Mat1062: Introductory Numerical Methods for PDE

Problem Set 2

Saturday, February 6, 2016

due: Monday February 22 by 5pm

1. **von Neumann stabilty of the θ-method.** In this problem, you will do the von Neumann stability analysis of a variety of schemes on $h\mathbb{Z}$. For each scheme, find whether or not there is a stability constraint on the size of the timestep, k. Try to get pen-and-paper arguments where you can but if you can't then feel free to use the computer to explore and come to a conclusion.

   (a) The θ-method for $u_t = Du_{xx}$:

   $$\frac{u_i^{n+1} - u_i^n}{k} = \theta L u^{n+1} + (1-\theta) L u^n$$

   where L is the usual centered-difference approximation for $D\,\partial^2/\partial x^2$.

   (b) The θ-method for $u_t = Du_{xx} + \gamma u$, again with a centered-difference approximation for $\partial^2/\partial x^2$. How does your answer depend on the sign of γ?

   (c) The θ-method for $u_t = \beta u_x$ with a centered difference approximation for $\partial/\partial x$. How does your answer depend on the sign of β?

   (d) The θ-method for $u_t = \beta u_x$ with a right-difference approximation for $\partial/\partial x$. How does your answer depend on the sign of β?

2. **A very special value of λ.** Revisit the truncation error analysis for the Forward Euler method. For general λ, we found that that the truncation error is $\mathcal{O}(k^2 + kh^2)$. However, there exists a $\lambda_0 \in (0, 1/2]$ for which the truncation error is like that of Crank-Nicolson — Forward Euler with this choice of refinement path will be much more accurate. Find this value of λ_0 via the truncation error analysis.

3. **Write a θ-method code.** Write a code for the θ-method (any $0 \leq \theta \leq 1$) for $u_t = Du_{xx}$ on $X_L < x < X_R$, with homogeneous Dirichlet boundary conditions at $x = X_L, X_R$. Because the θ-method is implicit if $\theta > 0$, you will need to solve a tri-diagonal linear system at each time step. In principle, you should write a tri-diagonal solver but to spare you this, create the full matrix that has to be inverted and then solve the required problem $AX = b$ via matlab's "slash" command (via $X = A\backslash b$). The slash command is quite robust and will solve the problem even if A is not invertible.

4. **Convergence study: simultaneously refining h and k.** Pick some simple Fourier initial data (such $\sin(\kappa x)$, $\cos(\kappa x)$, where κ has been chosen so that the initial data satisfies the boundary conditions) and compare your computed solution to the exact solution at a fixed time. *Make sure to choose $D \neq 1$ and $\kappa \neq 1$ because if you choose those values to equal 1 then there could be bugs that you don't see simply because multiplying by D or κ is the same as not multiplying at all if $D = 1$ or $\kappa = 1$.*

   From the convergence proof, $h \to 0$ and $k \to 0$ in such a way that $\lambda < 1/2$ is held fixed, we know that Forward Euler with centered differences gives us numerical approximations that converge to the true solution of the diffusion equation. Because you chose simple initial data, you know what the true solution is and so you can define the error at any moment in time.

You want to test your θ-method code, refining both h and k. Run your code in each of the following scenarios.

- $\theta = 0$. Choose $\lambda < 1/2$ and choose h and k so that $Dk/h^2 = \lambda$. Compute up to a certain time T and compute the L^∞ norm of the error. Now halve h and quarter k and repeat the process.
- $\theta = 0$. Choose $\lambda = \lambda_0$: the value you found in Problem 2. choose h and k so that $Dk/h^2 = \lambda_0$. Compute up to a certain time T and compute the L^∞ norm of the error. Now halve h and quarter k and repeat the process.
- $\theta = 1/2$. Choose h and k and compute up to a certain time T and compute the L^∞ norm of the error. Now halve both h and k and repeat the process.
- $\theta = 1$. Choose h and k and compute up to a certain time T and compute the L^∞ norm of the error. Now halve both h and k and repeat the process.

Create a pair of log-log plots. In one log-log plot, plot $\log(err(h,k))$ versus $\log(h)$ for the results from the above scenarios. In the other log-log plot, plot $\log(err(h,k))$ versus $\log(k)$ for the results from the above scenarios. Explain the observed slopes in terms of truncation error analysis.

