11 Elliptic PDEs : Computer Solution by Iteration

Consider the elliptic PDE in Cartesian (x, y) coordinates :

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0. \tag{11.1}$$

A simple centered finite-difference discretisation, with step sizes in both $(x.y)$-directions given by $\delta x = \delta y = 1$, with boundary values specified on the outer closed domain of a "*toy-problem*" is shown in figure 11.1. A general discretisation stencil for the unknowns in the interior is given by

$$2(1 + [\frac{\delta y}{\delta x}]^2)U_{i,j} = [\frac{\delta y}{\delta x}]^2(U_{i+1,j} + U_{i-1,j}) + U_{i,j+1} + U_{i,j-1} \tag{11.2}$$

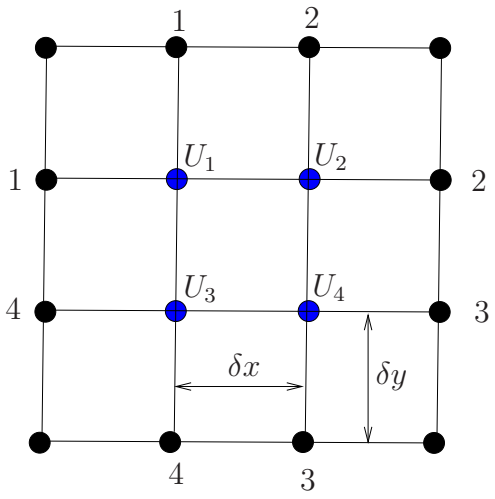

Figure 11.1: Discretised domain - Toy problem

Application of Eqn. (11.2), on each of the interior unknown points (U_1, U_2, U_3, U_4) in figure 11.1, results in a system of 4 coupled systems of equations, which may be written in matrix form as

$$[\boldsymbol{A}]\ \bar{\boldsymbol{U}} = \boldsymbol{b} \tag{11.3}$$

i.e.

$$\begin{pmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{pmatrix} \begin{pmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 8 \\ 6 \end{pmatrix}. \tag{11.4}$$

This may be easily solved by inverting the matrix $[\boldsymbol{A}]$, by use of Gaussian Ellimination, to give the following values :

$$U_1 = 1.833333333333333\,; \quad U_2 = 2.166666666666667\,;$$
$$U_3 = 3.166666666666667\,; \quad U_4 = 2.833333333333333\,.$$

Solution by the Gaussian elimination technique is called the **direct** approach.

11.1 Solution by Iteration

Let us try another technique, which does not require Gaussian elimination but a simple **iteration** technique.

We use the computer (or a calculator!) to solve this by an **iterative** approach, demonstrating the basic idea on a very idealised 4×4 grid with appropriate conditions enforced at the boundaries.

Four unknowns, $U_1 \;\; U_2 \;\; U_3 \;\; U_4$ which are all set, with an initial guess (or trial solution) to have values of zero. Define the next iterate by

$$\begin{aligned} 4y_1 &= U_2 + U_3 + 2 \qquad & 4y_2 &= U_1 + U_4 + 4 \\ 4y_3 &= U_1 + U_4 + 8 \qquad & 4y_4 &= U_2 + U_3 + 6 \end{aligned} \tag{11.5}$$

Then set $U_1 = y_1$ etc and repeat. The solution (corrections) proceeds as given in Table 1. Note the errors, by this so-called **Jacobi** iteration halve after each iteration.

Table 1: Solution of linear equations by Jacobi iteration

Iteration	U_1	U_2	U_3	U_4
0	0	0	0	0
1	0.5	1.0	2.0	1.5
2	1.25	1.5	2.5	2.25
3	1.5	1.875	2.875	2.5
4	1.6875	2.0	3.0	2.6875
5	1.75	2.09375	3.09375	2.75
6	1.796875	2.125	3.125	2.796875
∞	1.83333333	2.166666667	3.166666667	2.83333333

Now do the same thing but use the new iterates as soon as they become available – known as the **Gauss-Seidel (G-S)** method, *i.e.*,

$$\begin{aligned} 4U_1 &= U_2 + U_3 + 2 \qquad & 4U_2 &= U_1 + U_4 + 4 \\ 4U_3 &= U_1 + U_4 + 8 \qquad & 4U_4 &= U_2 + U_3 + 6 \end{aligned} \tag{11.6}$$

How the G-S corrections proceed is shown in Table 2.

