

Mat1062: Computational Methods for PDE

Mary Pugh

January 10, 2008

1 Ownership

These notes are built upon those of Rob Almgren who taught an analogous course in 2003. Whatever you learn of value from them is due to him. All mistakes and sources of confusion are to be blamed on me.

2 Boundary Conditions

We now want to discuss in detail methods for solving the linear diffusion equation for $u(x, t)$

$$u_t = D u_{xx} \tag{1}$$

We shall suppose that the domain is $X_L \leq x \leq X_R$, and that initial data $u_0(x)$ is given at $t = 0$:

$$u(x, 0) = u_0(x), \quad X_L \leq x \leq X_R.$$

and that suitable boundary conditions are given on $x = X_L$ and $x = X_R$ for $t > 0$. We will consider boundary conditions that are *Dirichlet*, *Neumann*, or *Robin*. Dirichlet boundary conditions specify the value of u at the endpoints:

$$u(X_L, t) = u_L(t), \quad u(X_R, t) = u_R(t)$$

where u_L and u_R are specified functions of time. Neumann conditions specify the derivative u_x at the endpoints:

$$u_x(X_L, t) = u_L(t), \quad u_x(X_R, t) = u_R(t).$$

Robin boundary conditions specify a linear combination of u and u_x at the endpoints:

$$a u(X_L, t) + b u_x(X_L, t) = u_L(t), \quad c u(X_R, t) + d u_x(X_R, t) = u_R(t)$$

where a , b , c , and d are fixed constants usually determined by material properties.

3 Space Discretization

For given X_L and X_R , we choose a number N . Our grid spacing is

$$h = \frac{X_R - X_L}{N}$$

and we place grid points at locations

$$x_j = X_L + jh, \quad j = 0, 1, \dots, N,$$

so that $x_0 = X_L$ and $x_N = X_R$. We represent our solution by a vector $U(t)$ of length $N + 1$, whose components are

$$U_j(t) \approx u(x_j, t), \quad j = 0, \dots, N.$$

That is, the j th component of the vector, $U_j(t)$ is supposed to approximate the PDE's solution at the x_j grid point: $u(x_j, t)$.

Ideally, we would like to choose X_L , X_R , N , and h , so that important points in the solution fall exactly on grid points. For example, if the initial data has a corner or a discontinuity, our pictures will look better if there is a grid point right there (though the solution won't be much affected).

We take the initial data to be an exact sample of the true initial data:

$$U_j(0) = u_0(x_j).$$

The idea is that we will approximate the spatial derivative operator on the right side of (1) by a linear operator L (that is, a matrix), operating on the vector U . Then we define $U(t)$ to satisfy a linear ODE $U_t = LU$, which we can solve by standard methods. Inhomogeneous boundary conditions introduce inhomogeneous terms into this system, as we discuss below.

3.1 Dirichlet Boundary Conditions

Let us consider the case of Dirichlet boundary data. Then the end values $U_0(t)$ and $U_N(t)$ are specified by the boundary value functions and so there are only $N - 1$ unknowns: $U_1(t) \dots U_{N-1}(t)$.

We can write the differentiation operation from last time as

$$\begin{pmatrix} u_{xx}(x_1) \\ \vdots \\ u_{xx}(x_{N-1}) \end{pmatrix} \sim \frac{1}{h^2} \begin{pmatrix} 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2 & 1 \\ & & & & 1 & -2 & 1 \end{pmatrix} \begin{pmatrix} U_0 \\ U_1 \\ \vdots \\ U_{N-1} \\ U_N \end{pmatrix}.$$

However, these expressions are not quite what we want. The vector on the left is of size $N - 1$ and will determine the interior grid values. The vector on the right is of size $N + 1$; it involves both the interior grid values and the boundary values. We need to express derivatives at the interior grid points in terms only of interior grid values and the Dirichlet boundary conditions. This means that we need to eliminate U_0 and U_N from the above.

Substituting the boundary values $u_0 = u_L(t)$, $u_N = u_R(t)$, we can write the derivative as the *inhomogeneous* linear operator

$$D_2U = M_2U + R_2,$$

where M_2 is a $(N-1) \times (N-1)$ tridiagonal matrix and R_2 is an $(N-1)$ -vector containing the boundary data:

$$M_2 = \frac{1}{h^2} \begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{pmatrix}, \quad R_2(t) = \frac{1}{h^2} \begin{pmatrix} u_L(t) \\ 0 \\ \vdots \\ 0 \\ u_R(t) \end{pmatrix}$$

R_2 depends on time through the boundary values. In practice, it is often more convenient to store the boundary values in their natural locations at the endpoints of a linear grid, and then the derivatives can be computed by doing the $(N - 1) \times (N + 1)$ multiplication above.

