18.03 Problem Set 2

Due by 1:00 P.M., Monday, March 1, 1999, in the boxes at 2-106, next to the Undergraduate Mathematics Office. Part II will be handed out Monday and won't look as long!

Syllabus

8. (F 19 Feb) Numerical methods: EP 6.1, 6.3; Notes G.3; Polking's chapter 5 is optional.

Part II: Second order linear equations

9. (M 22 Feb) Second order linear equations: EP 2.1, 2.2.

10. (W 24 Feb) Complex numbers: Notes C.

11. (F 26 Feb) Complex-valued functions; constant coefficent homogeneous second order linear equations: real roots: EP 2.3.

12. (M 1 Mar) Complex or repeated roots: EP 2.3, 2.4.

Part I.

8. (F 19 Feb) Notes I: 19.

9. (M 22 Feb) EP 2.1: 1, 5, 15.

10. (W 24 Feb) Notes C: 1–4, 6, 9, 11a.

11. (F 26 Mar) Notes II: 12ac, 14; EP 2.1: 33, 35, 37.

Part II.

8. (F 19 Feb) In this problem we use the three numerical methods described in class to estimate the number $e \simeq 2.718281828459045\ldots$. This number is *defined* as the value $y(1)$ of the solution to the initial value problem $\frac{dy}{dx} = y$, $y(0) = 1$.

(a) Suppose the step-size is $1/N$, with N steps. Let $y_0 = 1$ and y_n the nth step in the process, so that y_N is our estimate of e. Work out y_n in terms of y_{n-1} for each of the three processes (Euler, Heun, and Runge-Kutta). Simplify each as much as possible.

(b) Then express y_N *directly* in terms of N (using of course $y(0) = 1$).

(c) Make a table with three rows and four columns: a row for each process, and a column for each $N = 2, 4, 8, 16$. Using a calculator work out each entry to at least 5 places of accuracy.

(d) Make another table, showing the differences between your computed estimates of e and the true value. Theory indicates that halving the step size should multiply precision by about 2 in Euler, 4 in Heun, and 16 in Runge-Kutta. Is this borne out by your calculation?

(e) One way to estimate the cost of a method of solving the DE $dy/dx = f(x, y)$ is to keep track of the number of times $f(x, y)$ has to be evaluated. Each step of the Euler method uses one evaluation; Heun's uses two, and the Runge-Kutta method uses four.

Thus a 16-step Euler method is as expensive as a 8-step Heun and a 4-step Runge-Kutta. My question is: if these cost the same, which would you use? Decide this by computing six more ratios: the precision of Euler with $2n$ steps divided by the precision of Heun with n steps for $n = 2, 4, 8$; and the precision of Heun with $2n$ steps divided by the precision of Runge-Kutta with n steps for $n = 2, 4, 8$. What's your conclusion?

8.5 (F 19 Feb) I did not have time to test-drive the whole of the original version of this problem, handed out Friday, on Athena, and for reasons I don't fully understand it led to several different problems. The present problem fixes them (without introducing new ones, I hope!), and takes the opportunity to make some improvements. I am very grateful to the score of students who tried this out over the weekend and wrote me with detailed reports of their difficulties, and I apologize to everyone for the expense of time.

This problem will use MATLAB again. As usual it is very talky and not as long as it looks! In connection with his manual, John Polking has written three very simple routines called `eul`, `rk2`, and `rk4`, that implement respectively the Euler, Heun, and (fourth-order) Runge-Kutta methods. The code of these routines which is printed in Polking's book (pp 80–83), reflects an older version, but you can find the `*.m` files in the Athena implementation of MATLAB and look at them if you like. In this problem we'll use these routines to study the Runge-Kutta methods applied to the equation $y' = y^3 - 3y - x$ that you used MATLAB to study in PS1. We will investigate the solution with initial condition $y(0) = 0$.

