Project 3 - Euler's Method

Due in class on Monday, 29 March 2010 in class. Please staple your project!

These notes basically are from the IODE project and are a modification of those written by P. Brinkmann. You are recommended to go to the Iode web site and click on Course Materials. You will find a longer description of the project in Project II and Lab II: Numerical Methods.

1. An introduction to numerical methods: Commands in Matlab

In this project, you will implement your own numerical method that is a modification of the Euler method. In order to do this, you need to understand how Matlab and Octave represent numerical solutions internally. The code in Figure 1 illustrates some of the main points, and you can type the code yourself at the prompt in the command window of Matlab or Octave. (But you will need to quit out of Iode before doing so, or else the plots might not show up correctly.)

The first command, $tc = 0 : 0.2 : 1$, creates a vector of t-values, ranging from 0 to 1 with step size 0.2. That is, it creates the vector $(0, 0.2, 0.4, 0.6, 0.8, 1)$.

The second command, $xc = \sin(pi * tc)$, creates a vector of x-values by evaluating the function $\sin(\pi t)$ at all the entries of the vector tc.

The third command, $plot(tc, xc)$, interprets tc as a list of coordinates on the horizontal axis and xc as a list of corresponding coordinates on the vertical axis. It plots all six points (t, x) and connects adjacent points with straight line segments. The resulting graph looks like a rough approximation of a part of a sine curve. We can obtain a better picture by decreasing the step size: if you replace the first line by $tc = 0 : 0.05 : 1$; and repeat the remaining two steps, then you will see a plot that looks like a piece of the sine curve.

The last few commands in Figure 1 show how to access certain information about the vector tc, such as its length (that is, the number of entries) and the value of the individual entries, numbered from one to six.

Figure 1: Representing and plotting functions with Matlab

```
>> tc = 0 : 0.2 : 1
tc =
0.000000.200000.400000.600000.800001.00000
>> xc = sin(pi * tc)
xc =
0.000000.587790.951060.587790.00000
>> plot(tc, xc)
>> length(tc)
ans = 6
>> tc(2)
ans = 0.20000
>> tc(3)
ans = 0.40000
```

Remember that there is a list of Matlab commands available on our course website. You must always use this format when working on this project.

2. Implementation of the Euler method in IODE

Now we are ready to inspect Iode's implementation of Euler's method. Recall that Euler's method has the following components. We will use variables (t, x) and the solutions will be $x(t)$.

We want to numerically compute the solution to the initial value problem

$$x'(t) = f(t, x),\ x(t_0) = x_0.$$

We begin at the point (t_0, x_0). Assuming a uniform step size h, we compute the n^{th}-value of the approximation x_n from the previous point (t_{n-1}, x_{n-1}) by the Euler update formula:

$$x_n = x_{n-1} + hf(t_{n-1}, x_{n-1}). \tag{2.1}$$

After n-steps, we obtain (t_n, x_n). This is supposed to be a good approximation of the true solution $x(t)$ at time t_n: $x(t_n)$.

To explore how Iode computes x_n, use the **Open menu item** (click on 'file' and then 'open' or in Matlab, open the Current Directory and click on the file euler.m) in the Matlab main window (not the Iode window!) to open the file *euler.m*. The file will open in an editor window. Don't change this file in any way but read through it. Figure 2 shows the contents of *euler.m*, without the comment lines (which begin with a percentage sign). We'll go through it line by line.

Figure 2: Matlab's implementation of the Euler Method

	1	**function** $xc = euler(fs, x0, tc)$;
For each i compute :	2	$x = x0$;
	3	$xc = [x0]$;
	4	**for** $i = 1 : (\mathbf{lenght}(tc) - 1)$
$h = t_{i+1} - t_i$	5	$h = tc(i+1) - tc(i)$;
$k_1 = f(t_i, x_i)$	6	$k1 = \mathbf{feval}(fs, tc(i), x)$;
$x_{i+1} = x_i + h \cdot k_1$, and then	7	$x = x + h * k1$;
append x_{i+1} to the vector of x values	8	$xc = [xc, x]$;
	9	**end**;

(1) Line 1 defines a new function xc called *euler*. When this function is called, it expects to receive three parameters as an input: 1) the parameter fs represents the function $f(t, x)$ from our ODE (2.1), 2) the value $x0$ is the initial x-value, and 3) tc is a vector of t-coordinates, like we have seen before (in particular the first entry of tc is $t0$). Line 1 also indicates that this function will return a value in the variable xc, which will turn out to be the desired vector of x-coordinates computed by Euler's method.