Of course, this matrix never actually exists in the computer, nor do we ever actually do a matrix multiply as written above: since most entries are

means that we need to use the Neumann boundary conditions to eliminate the values at the ghost points. This is done as follows:

$$u_L(t) = u_x(X_L, t) \sim \frac{u_1 - u_{-1}}{2h} \implies u_{-1} = u_1 - 2hu_L(t)$$

as a result

$$u_{xx}(X_L, t) \sim \frac{u_1 - 2u_0 + u_{-1}}{h^2} = \frac{2u_1 - 2u_0}{h^2} - \frac{2u_L(t)}{h}.$$

Similarly, at the right-hand endpoint

$$0 = u_x(X_R, t) \sim \frac{u_{N+1} - u_{N-1}}{2h} \implies u_{N+1} = u_{N-1} + 2hu_R(t)$$

hence

$$u_{xx}(X_R, t) \sim \frac{u_{N+1} - 2u_N + u_{N-1}}{h^2} = \frac{2u_{N-1} - 2u_N}{h^2} + \frac{2u_R(t)}{h}.$$

And so, we can write the derivative as the *inhomogeneous* linear operator

$$D_2\mathbf{U} = M_2\mathbf{U} + \mathbf{R}_2,$$

where M_2 is a $(N+1) \times (N+1)$ tridiagonal matrix and \mathbf{R}_2 is an $N+1$ -vector containing the boundary data:

$$M_2 = \frac{1}{h^2} \begin{pmatrix} -2 & 2 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 2 & -2 \end{pmatrix}, \quad \mathbf{R}_2(t) = \frac{2}{h} \begin{pmatrix} -u_L(t) \\ 0 \\ \vdots \\ 0 \\ u_R(t) \end{pmatrix}$$

A natural question to ask is why we used centered finite differences above to approximate $u_x(X_L)$ and $u_x(X_R)$ rather than using a left-difference for $u_x(X_L)$ and a right-difference for $u_x(X_R)$:

$$0 = u_x(X_L) \sim \frac{u_0 - u_{-1}}{h} \quad 0 = u_x(X_R) \sim \frac{u_{N+1} - u_N}{h}$$

We will revisit this question when we discuss the convergence and accuracy of our scheme.

3.3 Robin Boundary Conditions

We handle Robin boundary conditions in the same manner that we handle Neumann boundary conditions: by introducing ghost grid points the values at which are then determined from the boundary conditions.

$$\begin{aligned} u_L(t) &= a u(X_L, t) + b u_x(X_L, t) \sim a U_0 + b \frac{U_1 - U_{-1}}{2h} \\ \implies U_{-1} &= U_1 + \frac{2ah}{b} U_0 - \frac{2hu_L(t)}{b} \end{aligned}$$

as a result

$$u_{xx}(X_L, t) \sim \frac{U_1 - 2U_0 + U_{-1}}{h^2} = \frac{2U_1 - (2 - 2ah/b)U_0}{h^2} - \frac{2u_L(t)}{bh}.$$

Similarly,

$$\begin{aligned} u_R(t) &= c u(X_R, t) + d u_x(X_R, t) \sim c U_N + d \frac{U_{N+1} - U_{N-1}}{2h} \\ \implies U_{N+1} &= U_{N-1} - \frac{2ch}{d} U_N + \frac{2hu_R(t)}{d} \end{aligned}$$

hence

$$u_{xx}(X_L, t) \sim \frac{U_1 - 2U_0 + U_{-1}}{h^2} = \frac{2U_{N-1} - (2 + 2ch/d)U_N}{h^2} + \frac{2u_R(t)}{dh}.$$

In this way, we can write the derivative as the *inhomogeneous* linear operator

$$D_2U = M_2U$$

where M_2 is a $(N + 1) \times (N + 1)$ tridiagonal matrix:

$$M_2 = \frac{1}{h^2} \begin{pmatrix} -2 + 2ah/b & 2 & & & & & \\ & 1 & -2 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 1 & -2 & 1 & \\ & & & & 2 & -2 - 2ch/d & \end{pmatrix}, \quad R_2(t) = \frac{2}{h} \begin{pmatrix} -u_L(t)/b \\ 0 \\ \vdots \\ 0 \\ u_R(t)/d \end{pmatrix}.$$

As a reality check, note that if $a = c = 0$ and if $b = d = 1$ then this matrix is the same matrix as for the Neumann boundary conditions.