So fire up MATLAB and type `format long`. The first thing to do is remind yourself of what the graphs of the solutions to this ODE look like, so invoke `dfield5`, enter the ODE $y' = y^3 - 3y - x$, adjust the range of the independent variable x to $[-4, 4]$, and hit `proceed`. You could tap the left mouse button with the cursor at the point $(0, 0)$, but `dfield5` offers a more precise way to enter initial conditions. On the `Display` toolbar hit the `Options` button and then select `Keyboard input.` A new window will open. Set the initial values of `x` and `y` both to 0 and hit `Compute`. A trajectory will appear on the `Display` window. You can close the `Keyboard input` window now.

Notice that the solution is plotted first forward and then backward. `dfield5` itself includes a numerical ODE solver, very much like `ode45` in MATLAB. They both use a sophisticated version of Runge-Kutta. Notice too that the solution disappears off the bottom of the page at about $x = 3.3$. We'll try to understand more about this below. Don't quit `dfield5`; it will be convenient for you to have it in front of you later.

In the MATLAB window, type `help eul` to see a succinct description of what `eul` expects for input and what it gives as output. It has four input fields, separated by commas. In my description I'll assume x is the independent variable and y is the dependent variable. The first field is the name of a file which contains a description of the ODE; we'll return to this in a minute. The second is the span, over which the ODE is to be solved; it has the form `[a,b]`. (This differs from what appears in Polking's book in Chapter 5, and reflects a change to make `eul`, `rk2`, and `rk4` parallel with `ode45`.) The third is the initial condition `y0`, giving the desired value of the solution y at a. The final field is a number `h` giving the stepsize.

The routine `eul` returns a pair of column vectors (i.e. columns of numbers). The first vector is the list of x values, and the second is the list of corresponding y values as computed by Euler's method. Thus a typical `eul` call might be

```
[x,y]=eul('cub',[0,3],0,.5)
```

This will apply Euler's method to the ODE described by the file `cub.m` over the interval $[0,1]$ using initial condition $y(0) = 0$ and stepsize .5. What remains is to explain how the ODE is stored. This is done by creating a file. You can summon an editor from within MATLAB. The Athena installation of MATLAB uses `emacs` as the default editor, and I will assume that this is what comes up when you type `edit` in the MATLAB window. In the window that comes up, type

```
function yprime=cub(x,y)
yprime=y^3-3*y-x;
```

Then save it to your the directory from which you launched MATLAB, which I will assume is your home directory. You do this by typing `<control>X <control>S` and then in the command line at the bottom of the screen complete the line beginning `File to save in: ~/` to read `~/cub.m`. (I have the Student Edition of MATLAB 5.0 running under Windows 98 at home. `edit` summons an editor/debugger which works very nicely. However, to get a new file recognized I seem to have to exit MATLAB and start it up again.) You can quit `emacs` by typing `<control>X <control>C`.

What you have done is to add a function to the MATLAB library at your disposal, called `cub`. (In the first version of this problem I asked you to call it `cubic`. In the Athena implementation of MATLAB there is a predefined function called `cubic`. This should not have caused a conflict, because MATLAB puts the directory from which MATLAB was launched very high in its search path, but apparently it did cause some students problems. It's probably good practice to see if a file name has been used already before defining your own, as we see from this experience. One way to do this is to type, for example, `help cubic`. If `cubic.m` is on the path you will see some simple documentation or else "`No help comments found in cubic.m.`" If not you will see "`cubic.m not found.`")

Our new function `cub` accepts two inputs, x and y (in that order) and returns $y^3 - 3y - x$. Try it out: type `cub(4,3)` and check that `ans` is 14 (not 15 as I said earlier!). This function gives y' in the ODE we are studying.

Now type `[x1,y1]=eul('cub',[0,3],0,.5)` and admire the result.

The routines `rk2` and `rk4` work in exactly the same way, but use respectively the Heun or second order Runge-Kutta algorithm and the fourth order Runge-Kutta algorithm instead of the Euler method. Type `[x2,y2]=rk2('cub',[0,3],0,.5)` and `[x4,y4]=rk4('cub',[0,3],0,.5)`.