(2) Line 2 initializes the value of the variable x to be $x0$. The variable x always contains our current numerical approximation.

(3) Line 3 creates the vector xc that will contain our numerical approximations. Initially, it only contains the value $x0$. In particular, the first entry of xc is $x0$. So, we have $tc(1) = t0$ and $xc(1) = x0$.
(4) Line 4 is the beginning of a loop that lets the variable i range over all numbers from 1 to the length of the vector tc minus one. The body of this loop, lines 58, constitutes the Euler update step and will be executed for each value of i.
(5) Line 5 computes the step size h by computing the difference between the $(i+1)^{st}$ entry of tc and the i^{th} entry of tc. The expression $tc(i)$ stands for the i^{th} entry of tc, which we regard as the current t-value. Now, the variable x contains the numerical approximation of the solution at the current point $tc(i)$.
(6) Line 6 computes the slope of the solution at this point $tc(i)$ by evaluating the function given by fs at the point with coordinates $tc(i)$ and x. The variable $k1$ contains the result of this slope computation.
(7) Line 7 computes the Euler approximate solution value at the next point $tc(i+1)$.
(8) Line 8 appends this value to the vector of x-values, and thats it!

In Iode Project 3, when you write your own numerical routine for the Improved Euler Method, you can keep the framework of *euler.m*. You only need to change Line 1 in Figure 2, and then Lines 67 which compute the update formula.

3. Lab Project 3: Improved Euler Method

The improved Euler method is described by this improved update:

$$\begin{aligned} h &= t_{i+1} - t_i \\ k_1 &= f(t_i, x_i) \\ k_2 &= f(t_i + h, x_i + hk_1) \\ x_{i+1} &= x_i + h(k_1 + k_2)/2 \end{aligned} \tag{3.1}$$

Compare this line by line with the usual Euler method. We will now make a module in Iode for implementing this method.

(1) Explain the graphical meaning of the number k_1 and k_2. Draw a diagram. How is this scheme an improvement of the Euler method?
(2) Implement the Improved Euler Method by adding a module to Iode that computes solutions using this method. To do this, first OPEN the file *euler.m* using the menu in the Matlab main window. Immediately save this file under the new name *impeuler.m* by using the **File** → **Save As** menu item. Now you are ready to edit this file *impeuler.m* Line 1 in Figure 2 should now read:

$$\textbf{function } xc = impeuler(fs, x0, tc);$$

Do not use the extension notation *.m* here. Next, skip down to the end of the file, ignore any comment lines. At the end you will see lines (6)-(7) as in Figure 2 that describe the Euler update formula. Carefully modify these lines to reflect the improved Euler update given in (3.1). **Save your work!**

(3) Now test the new module *impeuler.m*. Start the direction fields module of Iode and choose the solution method **Other**. When prompted for the name of the module, enter *impeuler* (no *.m*). You will have to do this each time you choose Other. Now Iode will use this module to compute the solutions. Plot some solutions for ODEs where you already know the answer, like $f(t, x) = x$. You may have to debug the module if it does not work. Make sure you use the correct syntax.

Now that your module is working, do the following. Quit out of the direction field module of Iode and restart it to restore it to the default settings. We will study three initial value problems:

(1)
$$x'(t) = x,\ x(0) = -1.$$

(2)
$$x'(t) = x\sin(t) + e^{-\cos(t)},\ x(0) = e^{-1}.$$

(3)
$$x'(t) = e^x \sin(5t),\ x(0) = 0.8.$$

Relabel the variables in IODE so that t is the independent variable and x is the dependent variable.

WORK TO BE HANDED IN:

For each of these three ODEs, do the following and hand in the plots and explanations:

(1) For each solution method and for each of the three ODEs, compare solutions with $h = 0.1$ and $h = 0.5$. Try to explain the differences. You can experiment some more if you wish.
(2) Plot the exact solution by first solving the initial value problem by hand and then using the **Plot arbitrary function** command on the direction fields component of Iode.
(3) On the same plot, show the numerical solutions computed with Euler, your Improved Euler, and Runge-Kutta. Use different colors so the four plots are clearly distinguishable. On your print out, include the color code. Make sure to use the same step size for each of the four solution methods. Use display parameters $-3 \le t \le 3$ and $-3 \le x \le 3$ for all plots. Make sure you try at least two step sizes, $h = 0.1$ and $h = 0.5$. Choose a step size so that the Euler plot and the exact plot are roughly the same. If all four plots look the same, your step size is too small.
(4) Which numerical method (Euler, Improved Euler, or Runge-Kutta) is the best? Why?