M.Eng. 2.6 Mathematics: Iterative Methods for Elliptic PDEs

This sheet can be found on the Web: http://www.ma.ic.ac.uk/~ajm8/MEng26

We have seen how to solve **Parabolic** PDEs on a computer by time-stepping methods. Similar techniques can be used for **Hyperbolic** PDEs but they are not part of the syllabus. **Elliptic** PDEs are different, however, as there is no 'time-like' variable. Instead the solution must be found everywhere at once.

Consider the two-dimensional Poisson equation for $u(x,\, y)$ in a rectangle

$$\nabla^2 u \equiv u_{xx} + u_{yy} = f(x,\, y) \qquad \text{in} \quad 0 < x < a \quad 0 < y < b \tag{1}$$

with u known over the rectangular boundary. We divide the rectangle into MN $h \times k$ rectangles, where $Mh = a$ and $Nk = b$. We then seek an approximation to $u_{mn} \equiv u(mh,\, nk)$ for $m = 1, \ldots M-1$ and $n = 1, \ldots N-1$. The values u_{0n}, u_{Mn}, u_{m0} and u_{mM} are known from the boundary conditions and so of course we set $U_{0n} = u_{0n}$ etc.

As before, using Taylor series we can derive the formula

$$\left(u_{xx}\right)_{mn} = \frac{u_{m+1\,n} - 2u_{mn} + u_{m-1\,n}}{h^2} + O(h^2)$$

and similarly

$$\left(u_{yy}\right)_{mn} = \frac{u_{m\,n+1} - 2u_{mn} + u_{m\,n-1}}{k^2} + O(k^2).$$

Combining these, we can define an approximation U_{mn} to u_{mn} as the solution to

$$\frac{1}{h^2}\left(U_{m+1\,n} + U_{m-1\,n}\right) + \frac{1}{k^2}\left(U_{m\,n+1} + U_{m\,n-1}\right) - \left(\frac{2}{h^2} + \frac{2}{k^2}\right) U_{mn} = f_{mn}\ , \tag{2}$$

where $f_{mn} \equiv f(mh,\, nh)$. Equation (2) holds for $m = 1, \ldots M-1$ and $n = 1, \ldots N-1$ and so defines $(M-1) \times (N-1)$ equations in total. Furthermore, each equation is linear in the $(M-1) \times (N-1)$ unknowns U_{mn} (recall that U_{0n}, U_{Mn}, U_{m0} and U_{mM} are known.) In general, therefore, solving (1) is equivalent to solving a large number of simultaneous linear equations, which we can write as $A\mathbf{v} = \mathbf{b}$, where A is a square matrix, $\mathbf{v}$ and $\mathbf{b}$ are unknown and known vectors, respectively. We could order the unknowns by scanning the y-direction for fixed x, for example, so that

$$\mathbf{v} = \left(U_{11}, U_{12}, \ldots U_{1\,N-1}, U_{21}, U_{22} \ldots U_{M-2\,N-1}, U_{M-1\,1} \ldots U_{M-1\,N-1}\right)^T .$$

The vector $\mathbf{b}$ is determined by f_{mn} and the boundary conditions.

From now on, for simplicity, we will assume $h = k$. Then (2) can be rewritten as

$$U_{m+1\,n} + U_{m-1\,n} + U_{m\,n+1} + U_{m\,n-1} - 4U_{mn} = h^2 f_{mn}\ . \tag{3}$$

This is often known as the 5-point formula for the Laplacian.

Jacobi and Gauss-Seidel Iteration

The matrix A is very large, and very **sparse**: almost all its entries are zero. In each row of A there is a –4 on the diagonal, at most four entries of 1 while the rest are zero. We could use **Gaussian elimination** to solve $A\mathbf{v} = \mathbf{b}$, but this tends to 'fill in' many of the zeros. A very important idea is to find the solution iteratively.

Jacobi iteration: The method is as follows: First, rewrite the equations in the form U_{mn} = something. Then make some initial guess $U^{(0)}_{mn}$ of the solution for all m and n. Then use (3) to provide a new estimate $U^{(j+1)}_{mn}$ from the old estimate $U^{(j)}_{mn}$ according to:

$$U^{(j+1)}_{mn} = \tfrac{1}{4}\left[U^{(j)}_{m-1\,n} + U^{(j)}_{m+1\,n} + U^{(j)}_{m\,n-1} + U^{(j)}_{m\,n+1} - h^2 f_{mn}\right] \qquad \text{for} \quad j = 0, 1 \ldots \tag{4}$$

Each iteration, the entire grid is scanned using the old estimates on the right hand side. If the new estimate is the same as the old everywhere, then that must be the solution to (3).

Gauss-Seidel iteration: This is similar to Jacobi iteration, except that we scan the grid in a definite order, using the new estimates on the right hand side as soon as they become available. In a practical computer program, we might overwrite the old values with the new ones. If we scan the grid in the same order as the elements of $\mathbf{v}$ overleaf, the formula can be written

$$U^{(j+1)}_{mn} = \tfrac{1}{4}\left[U^{(j+1)}_{m-1\,n} + U^{(j)}_{m+1\,n} + U^{(j+1)}_{m\,n-1} + U^{(j)}_{m\,n+1} - h^2 f_{mn}\right] . \tag{5}$$

For this problem, it can be shown that both the Jacobi and Gauss-Seidel methods work, and that the latter converges faster and so is usually preferable. However, if you are working on a parallel processing machine, the Jacobi algorithm can be more appropriate.

Why not just use the exact formula for the inverse matrix?

You should have met the formula for an inverse matrix, in terms of the matrix of cofactors, or adjoints. So why not use a fast computer to calculate $\mathbf{v} = A^{-1}\mathbf{b}$? Suppose we have a Cray capable of 10^9 multiplications per second, and have a mere 20×20 matrix to invert. (In practice A may be $10^4 \times 10^4$). The cofactor of an element of an $n \times n$ matrix is essentially an $(n-1) \times (n-1)$ determinant. Suppose T_n multiplications are needed to calculate an $n \times n$ determinant. Clearly $T_2 = 2$. If we define the $n \times n$ determinant by expanding it along a row, then we need to calculate n smaller $(n-1)\times(n-1)$ determinants. Thus approximately

$$T_n = nT_{n-1} = n(n-1)T_{n-2} = n(n-1)(n-2)\ldots(3)(2) = n! \; .$$

Using the formula to invert an $n \times n$ matrix A requires calculating n^2 cofactors and then the determinant of A. So proceeding naively, we can find the inverse matrix A^{-1} using

$$\text{Total multiplications} = n^2 T_{n-1} + T_n = n^2(n-1)! + n! = (n+1)! \; .$$

Thus a Cray would take $21! \times 10^{-9}$ seconds to invert a 20×20 matrix. This is about 1619 years! Admittedly, the method used was very crude, but this demonstrates the point of thinking about the algorithm. An iterative method would take about a microsecond.