BPHS 4090 (Fall 2013) - Computer Exercise 1: Numerical Solution to Differential Equations

Due Date: Sept. 20, 2013 5:00 PM

Instructions

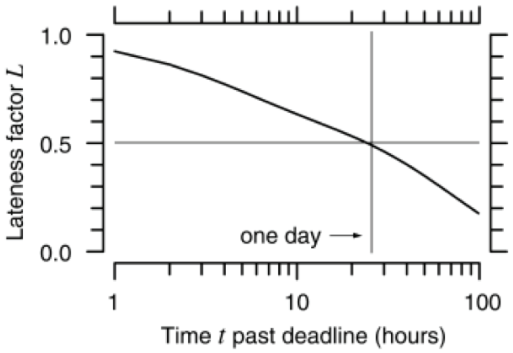
The primary purpose of this lab is to develop some understanding about how to numerically integrate differential equations in order to obtain particular solutions that satisfy specific initial conditions. A second goal is for you to gain further experience with basics of computer algorithms and writing scripts (for Matlab) that will perform computations based upon a paradigm you encode.

There will be two ways this assignments can be turned in. One is by hard copy, handed either to Prof. Bergevin or his mailbox (Petrie 129) by the above specified due date. Second (and preferred) is electronic submission via email. Electronic submissions are expected to be in PDF format and be submitted as one single file. Deviations from this format will be penalized. You are also expected to turn in the Matlab codes you are asked to write (attach these as separate .m files).

Also, the due date is firm and there is a severe lateness penalty (as per the course syllabus). The grade for a late report will be multiplied by a lateness factor

$$L = 0.3e^{-t/4} + 0.7e^{-t/72}$$

where t is the number of hours late. The lateness factor is plotted below. Notice that the maximum grade for a report that is more than ONE DAY LATE is less than 50%.



1 Background

Consider the one-dimensional differential equation of the form

$$\frac{dy}{dx} = f(x, y) \tag{1}$$

Depending upon the form of $f(x, y)$, the equation can be either autonomous or not, linear or nonlinear, etc... But at the most basic level, $f(x, y)$ is explicitly telling you how y is changing with respect to x. Suppose you start at some initial point (x_o, y_o) and you move forward some step Δx, then $f(x, y)$ provides information about what the corresponding change in y should be [or more specifically, how $y_o = y(x_o)$ changes to $y(x_o + \Delta x)$].

One of the simplest methods for numerically integrating differential equations is called *Euler's method*[1]. This method relates to the Taylor series approximation for $y(x)$ about x_o where only the first two terms of the expansion are considered (while not the focus of this particular lab, this concept of the series expansion for estimating $y(x)$ is a notion we will revisit next semester). Consider the definition of the derivative:

$$y'(x) = \frac{dy}{dx} = \lim_{\Delta x \to 0} \frac{y(x + \Delta x) - y(x)}{\Delta x} \tag{2}$$

Now assume Δx is small, but finite (we call this value the *step-size*). Considering the point $x = x_o$, we can rearrange terms such that we have

$$y(x_o + \Delta x) \approx y(x_o) + y'(x_o)\,\Delta x = y_o + f(x_o, y_o)\,\Delta x \tag{3}$$

Now we know all the terms on the right-hand side [the initial condition $y_o = y(x_o)$, the step-size Δx, and the derivative simply evaluated at the initial starting point, $f(x_o, y_o)$], thus this formula allows us to determine y at some different value of x. This knowledge is effectively similar to actually knowing the solution to the differential equation $y(x)$ (which may not actually be solvable analytically in closed form!).

This method we outlined is not entirely telling the whole story, as we can see when we consider things visually in Fig.1. The curve $y(x)$ is ultimately what we wish to determine, but we only know $dy/dx = f(x, y)$. The quantity $f(x_o, y_o)$ represents the slope of $y(x)$ at x_o (think of a tangent line to $y(x)$ at the point $[x_o, y_o]$). Given an initial condition $[x_o, y_o]$ and some step along x with size Δx, we can estimate $y(x_o + \Delta x)$. From the figure, it is clear that the sum of $y(x_o)$ and $f(x_o, y_o)\,\Delta x$ is in general only an approximation of $y(x_o + \Delta x)$. In particular, if $f(x, y)$ changes significantly over the course of Δx, then there can be a significant difference between the estimated value and the true value of $y(x_o + \Delta x)$ (we call this the error). Clearly, the smaller Δx is, the smaller that error will be.

Now imagine using the newly estimated point $[x_o, y(x_o + \Delta x)]$ as our new starting location and using it to estimate $y(x)$ at $x_o + 2\Delta x$ and so on. Thus given $dy/dx = f(x, y)$, one can estimate the solution $y(x)$ for a given initial condition and trace out the specific solution curve. This is the basic idea behind

[1] Leonhard Euler was a prolific Swiss mathematician from the 18'th century who has made numerous seminal contributions in many different fields, most notably mathematics and physics. As a random aside, his PhD advisor was Johann Bernoulli, another important figure in mathematical history.

