Fully Implicit Differencing

In Exercise 9.1 you should have found that the simple explicit integration scheme you used was *unstable* unless you chose a very small time step. This is quite typical of initial-value solvers for partial differential equations: they can be fast, but are prone to instability.

By applying the von Neumann stability analysis and looking for the fastest growing unstable modes, we found that the scheme is unstable unless

$$\alpha = \frac{D\Delta t}{\Delta x^2} \leq \tfrac{1}{2}\,.$$

In other words, the time step must satisfy

$$\Delta t \leq \frac{\Delta x^2}{2D}\,.$$

Check for yourself that the scheme you programmed is in fact stable when this condition is met.

This limit on the time step represents quite a severe restriction, especially when you realize that you generally want to reduce Δx in order to improve spatial resolution, but each reduction of Δx by a factor of 2 reduces Δt by a factor of 4 and hence the speed of the code by a factor of 8.

The way to deal with this sort of problem in the case of ordinary differential equations is to use *implicit* differencing. In the Euler method, for example, the explicit scheme for solving the equation

$$\frac{dy}{dx} = f(x, y)$$

simply sets

$$y_{n+1} = y_n + \Delta x f(x_n, y_n)\,,$$

and also is prone to instability. However, the implicit form,

$$y_{n+1} = y_n + \Delta x f(x_{n+1}, y_{n+1})\,,$$

while harder to solve, is *unconditionally* stable. Generally, the latter scheme is very good at picking out the long-term solution and ignoring problematic short-term fluctuations, although it may introduce unacceptable inaccuracy into the solution.

Similar ideas apply to the diffusion equation. The explicit form

$$u_j^{n+1} = u_j^n + \alpha(u_{j+1}^n - 2u_j^n + u_{j-1}^n)\,,$$

is prone to instability unless the time step is made inconveniently small. The fully implicit version of the differencing scheme is

$$u_j^{n+1} = u_j^n + \alpha(u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1})\,,$$

i.e. all the superscripts in the "∇^2" term are now $n+1$. Application of the von Neumann analysis to this system yields the following result for the amplification factor, $\xi(k)$:

$$\xi(1 + 4\alpha \sin^2 \tfrac{1}{2}k\Delta x) = 1,$$

that is,

$$\xi = (1 + 4\alpha \sin^2 \tfrac{1}{2}k\Delta x)^{-1}.$$

(Compare Equation 2 in the von Neumann analysis of this problem.) Note that in this case $|\xi| < 1$ always, so the method is *unconditionally stable*.

To implement the method, rewrite it as

$$-\alpha u_{j+1}^{n+1} + (1+2\alpha)u_j^{n+1} - \alpha u_{j-1}^{n+1} = u_j^n.$$

If we regard the u_j^n as known quantities and the u_j^{n+1} as an unknown vector of length J, then we can view the above rule as a matrix equation. setting $x_j = u_j^{n+1}$ and denoting the right-hand side of the equation (u_j^n) by r_j, we can write

$$\mathbf{A}\mathbf{x} = \mathbf{r}\,,$$

where the $J \times J$ matrix $\mathbf{A}$ is *tridiagonal* (connecting j with $j \pm 1$) and has the form

$$\mathbf{A} = \begin{pmatrix} ? & ? & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ -\alpha & 1+2\alpha & -\alpha & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & -\alpha & 1+2\alpha & -\alpha & 0 & \cdots & 0 & 0 & 0 \\ . & . & . & . & . & & . & . & . \\ . & . & . & . & . & & . & . & . \\ . & . & . & . & . & & . & . & . \\ 0 & 0 & 0 & 0 & 0 & \cdots & -\alpha & 1+2\alpha & -\alpha \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & ? & ? \end{pmatrix}.$$

As before, the above equation refers only to the interior points of the grid, so the top and bottom rows of the matrix are as yet undefined.

Such tridiagonal systems are easily solved. It is conventional to represent the nonzero elements of the matrix as three vectors: $\mathbf{b}$ is the diagonal, $\mathbf{a}$ and $\mathbf{c}$ are the off-diagonal elements, i.e.

$$\mathbf{A} = \begin{pmatrix} b_0 & c_0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ a_1 & b_1 & c_1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & a_2 & b_2 & c_2 & 0 & \cdots & 0 & 0 & 0 \\ . & . & . & . & . & & . & . & . \\ . & . & . & . & . & & . & . & . \\ . & . & . & . & . & & . & . & . \\ 0 & 0 & 0 & 0 & 0 & \cdots & a_{J-2} & b_{J-2} & c_{J-2} \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & a_{J-1} & b_{J-1} \end{pmatrix}.$$

Hence, for $1 \le j \le J-2$, we have $a_j = c_j = -\alpha$, $b_j = 1+2\alpha$, and $r_j = u_j^n$. Note that a_0 and c_{J-1} are not used. We use the first (b_0, c_0, and r_0) and last (a_{J-1}, b_{J-1}, and r_{J-1}) rows of the matrix equation to apply the *boundary conditions* of the problem. For example, to set $x_0 = 0$, we could choose $b_0 = 1$, $c_0 = 0$, $r_0 = 0$. To set $x_{J-2} = x_{J-1}$, we would set $a_{J-1} = -1$, $b_{J-1} = 1$, $r_{J-1} = 0$, and so on. In this case the vectors are particularly simple, as their elements are constant.

We can now rewrite our previous integrator to use the fully implicit scheme — Exercise 9.2. Each step now entails solving a matrix equation, but apart from that the logic of the program is unchanged.