Figure 11.2 shows the overall convergence of the solutions as the iterations proceed. Note with the G-S method, the solution has converged to an accuracy of about 7 decimal places after 14 iterations; contrast this with the Jacobi method where only an accuracy of 3 decimals is attained after about 14 iterations. Note the errors with G-S decrease by a factor of 4 after each iteration. Usually a log plot shows up convergence behaviour much better, since to graphical accuracy, looking at the first plot one would assume that both solutions were pretty similar by about 8 iterations – a log plot usually gives much clearer indications of convergence and errors.

Table 2: Solution of linear equations by Gauss-Seidel iteration

Iteration	U_1	U_2	U_3	U_4
0	0	0	0	0
1	0.5	1.125	2.125	2.3125
2	1.3125	1.90625	2.90625	2.703125
3	1.703125	2.1015625	3.1015625	2.80078125
4	1.80078125	2.150390625	3.150390625	2.825195312
5	1.825195313	2.162597656	3.162597656	2.831298828
6	1.831298828	2.165649414	3.165649414	2.832824707
∞	1.833333333	2.166666667	3.166666667	2.833333333

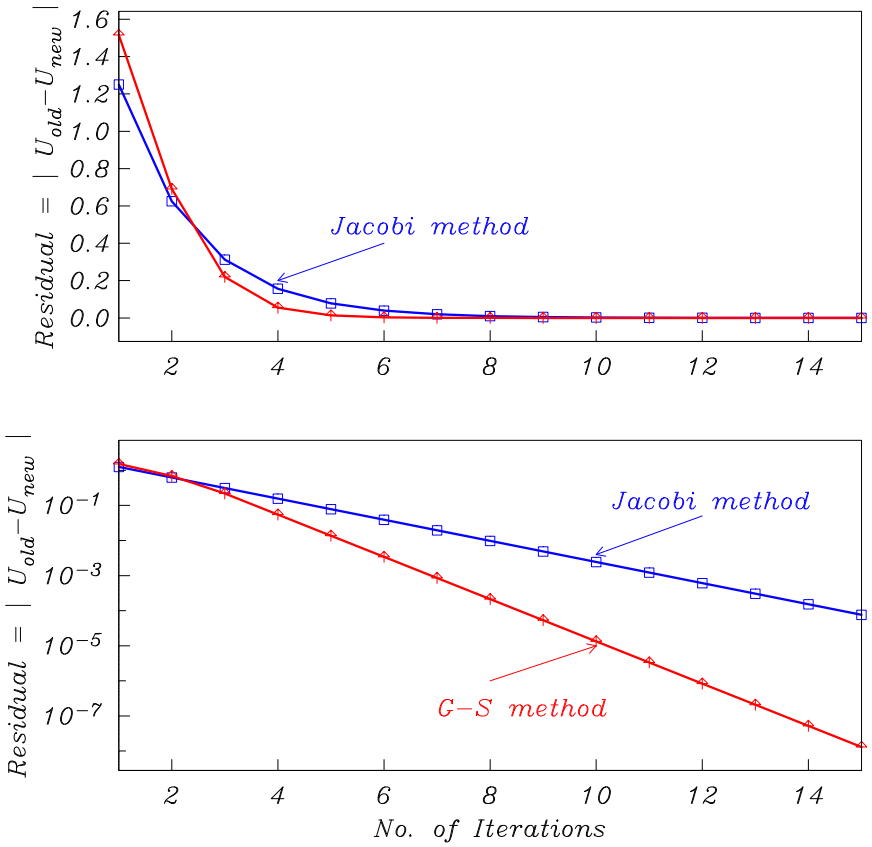

Figure 11.2: Residuals of the solution, as the iterations proceed, comparing Jacobi and Gauss-Seidel (G-S) methods.