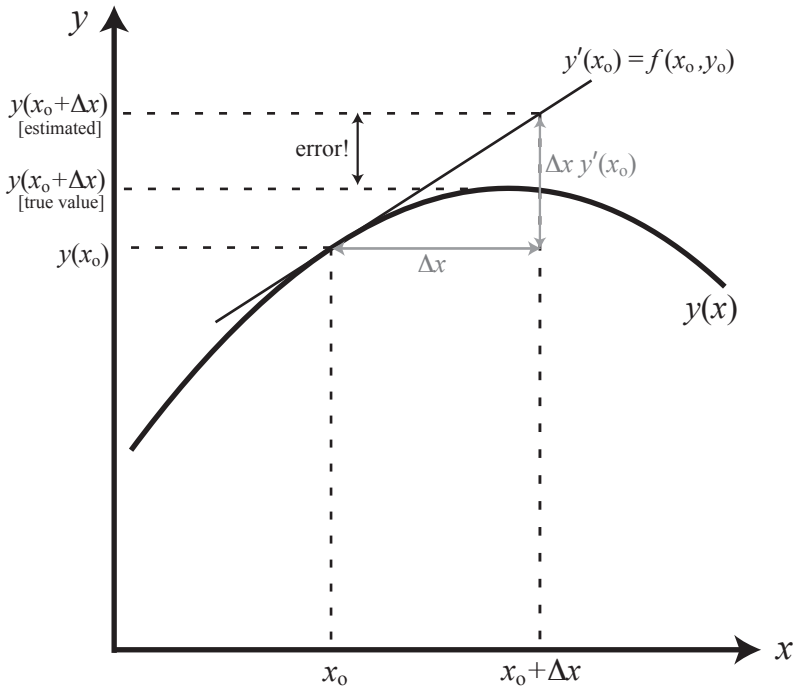


Figure 1: Schematic demonstrating how Euler's method can be used to estimate the solution curve $y(x)$. Keep in mind that from the ODE, you know dy/dx [that is, the slope of $y(x)$], and need to essentially backward engineer what $y(x)$ is.

how the computer draws solutions curves in dfield7 when you click on an initial condition in the slope field. Note that Euler's method does not give you an explicit formula for $y(x)$, but yields estimates of y for various (specified) locations along x.

For a given differential equation and a specified step-size, one can compute an estimated solution by hand using Euler's method. For example, consider the differential equation

$$\frac{dy}{dx} = y^2 - 1 = f(x, y)$$

Now suppose we wanted to estimate the value of $y(x)$ at $x = 1$ starting from the initial condition $(0, 0)$ [i.e., the solution curve we are interested in passes through $y(0) = 0$ and we wish to know approximately what the value of $y(1)$ is] using a step-size of $\Delta x = 1/5$. We know the solution to this differential equation, $y(x) = \frac{1-e^{2x}}{1+e^{2x}} = -\tanh(x)$, as solved using separation of variables and partial fraction expansion. So in this case we can compare our estimate to the exact value each step of the way.

We start at the initial condition such that $y(0) = 0$, a value we subsequently use for the next step along x. According to Euler's method, we next have $y(1/5) = y(0) + f(0, 0) \cdot (1/5) = 0 + (-1)(1/5) = -1/5$. Note that since we explicitly know $f(x, y)$, it can be evaluated at any point. Going further, we have $y(2/5) = y(1/5) + f(1/5, 1/5) \cdot (1/5) = -1/5 + (-24/25)(1/5) = -49/125$. We can just repeat this process, using the values computed at the last step and build up a table of values:

Euler's method estimates to $dy/dx = y^2 - 1$ where $y(0) = 0$					
step i	x_i	y_i	$\Delta y_i = f(x_i, y_i) \cdot \Delta x$	y_{i+1} (next step)	y_i (exact)
0	0	0	-0.2	-0.2	0
1	1/5	-0.2	-0.192	-0.392	-0.197
2	2/5	-0.392	-0.0963	-0.561	-0.380
3	3/5	-0.561	0	-0.698	-0.537
4	4/5	-0.698	0	-0.801	-0.664
5	1	-0.801	-	-	-0.762

It is apparent that there is some error associated with our estimate. That error will decrease as the step-size gets smaller (see Fig.1). As we saw above, it is possible to compute a solution by hand, but as the step-size decreases, more and more work needs to be done as there are more steps over the interval. This can be tedious and actually unwise, as one small algebraic mistake can propagate through the calculations and end up giving one an estimate that is incorrect. One way to get around this is to use a computer to perform the necessary computations.

2 Computer Coding 101

Below is a very brief/basic overview of some Matlab syntax. May be of (great) help, may not be.

You are asked in this lab to write a computer code to numerically solve a differential equation by writing a Matlab script using the paradigm outlined in the previous section. This exercise will entail using computer logic techniques that may or may not be familiar to you such as *for* loops and array building. This section is designed to provide a brief overview to get you started.