In fact, the big MATLAB ODE solver, `ode45`, works the same way as well, except that you don't have to specify a stepsize—it does this for you, and adjusts it according to the slope of the direction field. In fact, `ode45` is so accurate that we'll take it as the truth for the purposes of this exercise. It's configured to attempt to give 6 place accuracy. To see it's attempt at a solution, type `[x,y]=ode45('cub',[0,3],0)`. Notice that the stepsize varies; this is part of the genius of the `ode45` implementation.

Now we'll visualize the result, using the MATLAB function `plot`. This takes a pair of column vectors, just like the `[x,y]` we just produced, regards it as a list of coordinates of points on the plane, and plots them. Type `plot(x1,y1,'*')`. The sequence of pairs $(x1, y1)$ appear on the `DFIELD5 Display` window.

The command `plot` plots onto the "current figure," which happens to be `DFIELD5`

`Display` at the moment. Things will get confusing if we continue to use that window, so I want you to create a new one. You can do this by typing `figure`. This will create a blank window on your screen, which should be labelled `Figure No. 3`. (Figure No. 1 is the `DFIELD5 Setup` window and No. 2 is the `DFIELD5 Display`.) You can set the current figure by typing `figure(3)` for example to set it to 3. You don't need to do this now because creating the window set the current figure to 3. Also, you can remove the *'s from `DFIELD5 Display` by selecting `Edit` from its command bar, then `Delete a graphics object.`, and then positioning the crosshairs on one of the *'s and clicking the left mouse button.

Now type `plot(x,y)` to record the output of `ode45`. Leaving off the final argument in `plot` causes it to interpolate between the points with straight line segments. You should have a smooth curve on the window now. Let's add some gridlines, by typing `grid`. Now we'll want to plot the results of `eul`, `rk2`, and `rk4`. If we just type `plot(x1,y1,'*')`, the curve will vanish and stars will appear. I want both simultaneously, and for this, type `hold on` first and then in sequence `plot(x1,y1,'*')`, `plot(x2,y2,'o')`, and `plot(x4,y4,'.')`. (You may want to touch the "full screen" button on the `Figure No. 3` tool bar to see the picture better.) Select `File` from the command bar of `Figure No. 3` and then `Print`, and turn this page in as part **(a)**.

Next, I want to think about the way this solution seems to head south a little further on. For this, type `[x,y]=ode45('cub',[0,4],0)` and watch the fireworks. If you scroll up you'll find that y has gotten very small (near $-\infty$), while x has pretty much stopped growing. (The list of `y` values may have gotten so long that it fills your history buffer. In this case you can list the `x` values again by typing `x`.) The stepsize got very very small at the end, didn't it, as `ode45` tried to adjust for the tremendously large values of $y^3 - 3y - x$ it encountered. This reflects the fact (though it doesn't prove) that this solution "blows up" in finite time. The solution to our initial value problem *can't be extended* to arbitrarily large values of x. Its graph becomes asymptotic to a vertical straight line, $x = a$. For **(b)** write down `ode45`'s estimate of a.

Finally, I want to extend this solution to *negative* values of x, because some really bad behavior shows up there. Under my Student Edition of MATLAB 5.0, `eul` and the rest accepted intervals of the form `[0,-3]`, but this doesn't seem to work in the unix installations. (In addition, to be honest, the lessons learned from stepping backwards are a little obscure.) So I want to start with a general first order ODE $dy/dx = g(x, y)$ and introduce a new variable, say t, related to x by $t = -x$, and think of y as a function of t instead of x. This will have the effect of reflecting the graph of $y(x)$ through the y axis, right? For you, **(c)** is: verify that y satisfies the ODE $dy/dt = -g(-t, y)$.

In our case, the new ODE is: viewing y as a function of t, $y' = -y^3 + 3y - t$. Install this into `dfield5` and verify that the direction field is simply the reflection around the y axis of the direction field we had before.