Above we have solved a quite trivial problem and used quite a simple means to establish that our solution has converged, namely by monitoring how the value of each unknown behaves and whether they approach constant values as the iterations proceed. With just 4 unknowns this is quite adequate, however in more complex problems where we may be dealing with perhaps many hundreds, thousands or millions of unknowns, more sophisticated computer automated monitoring of the solutions and convergence criteria needs to be programmed for. Sometimes (or usually !) due to a bug in your program or for some other reason convergence to a solution does not occur, in which circumstances you may find your program to be in an endless loop – a well written computer program monitors and usually has traps programmed to terminate computations if convergence or the *residuals*

do not appear to be decreasing during the iterations.

In figure 11.2 how do you think we have defined the residuals? Are we monitoring just one of the U_i solution points, some or all of them? What would be the best means to assess a converged solution?

11.2 Direct versus iterative techniques

We have shown above that finite-difference discretisation for solving a boundary value problem (BVP) leads to a system of algebraic simultaneous equations. In most practical cases, even say when solving a nonlinear BVP, the solution discretisation procedure usually still involves solving linear systems of coupled equations. Generally their number is large possibly running into tens of thousands or even millions of unknowns, and thus their solution is a major problem – observe in our simple example above we only had 4 unknowns, see Eqn. (11.3).

We also showed that there are two different ways of solving linear systems arising from elliptic differential equations. A direct method such as Gaussian elimination produces an exact solution in a finite number of operations. Gaussian elimination solves the system of equations in a known number of arithmetic operations, and any errors in the solution arise solely from rounding errors arising from the computer.

An iterative method, starts with an initial guess for the solution and attempts to improve it through some correction procedure – the iterations are repeated and corrected values monitored to ascertain a possible convergence to an unchanging solution with increasing number of iterations. Usually the user specifies a convergence criterion to stop the iterations once a sufficiently good approximation has been obtained.

Direct inversion of the matrix $[\boldsymbol{A}]$ in Eqn. (11.3) will always work (provided $[\boldsymbol{A}]$ can be inverted), while iterative techniques are not always guaranteed to work (see later). However there are many class of problems which can be discretised where $[\boldsymbol{A}]$ has the form of a large *sparse* matrix and is *diagonally dominant*, under such conditions iterative methods are often preferable.

Finite-difference approximations often lead to sparse matrices, and, in what follows, we will concentrate on iterative techniques for solving linear systems resulting from discretisation of elliptic PDEs.

More details on the above and other aspects concerning solution of elliptic PDEs are in Chapters 3-4 of Leveque and Chapter 5 of G. D. Smith

12 Iterative Methods for Elliptic Problems

Last lecture we showed in a very simple way how the elliptic Poisson equation in the unit square, namely

$$\nabla^2 u \equiv u_{xx} + u_{yy} = f \quad \text{in } 0 \le x \le 1, \;\; 0 \le y \le 1, \tag{12.1}$$

with appropriate conditions on the 4-sided square boundaries (we had set $f = 0$ in our "*toy-problem*" example), could be reduced using finite-differences to requiring the solution of $M \equiv (P-2)(P-2) \equiv N \times N$ simultaneous algebraic equations (*i.e.* in the toy problem considered earlier $P = 4$, with $N = P - 2$) having the form

$$\boldsymbol{A}\boldsymbol{U} = \boldsymbol{b}, \;\; \text{where } \boldsymbol{A} \text{ is an } N \times N \text{ sparse matrix}, \tag{12.2}$$

and $\boldsymbol{U}$ was an N-vector of the unknowns,

$$\boldsymbol{U} = (U_{1,1}, U_{1,2}, \ldots, U_{1,N}\,; U_{2,1}, U_{2,2}, \ldots, U_{2,N}\,; U_{N,1}, U_{N,2}, \ldots, U_{N,N}). \tag{12.3}$$

