MAE 250H, Spring 2019
J. D. Eldredge
Homework 4, Due Tuesday, April 30

This homework is focused on solutions of two-dimensional diffusion equation. In anticipation of solving Navier–Stokes (which we will do right after this homework!) we will work on a staggered grid. The diffusion problem will be solved on edge data with Dirichlet boundary conditions (as for velocity).

Solution of 2-d diffusion equation. Here, you are to solve the 2-d diffusion equation

$$\frac{\partial f}{\partial t} = \nu \nabla^2 f$$

using 2nd-order central differencing for the Laplacian (a 5-point stencil) and two different choices of time marching scheme, and then test the solver on two problems:

1. Decay of a Gaussian pulse. For an initial condition

   $$f(x,y,0) = f_0(x,y) = \frac{1}{\pi\sigma_0^2} \mathrm{e}^{-(x^2+y^2)/\sigma_0^2}$$

   the exact solution at time t is simply the same function with $\sigma(t) = \sqrt{\sigma_0^2 + 4\nu t}$. Solve this with, say, $\sigma = 0.3$ on a domain of size $[-4,4] \times [-4,4]$ and zero on all boundaries. Make sure you only solve for sufficiently short times so that the solution does not reach the boundaries (at which point the exact solution would be wrong).

2. Spread of boundary data. Here, the initial condition is zero everywhere on a domain of size $[0,1] \times [0,1]$, and the boundary conditions are zero on all walls except the upper wall, where it is

   $$f(x,1,t) = \sin(\pi x).$$

   The exact solution to this problem can be obtained from separation of variables, and is

   $$f(x,y,t) = f_h(x,y,t) + f_p(x,y),$$

   where the homogeneous solution is

   $$f_h(x,y,t) = \frac{2}{\pi} \sin(\pi x) \sum_{n=1}^{\infty} \frac{(-1)^n n}{n^2+1} \mathrm{e}^{-\nu(n^2+1)\pi^2 t} \sin(n\pi y)$$

   and the particular (i.e. steady-state) solution is

   $$f_p(x,y) = \frac{\sin(\pi x)\sinh(\pi y)}{\sinh \pi}$$

For the time marching, you are to evaluate **both** 4th-order Runge-Kutta and 2nd-order trapezoidal (i.e. Crank-Nicolson). For the former, there is little difference with how you used it in the 1-d problem. You simply have a different right-hand side, and with some new consideration for non-zero boundary conditions. For the latter, you should use operator splitting with the ADI method. As we will discuss in class, this leads to the following discrete system,

$$\left(I - \frac{1}{2}\Delta t A_x\right)\left(I - \frac{1}{2}\Delta t A_y\right) u^{n+1} = \left(I + \frac{1}{2}\Delta t A_x\right)\left(I + \frac{1}{2}\Delta t A_y\right) u^n + \Delta t (bc)^{n+1}, \qquad (1)$$

where the full right-hand side matrix has been split $A = A_x + A_y$ into approximations to $\partial^2/\partial x^2$ and $\partial^2/\partial y^2$. Let's define $u^{n+1/2} = \left(I - \frac{1}{2}\Delta t A_y\right) u^{n+1}$ and b as the entire right-hand side of (1). Then, this factored system requires solution of two problems. First, solve

$$\left(I - \frac{1}{2}\Delta t A_x\right) u^{n+1/2} = b \qquad (2)$$

for $u^{n+1/2}$, then solve

$$\left(I - \frac{1}{2}\Delta t A_y\right) u^{n+1} = u^{n+1/2} \qquad (3)$$

for the final end-of-step solution, u^{n+1}. For the success of this method, it is critical to notice that both of these problems should involve solutions of regular tridiagonal systems. However, the data needs to be organized appropriately for the matrices to actually have contiguous tridiagonal bands. If your grid data is stored in a two-dimensional array, with the first index representing variation in the x direction and columns representing variation in the y direction, then it is quite simple to organize the data as needed.

Equation (2) represents independent tridiagonal equations for each of the interior rows of the grid. For example, for x edge components, there are N_y interior rows (the number of interior cells in the y direction), and each of these rows consists of a triadiagonal equation of size $N_x - 1$. For the y edge components, these sizes are $N_y - 1$ and N_x, respectively.

Julia and Matlab (like Fortran) store data in column-major format, meaning that the data in an array is physically stacked matrix column by matrix column. Since each matrix column represents a single grid grow, the data is perfectly set for the tridiagonal solutions in the x direction, to obtain $u^{n+1/2}$ for each row. However, note that C and $C++$ store arrays in row-major format, so they are set up for solving the problem in the grid columns first.

The system (3) represents independent tridiagonal equations at each of the interior columns of the grid. Here, the data is not set up quite right (in Matlab and Julia). But if we simply transpose the data (i.e. take $u = u'$ in Julia or Matlab), then we are set for another round of tridiagonal solves. We simply have to remember to **transpose back** after we are done.

Generally, to avoid build-up of round-off error, it is advisable to swap these every time step, so that you actually do the column solves first and row solves second in alternating time steps. This is the meaning of "alternating" in ADI.

When solving the test problems, please do the following:

- Verify that the spatial order of accuracy is as expected.
- Determine the stability limits on the Fourier number $\nu\Delta t/\Delta x^2$. You should find that there are no limits for Crank-Nicolson.