Also, keep track of the runtimes for the four scenarios, using matlab's tic toc command. For each scenario, find the amount of time it would take to reach en error of 10^{-10}. Use least-squares to fit the exponents.

5. **Do norms matter for this problem?** For one of the scenarios in Problem 4, you want to measure the size of the error using three different norms: the L^∞ norm, the L^1 norm, and the L^2 norm. This will give you three things that you can plot on a log-log plot. Plot all three and see if the choice of the norm makes a difference in terms of the rate of convergence. (From section 8.3 of the lecture notes, you know that norms don't matter for this problem; I just want you to check it yourself!)

6. **Convergence study: fixed h, decreasing k.** In problem 4, you refined both h and k. This was, in effect, testing both the time-stepping part of the code and the spatial discretization part of the code at the same time. If you didn't get the right behaviour, you'd know there's a bug and you'd have to figure out: "Is my time-stepper bad or is my spatial discretization bad or are both bad?"

   The first step would be to test the time-stepper. If you fix N (the number of spatial subintervals) then you can consider the various time-steppers as time-stepping an $N-1$-dimensional system of ODEs. You can study the convergence your code as $k \to 0$. You are testing the numerical ODE aspect of your code; see the truncation error discussions in Section 5.3.

   NOTE: Because N is fixed, you don't know what the true solution is! If you wanted to define an error, you'd have to know the true solution of the $N-1$-dimensional system of ODEs. Because you don't know the true solution, you're going to have to compare the discrete solutions to one another — see the discussion right before section 8.1 in the lecture notes.

   Run your code in each of the following three scenarios.

- $\theta = 0$. You've fixed h. Choose k so that $\lambda < 1/2$. Compute up to a certain time T. Now halve k and repeat the process.
- $\theta = 1/2$. You've fixed h. Choose k and compute up to a certain time T. Now halve k and repeat the process.
- $\theta = 1$. You've fixed h. Choose k and compute up to a certain time T. Now halve k and repeat the process.

For each sceario create the analogue of Table 2 in Section 8 of the class notes. Also create a single log-log plot that presents $\log(\|\text{diff}(k)\|)$ versus $\log(k)$. Here "$\text{diff}(k)$" is the difference between a solution and the next finer solution; see the first column of Table 2. Add "sight-lines" to your plots to show if the curves have the expected slopes. (That is, if you expect the curves to have slope 1 then add dashed-lines to the plot that have slope 1, with whatever vertical shift is needed so that the sight-lines are close to, but not overlapping, the data curves.)

As a sanity check, you should make the analogue of Table 1 or plot the plots of Problem 4 and see what would have happened had you tried to define the "error" as the deviation from the solution of the PDE. Would you have seen the right ratios? Might you have concluded, wrongly, that your code was buggy? (No need to hand this in!)

There's an art to choosing a good range of k values. If k is "too large" then the errors won't be in the convergence regime (the higher-order terms in the Taylor series expansions won't be small enough for the ratios to work out) and if k is "too small" then you'll get contamination from round off error.

7. **Convergence study: fixed k, decreasing h.** Repeat the above exercise but with a fixed value of k. You will need to choose k quite small so that the contributions from k in the higher-order terms of the truncation error are quite small.

   In problem 4, you used the error — you subtracted off the true solution of the PDE. In problem 6, you didn't know the true solution of the system of ODEs and so you looked at the norms of the differences of subsequent refinements. For this problem, try *both* approaches.

   Run your code in each of the following three scenarios. You can present the log-log plots only, if doing the tables is too much of a hassle.

   - $\theta = 0$. You've fixed k. Choose h so that $\lambda \ll 1/2$. Compute up to a certain time T. Now halve h and repeat the process. Keep track of your value of λ because when h gets quite small, your λ will be greater than $1/2$ and your code will go unstable.
   - $\theta = 1/2$. You've fixed k. Choose h and compute up to a certain time T. Now halve h and repeat the process.
   - $\theta = 1$. You've fixed k. Choose h and compute up to a certain time T. Now halve h and repeat the process.