For this problem on a square grid, $\boldsymbol{A}$ can be written in block form

$$\boldsymbol{A} = \left(\begin{array}{c|c:c|c} \mathcal{T} & \mathcal{I} & \mathcal{O} & \mathcal{O} \\ \hline \mathcal{I} & \mathcal{T} & \ddots & \mathcal{O} \\ \hline \mathcal{O} & \ddots & \ddots & \mathcal{I} \\ \hline \mathcal{O} & \mathcal{O} & \mathcal{I} & \mathcal{T} \end{array}\right); \quad \mathcal{T} = \begin{pmatrix} -4 & 1 & 0 & 0 \\ 1 & -4 & \ddots & 0 \\ 0 & \ddots & \ddots & 1 \\ 0 & 0 & 1 & -4 \end{pmatrix}.$$

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 $\boldsymbol{A}$ 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* operations. 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 may be no gain over Gaussian elimination (or Lower-Upper (LU) decomposition) requiring $O(P^3) = O(N)^6$ operations.

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

$$\boldsymbol{B}\boldsymbol{U}^{j+1} = \boldsymbol{b} - C\boldsymbol{U}^j \;\; \text{or} \;\; \boldsymbol{U}^{j+1} = \boldsymbol{B}^{-1}\boldsymbol{b} - (\boldsymbol{B}^{-1}C)\boldsymbol{U}^j. \tag{12.4}$$

If we introduce the error vector $\boldsymbol{Z}^j = \boldsymbol{U}^j - \mathcal{U}$, where $\mathcal{U}$ is the exact solution to (12.2) then

$$\boldsymbol{Z}^{j+1} = (\boldsymbol{B}^{-1}C)\boldsymbol{Z}^j. \tag{12.5}$$

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

*central processing unit

In the Jacobi-method we compute new values for U entirely from the previous iteration, as required from the equation. In the Gauss-Seidel-method we have already updated U_{mn-1} and U_{m-1n} before we update U_{mn} ; and these new values will be used instead of the old ones. The nice feature of both these iterative approaches is

- the matrix $\boldsymbol{A}$ is never stored, as would be the case with a direct method involving Gaussian elimination.
- Storage is optimal. Only N^2 values are stored for the Gauss-Seidel method, while $2N^2$ values are stored for the Jacobi method.
- Each iteration requires N^2 work. The total work depends on the number of iterations necessary to reach a desired level of accuracy.
- With direct methods, typically Gaussian elimination, it's running time is of $O(N^3)$, thus for very large N values iterative approaches are usually preferred – unless the form of the matrix $\boldsymbol{A}$ is such that the iterative approach is known to fail or is *ill-conditioned* – namely where the spectral radius $\rho(\boldsymbol{T}) \geq 1$ and strict diagonal dominance does not arise in the matrix.

Though we do not prove it in this course, the convergence of a linear iterative system is governed by the spectral radius or largest eigenvalue (in modulus) of the coefficient matrix. Thus, a fundamental stability criterion is that *a linear iterative system is asymptotically stable and convergent if and only if all its (complex) eigenvalues (λ) have modulus strictly less than one:* $|\lambda_j| < 1$. The spectral radius of a matrix $\boldsymbol{T}$ is defined as the maximal modulus of all of its real and complex eigenvalues: $\rho(\boldsymbol{A}) = max\{|\lambda_1|, \ldots, |\lambda_k|\}$. Unfortunately, finding accurate approximations to the eigenvalues of most matrices is a nontrivial computational task.

The spectral radius $\rho(\boldsymbol{T})$ of the coefficient matrix will govern the speed of convergence. Therefore, our main goal is to construct an iterative scheme whose coefficient matrix has as small a spectral radius as possible. At the very least, the spectral radius must be less than 1.

