AMS 213B Homework 4 – due Friday, May 6, 2016

Note: Please submit your MATLAB codes to your git repo.

Consider the BVP for **Problem 1** and **Problem 2**, given by

$$\begin{cases} u''(t) = f(u) + g(t), \quad 1 \le t \le 2, \\ u(1) = 0, \quad u(2) = \ln 2, \end{cases} \tag{1}$$

where $f(u) = -(u')^2 - u$ and $g(t) = \ln t$. The exact solution is given as $u(t) = \ln t$.

Problem 1

Based on the MATLAB example code of the shooting method, please modify the code to implement a shooting method algorithm to solve this *nonlinear* BVP. Please run your code with the following choices:

- Choose the Forward Euler method for the time marching of the IVPs.
- Use three different temporal resolutions, $N = 8, 16, 32$.
- Use the MATLAB example root find code which uses the Newton's method to iteratively find a best possible guess of the initial slope at $t = 0$, $y_2^{(k)}(0)$. For this, you would need a small threshold value $\epsilon > 0$ so that your exit conditions of the root finding iteration becomes

  $$\text{exit if estimator} = |h^{(k)}/(h^{(k)})'| < \epsilon \text{ or } N_{iteration} > 100. \tag{2}$$

  Notice here that the second exit condition will prevent the root finding search from going to an infinite loop when a choice of your initial guess $y_2^{(0)}(0)$ is not proper. Please use $\epsilon = 10^{-5}$, and use the exact derivative method to obtain $(h^{(k)})'$ (note: this exact method has been already done in the example code).

(a) Convert the second-order ODE to a system of first-order ODEs.

(b) How many iterations do you see in finding the best possible root $y_2^{(k)}(0)$ for the three different temporal resolutions? Use $y_2^{(0)}(0) = 1$ for this case.

(c) Now try $y_2^{(0)}(0) = -1, 0, 7, 8$ and see how many iterations $N_{iteration}$ you need using $N = 16$ for each initial guess. Do you see which initial guess(es)

make the solution diverge? Which choice of initial guess among the four make the the solution converge the fastest and slowest? Based on this observation, can you find experimentally the minimum and maximum values of the initial slopes within the range $-1 \leq y_2^{(0)}(0) \leq 8$ that make the solution converge? Please find the minimum and maximum slopes up to the tenth decimal points (e.g., 5.9).

(d) Plot all your results with the exact solution.

Problem 2

Based on the MATLAB example code of the finite difference method, please modify the code to solve the above BVP using the finite difference method we studied in the class. Note that you are going to need to solve a linear system $\mathbf{Ax} = \mathbf{b}$ for which you can use the MATLAB's command `linsolve` (please know what this command does to solve the linear system). Please do NOT use the direct inversion. Use the same grid resolutions, $N = 8, 16, 32$ as in **Problem 1**.

(a) Write down the forms of $\mathbf{A}$, $\mathbf{x}$, and $\mathbf{b}$ explicitly using the second-order finite differencing formulas for u' and u'' we studied in the class.

(b) In $\mathbf{b}$ you notice that there are U^n, $n = 0, 1, \ldots, N+1$ which are not known yet. To execute your code, you need an initial guess for U^n for which you can use a straight line through $(t_a, u(t_a)) = (1, 0)$ and $(t_b, u(t_b)) = (2, \ln 2)$. With this initial guess of $U^{n,(0)}$, you can initiate the run until the L_1 error between the current solution and the previous solution, given by

$$||E||_1 = \Delta t \sum_{n=1}^{N} |U^{n,(k)} - U^{n,(k-1)}|, \tag{3}$$

becomes smaller then ϵ, say, $||E||_1 < \epsilon$. Please use $\epsilon = 10^{-5}$. Plot your three numerical solutions at $N = 8, 16, 32$ together with the exact solution.

(c) What are the number of steps for convergence, i.e., $||E||_1 < \epsilon$, at each resolution?

(d) Compared to the solutions of the shooting method at the same resolutions, please quantify which one is more accurate based on the L_1 error defined by

$$||\tilde{E}||_1 = \Delta t \sum_{n=1}^{N} |u_{exact}(t^n) - U^n|. \tag{4}$$

Problem 3

Please modify the MATLAB example code, `ForwardEuler_while.m`, and implement two implicit methods, (a) the Trapezoidal method and (b) the Backward Euler method to solve the stiff problem we did in the class (Ex 3 from Lecture note 10). Reproduce the two results in Figure 9.4 in the lecture note on $0 \le t \le 3$.

In order to implement an one-step implicit method, you need an iterative algorithm to search for the unknown, i.e., U^{n+1}, that appears in both left and right hand sides of the difference equation. For examples,

(a) Trapezoidal method:

$$U^{n+1} = U^n + \frac{\Delta t}{2}\Big[f(U^n) + f(U^{n+1})\Big], \tag{5}$$

(b) Backward Euler method:

$$U^{n+1} = U^n + \Delta t f(U^{n+1}), \tag{6}$$

In MATLAB, this can be done very easily using the single-variable nonlinear zero finder with `fzero`. For instance, the Backward Euler method implements a line such as

```
U_new=fzero(@(U_new) U_new-(U_soln(n)+dt*f(U_new,t(n))),U_soln(n));
```

This command line is pretty much everything you need for implicit scheme implementations using MATLAB, although however, you can write your own routine by modifying the example code `rootFind.m` we used in the class.

Remark: There is a typo in Ex 2 in Lecture note 10: the correct ODE in the IVP should be given as

$$\begin{cases} u'(t) = f(t,u) + g(t), \;\; 0 \le t \le 3, \\ \\ u(0) = 1, \end{cases} \tag{7}$$

where $f(t,u) = \lambda(u - \cos t)$ and $g(t) = -\sin t$.