Math 447

Due: Friday, 24 April 2015

Programming Assignment: Initial-Value Problem for Ordinary Differential Equations

Consider the initial-value problem

$$y'(t) = f(t, y),\ y(t_0) = y_0,\ \text{from } t = t_0 \text{ to } t = t_f. \tag{1}$$

1 Develop and Test your code

1.1 Euler's Method

Consider the IVP

$$y' = -t * y^2, \quad y_0 = 2,\ \text{from } t = 0 \text{ to } t = 2.$$

Program Euler's method to solve this initial value problem (IVP) in a file `euler.m`, which is a function that takes arguments `h` (the step-size), `t0` (the initial time), and `tf` (the final time), and outputs a vector `t` containing all the time values, and a vector `y` containing all the function values. Your code should work for arbitrary choice of the step-size h, so long as h is small enough (large h should give bad oscillations, i.e., instability).

The (forward) Euler method is

$$y_{k+1} = y_k + h * f(t_{n+1}, y_{n+1})$$

Assume that the interval $[0, 2]$ is divided into $N-1$ equal sub-intervals (so there are N points) and denote the length of the sub-intervals by h. Then $h = \frac{2-0}{N-1}$, and $t_k = 0 + hk$ for $k = 0, 1, \ldots, N-1$.

Verify that the code works correctly by comparing the results with the exact solution,

$$y_{\text{exact}}(t) = \frac{2}{t^2 + 1},$$

in a testing file `testEuler.m`, using the L^∞-norm of the error (i.e., the maximum of the absolute value of the difference).

$$\texttt{error} = \max_{0 \le i \le N-1} |y_{exact}(t_i) - y_i|$$

where i is the approximate solution computed by Euler's method. (Note that MATLAB starts indexing at 1 instead of 0, so your index may be a little different. Also, MATLAB's `max` and `abs` functions may be useful to you here.)

1.2 Improved Euler's Method and RK4

(Note: We will discuss these methods on Monday, or you can find them, e.g., in an old ODE book or on Wikipedia.)

Now, implement the improved Euler's method, also called the "Huen Method" and "Runge-Kutta-2". Use your code for the improved Euler's method to solve the ODE above. Name it `rk2.m`.

Do the same with Runge-Kutta-4. Name it `rk4.m`.

2 Convergence Rates for One-Step Methods

Now that we have developed working code, we will compare the order of the error in each of the methods as described below.

Use your programs to compute the approximate solution of the IVP (1) for $N = 5$, 10, 20, 40, 80, 160, 320, 640, 1280, 2560 using forward Euler, Modified Euler, and RK-4. Plot h versus h^p together with h versus the error in *loglog*-plot in MATLAB. Use $p = 1$ for Euler and $p = 2$ for Modified Euler, and $p = 4$ for RK-4. Include there three plots in your report (just print them), and briefly discuss your findings.

MATLAB tips: Type the MATLAB command `help plot` for some useful plotting commands. The MATLAB function `loglog` may also be helpful. Here is a way to plot two functions on the same plot:

```
x = linspace(0,1,100);
plot(x,x.^2);
hold on;
plot(x,x.^3);
```

3 Backward Euler and Stability

Consider the IVP

$$y' = 10 - y, \quad y_0 = 2, \text{ from } t = 0 \text{ to } t = 10.$$

Implement the *backward* Euler method to solve this. This will require some minor algebra. Name it `backwardEuler.m`. The backward Euler method is:

$$y_{k+1} = y_k + h * f(t_{n+1}, y_{n+1})$$

Backward Euler has the same order of accuracy as forward Euler (check it with your code if you want), so we won't look at that here. Instead, we will look at the stability. Taking steps is expensive, so we usually want to take as few steps as we can (i.e., we want to take a small number of large steps). Compare your *forwardEuler.m* with your `backwardEuler.m` on this problem. How small can you make N (i.e., how large can you make h) for these two methods, while still trusting the accuracy? (Note: $h = 10/(N-1)$.) What happens for the two approximations when N gets really small? Print out a graph of an interesting case.

4 Bonus (30 pts possible)

Read about Newton's method, or the binary search algorithm for finding the zeros of functions. Implement backward Euler, approximately solving for y_{n+1} using Newton's method (or binary search) at each step (you can have the derivative function in Newton's method input separately, taking the derivative by hand on paper). Keep your Newton solver (or binary solver) in a separate file named `newton.m` (or `binarySearch.m`), and call it in your backward Euler loop. Use your code to solve

$$y' = 0.5 * (1 - y/100) * y + 10 * \sin(t), \quad y_0 = 10, \text{ from } t = 0 \text{ to } t = 40.$$

with backward Euler. This is the Logistic equation from population dynamics with periodic forcing.