Completely general conditions guaranteeing convergence of the Gauss-Seidel and Jacobi methods are hard to establish. But both are guaranteed to converge when the original coefficient matrix $\boldsymbol{A}$ is strictly diagonally dominant, i.e.

$$|a_{i,1}|+|a_{i,2}|+\ldots+|a_{i,i-1}|+0+\ |a_{i,i+1}|+|a_{i,i+2}|+\ldots+|a_{i,m}| \leq\ |a_{i,i}|. \tag{12.6}$$

This states, that convergence is assured if in each row of $\boldsymbol{A}$ the modulus of the diagonal element exceeds the sum of the moduli of the off-diagonal elements.

12.1 Technical note on convergence

Consider the matrix

$$\boldsymbol{AU} = \boldsymbol{b}. \tag{12.7}$$

We next express that $\boldsymbol{A} = \boldsymbol{D} + \boldsymbol{C}$, where $\boldsymbol{D}$ represents the main diagonal elements and $\boldsymbol{C}$ the *off*-diagonal elements. We further split $\boldsymbol{C} = \boldsymbol{L} + \boldsymbol{V}$, the strictly lower triangular and strictly upper triangular elements, namely

$$\boldsymbol{D} = \begin{pmatrix} a_{11} & 0 & 0 & 0 & 0 \\ 0 & a_{22} & 0 & 0 & 0 \\ 0 & 0 & \ddots & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & a_{nn} \end{pmatrix};$$

$$\boldsymbol{V} = \begin{pmatrix} 0 & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & 0 & a_{23} & \dots & a_{2n} \\ 0 & 0 & 0 & \dots & \dots \\ 0 & 0 & 0 & 0 & a_{n-1\,n} \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}; \quad \boldsymbol{L} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ a_{21} & 0 & 0 & 0 & 0 \\ a_{31} & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & 0 & 0 \\ a_{n1} & a_{n2} & \dots & a_{n\,n-1} & 0 \end{pmatrix}.$$

Eqn. 12.7 can thus be written as

$$\boldsymbol{DU} = \boldsymbol{b} - \boldsymbol{CU} = \boldsymbol{b} - (\boldsymbol{L} + \boldsymbol{V})\boldsymbol{U} \tag{12.8}$$

The simplest iterative scheme essentially takes $\boldsymbol{B} = \boldsymbol{D}$ (see Eqn. 12.4), where $\boldsymbol{D}$ is the diagonal part of $\boldsymbol{A}$ and $\boldsymbol{C} = \boldsymbol{L} + \boldsymbol{V}$ has zeros on its diagonal. We then defined the iterative **Jacobi** scheme

$$\boldsymbol{DU}^{j+1} = \boldsymbol{b} - C\boldsymbol{U}^{j}$$

giving

$$\boldsymbol{U}^{j+1} = \boldsymbol{D}^{-1}(\boldsymbol{b} - C\boldsymbol{U}^{j}) \tag{12.9}$$

or

$$U_{mn}^{j+1} = \frac{1}{4}\left[U_{mn+1}^{j} + U_{mn-1}^{j} + U_{m+1n}^{j} + U_{m-1n}^{j} - h^2 f_{mn}\right].$$

This calculates all the new iterations at level $j+1$ before updating U. The matrix $\boldsymbol{D}^{-1}\boldsymbol{C}$ is called the *point Jacobi iteration* matrix.

We also considered the **Gauss-Seidel** (G-S) 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

$$U_{mn}^{j+1} = \frac{1}{4}\left[U_{mn+1}^{j} + U_{mn-1}^{j+1} + U_{m+1n}^{j} + U_{m-1n}^{j+1} - h^2 f_{mn}\right]. \tag{12.10}$$

