Lecture 10-11: Iterative methods for Elliptic problems

Last lecture we reduced the Poisson equation in the unit square

$$\nabla^2 u \equiv u_{xx} + u_{yy} = f \qquad \text{in } 0 < x < 1,\, 0 < y < 1, \tag{10.1}$$

with Dirichlet conditions $u = 0$ on the 4 sides of the square to the set of $M \equiv (N-1)(N-1)$ simultaneous algebraic equations

$$A\mathbf{U} = \mathbf{b}, \qquad \text{where } A \text{ is an } M \times M \text{ sparse matrix,} \tag{10.2}$$

and $\mathbf{U}$ was an M-vector of the unknowns,

$$\mathbf{U} = \big(U_{11}, U_{12} \ldots U_{1N-1}, U_{21} \ldots U_{N-2,N-1}, U_{N-1\,1} \ldots U_{N-1N-1}\big). \tag{10.3}$$

For this problem, on a square grid A can be written in block form

$$A = \left(\begin{array}{ccccccc} T & | & I & \vdots & O & | & O \\ --- & + & --- & + & --- & + & --- \\ I & | & T & \vdots & \ddots & | & O \\ \cdots\cdots & + & \cdots\cdots & + & \cdots\cdots & + & \cdots\cdots \\ O & \vdots & \ddots & \vdots & \ddots & | & I \\ --- & + & --- & + & --- & + & --- \\ O & | & O & \vdots & I & | & T \end{array}\right) \qquad T = \begin{pmatrix} -4 & 1 & 0 & 0 \\ 1 & -4 & \ddots & 0 \\ 0 & \ddots & \ddots & 1 \\ 0 & 0 & 1 & -4 \end{pmatrix}. \tag{10.4}$$

If you ever wish to construct such a block matrix in Matlab, the *kron* function is useful. Usually, however, we will not wish to deal with the matrices directly. The critical point is that the matrix A is **sparse** – almost all its entries are zero. Yet because of its penta-diagonal structure, its inverse is full. This renders direct solution methods unattractive, requiring large amounts of storage and CPU. In contrast, iterative methods require very little storage or time *per iteration.* What we have to ascertain, is whether they converge, and how quickly. If we need too many iterations, there way be no gain over Gaussian elimination (or LU decomposition) requiring $O(M^3) = O(N-1)^6$ operations.

An iterative scheme essentially splits the matrix A into two components, $A = B + C$, where B is chosen to be easy to invert. It then solves

$$B\mathbf{U}^{j+1} = \mathbf{b} - C\mathbf{U}^j \qquad \text{or} \quad \mathbf{U}^{j+1} = B^{-1}\mathbf{b} - (B^{-1}C)\mathbf{U}^j. \tag{10.5}$$

If we introduce the error vector $\mathbf{Z}^j = \mathbf{U}^j - \mathbf{U}$, where $\mathbf{U}$ is the exact solution to (10.2) then

$$\mathbf{Z}^{j+1} = (B^{-1}C)\mathbf{Z}^j. \tag{10.6}$$

In this form it is clear that fastest convergence occurs for small spectral radius $\rho(B^{-1}C)$.

The simplest iterative scheme essentially rakes $B = D$, where D is a diagonal part of A and C has zeros on its diagonal. We then defined the iterative **Jacobi** scheme

$$\mathbf{U}^{j+1} = D^{-1}(\mathbf{b} - C\mathbf{U}^j) \quad \text{or} \quad U_{mn}^{j+1} = \tfrac{1}{4}\Big[U_{mn+1}^j + U_{mn-1}^j + U_{m+1n}^j + U_{m-1n}^j - h^2 f_{mn}\Big]. \tag{10.7}$$

This calculates all the new iterations at level $j+1$ before updating U. We also considered the **Gauss-Seidel** scheme, which uses the new calculated values as soon as they become available. Assuming m and n are scanned in an increasing direction, this scheme can be written as

$$U_{mn}^{j+1} = \tfrac{1}{4}\Big[U_{mn+1}^j + U_{mn-1}^{j+1} + U_{m+1n}^j + U_{m-1n}^{j+1} - h^2 f_{mn}\Big]. \tag{10.8}$$

This essentially involves back-substitution of the calculated values. In terms of the matrix, $B = D + L$ where L is the lower triangular part of A. The spectral radii of the Jacobi & Gauss-Seidel matrices can be found. In fact, for large $M \equiv (N-1)^2$ it can be shown that

$$\rho_J = \cos\left(\frac{\pi}{N}\right) \simeq 1 - \frac{\pi^2}{2N^2}, \qquad \rho_G = \rho_J^2 \simeq 1 - \frac{\pi^2}{N^2}. \tag{10.9}$$

This agrees with the factor of 4 error reduction we found using the human computer last time. We can therefore estimate how many iterations are needed to obtain p figures of accuracy. For the error to be reduced by a factor of 10^p. If $\rho^\alpha = 10^{-p}$ then

$$\alpha = \frac{p\log 10}{-\log\rho} \simeq 0.467pN^2 \quad \text{or} \quad 0.233pN^2$$

for Jacobi or Gauss-Seidel. This is disappointingly large for large N, but faster than direct methods. Can we do better somehow?

We may represent the scheme in terms of the **residual** vector $\mathbf{r}^j = \mathbf{b} - A\mathbf{U}^j$, which is the amount by which the j^{th} estimate fails to satisfy the equation. We then have that

$$\mathbf{U}^{j+1} = \mathbf{U}^j + B^{-1}\mathbf{r}^j \qquad \text{or} \quad \mathbf{r}^{j+1} = CB^{-1}\mathbf{r}^j.$$

If we assume that we are altering the estimate in a good direction, we might try to take larger steps, and choose a parameter $\omega > 1$ and define the **overrelaxation** scheme

$$\mathbf{U}^{j+1} = \mathbf{U}^j + \omega B^{-1}\mathbf{r}^j. \tag{10.10}$$

In terms of the code we modify (10.7) to read

$$U_{mn}^{j+1} = (1-\omega)U_{mn}^j + \frac{\omega}{4}\Big[U_{mn+1}^j + U_{mn-1}^{j+1} + U_{m+1n}^j + U_{m-1n}^{j+1} - h^2 f_{mn}\Big]. \tag{10.11}$$

We can then investigate which values of ω give best behaviour. This is a very powerful idea in general. Even if we start with an unstable scheme, it may become stable by choosing a small value of ω. For a stable scheme, choosing $\omega > 1$ probably will speed up the convergence.

In general the optimum ω varies with N. But the optimal ω brings a very marked improvement to $\rho \simeq 1 - 2\pi/N$ and $\alpha \simeq 0.37pN$. The **Successive Over-Relaxation (SOR)** method (10.11) requires $O(N)$ iterations not $O(N^2)$. But can we do better still?

Let's investigate all this on a computer.