4 Time discretization

For Dirichlet boundary conditions, we have now defined an evolution problem for a vector $U(t)$ of length $N - 1$:

$$\frac{dU}{dt} = LU = MU + R(t), \quad U(0) \text{ given.} \quad (3)$$

For Neumann and Robin boundary conditions, we have an evolution problem with the exact same structure except the vector $U(t)$ has length $N + 1$. The above notation is slightly confusing, since L is *not a matrix*: we use it to denote this “affine linear” operation.

To solve this system of ODEs, we choose a time step k , define time levels $t_n = nk$ for $n = 0, 1, \dots$, define $U^n \approx U(t_n)$,¹ and give a rule for computing U^{n+1} in terms of U^n , and possibly time levels earlier than n , for $n = 0, 1, \dots$ (We use a *superscript* n to denote time levels.)

If we want the solution value at a specific time $t = T$, it will be convenient if $T = t_M = Mk$ for some integer M ; if not, then $Mk < T < (M + 1)k$ for some M and we will do an interpolation in time between U^M and U^{M+1} .

There is an immense variety of schemes for numerical “integration” (solution) of ordinary differential equations, often with very high order accuracy and various other desirable properties (Runge-Kutta methods, for example). When the ODE system arises from discretization of a PDE, the major errors usually do not come from the time integration. It is most important to have a scheme that is simple, efficient, and especially *stable*, as we discuss below.

Our three basic methods are:

$$\begin{array}{ll} \text{Forward Euler} & \frac{1}{k}(U^{n+1} - U^n) \sim U_t(nk) = LU^n \\ \text{Crank Nicolson} & \frac{1}{k}(U^{n+1} - U^n) \sim U_t((n + 1/2)k) \sim \frac{1}{2}LU^n + \frac{1}{2}LU^{n+1} \\ \text{Backward Euler} & \frac{1}{k}(U^{n+1} - U^n) \sim U_t((n + 1)k) = LU^{n+1} \end{array}$$

All our formulas can be summarized by the expression

$$\frac{1}{k}(U^{n+1} - U^n) = \theta LU^{n+1} + (1 - \theta) LU^n$$

¹We are not being quite consistent in the notation. We introduced capital U to denote the space-discrete values $U_j(t)$, and in principle we should introduce a new symbol to denote the time-discrete values.

where θ with $0 \leq \theta \leq 1$ is an *implicitness parameter*. In the three cases of most interest, we have special names for the methods:

$$\frac{1}{k}(U^{n+1} - U^n) = \begin{cases} LU^n, & \text{Forward Euler } (\theta = 0) \\ \frac{1}{2}(LU^{n+1} + LU^n), & \text{Crank-Nicolson } (\theta = \frac{1}{2}) \\ LU^{n+1}, & \text{Backward Euler } (\theta = 1). \end{cases}$$

As you may guess from the terminology, the forward Euler method is a *fully explicit* method, and the backward Euler method is sometimes called the *fully implicit* method. (There are a large variety of fully explicit methods, most notably “linear multistep” methods of which second-order Adams-Bashforth is occasionally useful for PDEs.)

When we substitute the expression for L , we obtain the formula for U^{n+1} :

$$(I - \theta kM)U^{n+1} = (I + (1 - \theta)kM)U^n + k(\theta R^{n+1} + (1 - \theta)R^n). \quad (4)$$

Note that this formula tells us exactly at what times we should evaluate the time-dependent boundary conditions. The actual computations we do are very different, depending on whether θ is zero or not.

- If $\theta = 0$ (forward Euler method), then this is the explicit formula

$$U^{n+1} = U^n + k(MU^n + R^n).$$

This says, just compute the time derivative in the current configuration, and advance U as though you moved at that speed for the entire time k . Only a matrix multiplication is required; but since the matrix is tridiagonal it is trivial: U_j^{n+1} is given by a difference formula in terms of U_{j-1}^n , U_j^n , and U_{j+1}^n .

- If $\theta > 0$, then we must solve a linear system to get U^{n+1} . In this problem, with only one space dimension, this system is tridiagonal. We can thus obtain the solution in time $\mathcal{O}(N)$ by a simplified version of Gaussian elimination; pivoting is not usually necessary for problems arising from diffusive PDEs. For problems in more space dimensions, this linear solve can become the most time-consuming part of the whole computation. Note that each component U_j^{n+1} depends on *all* components U_j^n at the previous level.