Now, finally, on a new figure, use `plot` as before to plot the estimates by `ode45`, `eul`, `rk2`, and `rk4`, of the solution with initial value $(0, 0)$, through $t = 3$. Again use stepsize .5 for the last three and have `plot` use, successively, interpolated line segments, `*`, `o`, and `.`. Finally, using `'>'`, plot the values given by `rk4` using stepsize .25. (They should lie right on the solution given by `ode45`.) Add a grid, print this out, and turn it in as **(d)**.

That last try by `eul`, at $t = 3$, was pretty bad, wasn't it? It forced `plot` to rescale. **(e)**

Explain why `eul`'s guesses oscillate increasingly wildly about the true solution.

9. (M 22 Feb) Show that the following pairs of functions are linearly independent on any interval $[c, d]$ with $c < d$, by showing that $af + bg = 0$ forces $a = b = 0$ by checking a few values.

(a) $\{\sin(x), \cos(x)\}$.

(b) $\{e^x, xe^x\}$.

(c) $\{x^m, x^n\}$, where $m \neq n$ and $c > 0$.

10. (W 24 Feb) (a) The complex exponential makes some standard integrations from 18.01 easier. For example, using the fact that

$$\mathrm{Re}\ e^{(k+i\omega)t} = e^{kt}\cos(\omega t),$$

show that

$$\int e^{kt}\cos(\omega t)\,dt = e^{kt}\,\frac{k\cos(\omega t) + \omega\sin(\omega t)}{k^2+\omega^2} + c.$$

Draw the point $k + i\omega$ on the complex plane and mark r and φ (real) for which $k + i\omega = r\,e^{i\varphi}$. Express ω/k in terms of φ, and r in terms of ω and k. Compute the integral again to find

$$\int e^{kt}\cos(\omega t)\,dt = \frac{1}{r}\,e^{kt}\cos(\omega t - \varphi) + c. \tag{1}$$

(b) Using (1) and the method of integrating factors, give the general solution to the differential equation

$$\frac{dx}{dt} + kx = ak\cos(\omega t).$$

11. (F 26 Feb) The following observations were made in class about the second order linear homogeneous constant coefficient equation

$$y'' + py' + qy = 0.$$

Let $\omega = \sqrt{|(p/2)^2 - q|}$. If $p^2 < 4q$ the equation is underdamped and its general solution is

$$y = e^{-px/2}(a\cos(\omega x) + b\sin(\omega x)). \tag{2}$$

If $p^2 = 4q$ it is critically damped its general solution is given by

$$y = e^{-px/2}(a + bx). \tag{3}$$

If $p^2 > 4q$ it is overdamped and the general solution is given by

$$y = e^{-px/2}(c_1 e^{\omega x} + c_2 e^{-\omega x}). \tag{4}$$

It will be convenient in this problem to rewrite this using the *hyperbolic functions*

$$\cosh(z) = \frac{e^z + e^{-z}}{2}, \quad \sinh(z) = \frac{e^z - e^{-z}}{2}$$

as

$$y = e^{-px/2}(a\cosh(\omega x) + b\sinh(\omega x)). \tag{5}$$

(a) Compute a, b in terms of c_1, c_2 to relate (4) and (5).

(b) Take the equations (2), (3), and (5), and compute the two constants a and b given the initial conditions $y(0) = 1, y'(0) = 0$. Write down the corresponding particular solutions.

These three functions look very different from one another, but in fact as p and q vary the solutions vary smoothly too, even if we pass though the critical value where $p^2 = 4q$.

(c) For any fixed value of p let y_p be the solution to

$$y'' + py' + 4y = 0, \tag{6}$$

(where p is a constant) for which $y(0) = 1, y'(0) = 0$. Show that for any x,

$$\lim_{p \to 4} y_p(x) = y_4(x).$$

(You have to consider the cases $p > 4$ and $p < 4$ separately. Notice that as $p \to 4$, $\omega \to 0$. L'Hopital's Rule or the definition of the derivative will come in handy.)