To write a Matlab script, open up Matlab and under 'File', choose to create a new M-file. This will bring up a new window where you can type commands (this is called the Editor). You need to save this file using a name such as *testcode.m*. You can then execute it at the command line in the main Matlab window by typing *testcode*. In the Editor, you can type various commands which are sequentially executed when the code is executed[2].

To start you off, below is an example code that creates an array of numbers starting at one hundred and going down to thirty nine in integer steps. This skeleton of a code will form a foundation for how you might want to think about setting up your code to solve the differential equations as outlined below. Think carefully about the syntax and make sure to ask questions if there is something you don't understand!

```
% testcode1.m                          My Name (date)

% code to generate an array y containing the integers from 100 to 39

startVAL= 100;       % initial value
endVAL= 39;       % ending value

for nn=1:abs(startVAL-endVAL)+1

% the for loop above basically tells the loop to go around 62 times,
% from nn=1 to nn=62; the reason why we didn't write for nn=1:62
% is because we wanted flexibility in case we chose to differ the
% initial of ending value and still get all the desired values

   % create array y, using nn as an index
   y(nn) = startVAL -nn+1;

% also use nn here as a means to subtract an integer value from
% the initial value; the +1 is so that we don't lose the initial value itself

end


% now I have a 62x1 array (i.e. a matrix containing 62 elements) called
% y that has the desired values
```

[2] You can add comments to your code by starting a given line with the % symbol. Commenting your code is a very good idea so to make it clear to others (as well as yourself!) as to what the various commands do.

3 Questions

Part I

Consider the differential equation

$$\frac{dy}{dx} = y^2 + 1$$

1. Solve this equation analytically to obtain an expression for $y(x)$. Your answer should contain an arbitrary constant.

2. Solve the differential equation numerically using Euler's method on the interval $x \in [0, 1]$ for the initial condition $y(0) = 0$. On a single figure, plot your estimated solution curve using the following step sizes for Δx: 0.5, 0.2, 0.1, 0.05, and 0.01. Make clear which curve corresponds to each step-size (Hint: use different line styles/colors).

3. On a different figure, plot your numerical solution above for $\Delta x = 0.05$ together with the exact solution you solved for analytically (given the specified initial condition). Make sure that it is clear which of the two different curves is which. Discuss any differences you see between the two plots and try to explain how those differences arise. How does this difference change as Δx gets smaller?

4. Extend the range of comparison between the numerical solution and the exact solution to $x \in [0, \pi/2]$. How does your numerical solution compare to the analytical solution as x gets close to $\pi/2$? Explain why one is larger than the other.

Part II

Consider the differential equation

$$\frac{dy}{dx} = y^2 - c$$

where c is a constant greater than zero.

1. Determine all equilibrium solutions and their stability.

2. Solve this equation analytically to obtain an expression for $y(x)$. Your answer should depend upon c and contain an arbitrary constant.

3. Write a code to solve the equation numerically using Euler's method on the interval $x \in [0, 5]$ for the initial condition $y(0) = 0$ and with $c = 4$. On a single figure, plot your estimated solution curve using the following step sizes for Δx: 0.5, 0.2, 0.1, 0.05, and 0.01. Make clear which curve corresponds to each step-size. How does the solution depend upon Δx?

4. Using $\Delta x = 0.01$, find solution curves for different initial conditions $y(0) = y_o$. How do the solutions depend upon y_o?

5. Explain your answer to the last part in terms of your analytic solution. Are the two results consistent?

6. What is the effect of varying c? Explain in the contexts of both your analytical answer and numerical simulations. Do both agree?

Extra Credit

Euler's method estimates the function $y(x)$ using the slope at the left end-point (at $f(x_o, y_o)$ that is). We have seen that this may lead to an over/under-estimate because the derivative is not necessarily constant over the interval spanned by Δx. Consider using the slope computed halfway along the step, where $x_{mid} = x_o + \frac{\Delta x}{2}$ and $y_{mid} = y(x_{mid}) = y_o + \frac{\Delta x}{2} f(x_o, y_o)$. Then the estimate over the entire interval becomes

$$y(x_o + \Delta x) \approx y(x_o) + f(x_{mid}, y_{mid})\, \Delta x \tag{4}$$

This is called the modified Euler method.

1. Explain graphically why this method can lead to a better estimate for $y(x)$.

2. Modify your Matlab code to use the modified Euler method for the ODE in Part II. Describe how the accuracy has changed with respect to your previous answer.

Above and Beyond

For those who want a further challenge, modify your code such that in incorporates a fourth–order *Runge-Kutta* (RK4) formulation. This is a higher-order method that provides significantly more accuracy[3]. Do some research to find the definition of this method and how to go about implementing it (you already have the core foundation laid with your codes implementing Euler's method). How does the accuracy improve and what is the resulting change in computational time?

[3] In a fashion similar to how Simpon's rule is better than the Midpoint/Trapezoid rules for numerical intergration (Riemann sums), which is in turn better than Left/Right rules, RK4 improves upon Euler's method.