at the initial value.

for i = 1:(N-1)
  Y(i+1) = Y(i) + h*f(t(i),Y(i)); % Update approximation Y at t+h
end

%%% END FILE %%%
```

CHECK: Are there any orange or red lines on your scroll bar? These are warnings and errors. Hover your mouse over them to see what they are. Clear them up before running your code. If you have a green box at the top, you are ready!

Save the above file as "forward_euler.m". It must have this exact name, and must be in the your current working directory for Matlab to be able to find it. Use the command `pwd` (print working directory) to check your current directory, `ls` (list) to list what's in it, and `cd <directory name>` to change directory.

We will now solve the IVP (initial value problem) given by:

$$\begin{cases} \dfrac{dy}{dt} = \sin(ty), \\ y(0) = 0.2. \end{cases}$$

Run the code by entering these commands in the terminal:

```
[Y t]= forward_euler(@(t,y) sin(t*y),0,5,0.2,10);
plot(t,Y,'-o');
```

Try playing with the inputs. Can you increase the number of points to get a smoother-looking graph? What happens when you change the initial value, or even the function?

The Runge-Kutta Method

The following numerical algorithm (written in "pseudo-code") uses the Runge-Kutta method to yield an approximation Y_i of y at $t_i = t_0 + (i-1)h$ for $i = 1, \ldots, N+1$.

$$
\begin{aligned}
&t \leftarrow t_0 \\
&y \leftarrow y_0 \\
&t_i \leftarrow t \\
&Y_1 \leftarrow y \\
&\text{FOR } i = 2, \ldots, N+1 \\
&\quad k_1 \leftarrow f(t, y) \\
&\quad k_2 \leftarrow f(t + 0.5\ h, y + 0.5\ h\, k_1) \\
&\quad k_3 \leftarrow f(t + 0.5\ h, y + 0.5\ h\, k_2) \\
&\quad k_4 \leftarrow f(t + h, y + h\, k_3) \\
&\quad\ t \leftarrow t + h \\
&\quad\ y \leftarrow y + h \cdot (k_1 + 2k_2 + 2k_3 + k_4)/6 \\
&\quad t_i \leftarrow t \\
&\quad Y_i \leftarrow y \\
&\text{END}
\end{aligned}
$$

Copy your Euler method file and save it in a new file called `rk4.m`, Adapt your code to implement the Runge-Kutta method. To test your implementations, proceed as follows. We start by *choosing beforehand* an exact solution

$$y(t) = e^{-t} \sin(t^2),$$

with $t_0 = 0$ and $T = \pi \approx 3.14159$. Find y', and use it to deduce the corresponding expression for $f(t, y)$ (**Your f should have both a t and a y in it. Simplify it to find the y!**). Run your implementation with resolutions $N = 100, 200, 400, 800, 1600$ and plot the maximum global error, defined by

$$\texttt{error(N) = max(abs(y}_{\texttt{exact}}\texttt{ - Y}_{\texttt{calculated}}\texttt{))},$$

versus N in a log-log plot,[2] where for each N, the max is taken over all time. Check that you obtain a line with the expected slope by comparing with the plot of $g(N) = N^{-4}$. Perform the same computation with the explicit Euler method given above and discuss your results.

[2]**Note:** log-log plots are used to find the relationship between two quantities related by a power function. For example, if $y = 5.4x^{-2.3}$, then $\log(y) = -2.3\log(x) + \log(5.4) \approx -2.3\log(x) + 1.6864$, so if we plot $\log(x)$ vs. $\log(y)$, we should see a line with slope -2.3. To plot a log-log plot in Matlab, just use `loglog` instead of `plot`. For example, you could use `loglog(1:N,error(1:N))` to plot your error.

Part II: Exploring Numerical Methods

Analyzing Error

Consider the initial-value problem

$$\begin{cases} y' = \dfrac{t^2}{1+y}, \\ y(0) = 1. \end{cases}$$

Our goal in this section is to compute the value of $y(2)$ to 3 decimal places using Euler's method.

(a) Using the Euler's method error bound we saw in class ($|\text{Error}| \approx C(T - t_0)h$, assume $C \approx 1$), estimate the step size h needed to get an error in $y(2)$ smaller than 10^{-3}. Based on this, what is the minimum number of steps using Euler's method should we take to get from $t_0 = 0$ to $T = 2$?

(b) Using the Euler and Runge-Kutta methods to compute y(2) with the number of steps you predicted in part (a). Compare these values.

(c) Repeat part (b) with twice as many steps. How does the computed value of $y(2)$ change? Does it seem that you have computed $y(2)$ correct to within 3 decimal places? Why?

(d) Solve the initial-value problem on paper and thus determine what $y(2)$ should be exactly.

Failure of numerical methods

Consider the initial-value problem

$$\begin{cases} y' = y - \dfrac{1}{(1+t)^2} - \dfrac{1}{1+t}, \\ y(0) = 1. \end{cases}$$

(a) Verify that $y(t) = \frac{1}{1+t}$ is a solution to the initial value problem.

(b) Graph the slope field for this ODE using dfield8.m, and the solution curve starting at $(t_0, y_0) = (0, 1)$ for $0 \leq t \leq 20$. Set the scale of the plot so that you can make a visual comparison of the solution curve with the exact solution $y(t) = \frac{1}{1+t}$.

(c) How does your graph in part (b) compare to the exact solution from part (a)? Based on the slope field, explain why the numerical solution fails.