This essentially involves back-substitution of the calculated values. Consider Eqn. 12.8, the G-S iteration is defined by

$$\boldsymbol{DU}^{j+1} = \boldsymbol{b} - \boldsymbol{LU}^{j+1} - \boldsymbol{VU}^{j}, \tag{12.11}$$

hence

$$(\boldsymbol{D} + \boldsymbol{L})\boldsymbol{U}^{j+1} = \boldsymbol{b} - \boldsymbol{VU}^{j}, \tag{12.12}$$

giving

$$\boldsymbol{U}^{j+1} = (\boldsymbol{D} + \boldsymbol{L})^{-1}\boldsymbol{b} - (\boldsymbol{D} + \boldsymbol{L})^{-1}\boldsymbol{VU}^{j}. \tag{12.13}$$

Here $(\boldsymbol{D} + \boldsymbol{L})^{-1}\boldsymbol{V}$ is called the Gauss-Seidel iteration matrix.

A necessary and sufficient condition for convergence

The iterative methods described can be written as

$$\boldsymbol{x}^{j+1} = \boldsymbol{G}\boldsymbol{x}^{j} + \boldsymbol{e}, \tag{12.14}$$

where $\boldsymbol{G}$ is the *iteration matrix*. This was derived from the original discretised system by re-arranging them into the form

$$\boldsymbol{x} = \boldsymbol{G}\boldsymbol{x} + \boldsymbol{c}, \tag{12.15}$$

with $\boldsymbol{c}$ a column vector of known values. Now we know that a solution to an equation of form $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is the solution of Eqn. 12.15. The error to the exact solution is defined by

$$\boldsymbol{e}^{j+1} = \boldsymbol{x} - \boldsymbol{x}^{j}$$

in the j-th approximation, so by subtracting the above two equations, it follows that

$$\boldsymbol{e}^{j+1} = \boldsymbol{G}\boldsymbol{e}^{j}. \tag{12.16}$$

Hence, it follows

$$\boldsymbol{e}^{j} = \boldsymbol{G}\boldsymbol{e}^{j-1} = \boldsymbol{G}^{2}\boldsymbol{e}^{j-2} = \ldots = \boldsymbol{G}^{n}\boldsymbol{e}^{0},$$

and hence the sequence of iterative values $\boldsymbol{x}^{1}, \boldsymbol{x}^{2}, \ldots, \boldsymbol{x}^{j}, \ldots$ will converge to $\boldsymbol{x}$ as $j \to \infty$ if

$$\lim_{j\to\infty} \boldsymbol{e}^{j} = 0.$$

Since $\boldsymbol{x}^{0}$ and $\boldsymbol{e}^{0}$ are arbitrary, it follows that the iterations will converge if and only if

$$\lim_{j\to\infty} \boldsymbol{G}^{j} = 0.$$

If a matrix $\boldsymbol{G}$ of order m has m linearly independent eigenvectors $\boldsymbol{v}_k,\ k = 1, \ldots, m$, then the arbitrary error vector $\boldsymbol{e}^{0}$ can be expressed uniquely as a linear combination, namely

$$\boldsymbol{e}^{0} = \sum_{s=1}^{m} c_k \boldsymbol{v}_k, \tag{12.17}$$

where c_k are scalars. Hence

$$\boldsymbol{e}^{1} = \boldsymbol{G}\boldsymbol{e}^{0} = \sum_{s=1}^{m} c_k \boldsymbol{G}\boldsymbol{v}_k,$$

but $\boldsymbol{G}\boldsymbol{v}_k = \lambda_k \boldsymbol{v}_k$ by definition of an eigenvalue, where λ_k is the eigenvalue associated with the eigenvectors $\boldsymbol{v}_k$. Hence

$$\boldsymbol{e}^{1} = \sum_{s=1}^{m} c_k \lambda_k \boldsymbol{v}_k,$$

and similarly

$$\boldsymbol{e}^{j} = \sum_{s=1}^{m} c_k \lambda_k^{j} \boldsymbol{v}_k.$$

Therefore $\boldsymbol{e}^{j}$ will tend to the zero vector as $j \to \infty$ for arbitrary $\boldsymbol{e}^{0}$, if

$$|\lambda_k| < 1$$

for all k, *i.e.* the iterations will converge for arbitrary $\boldsymbol{x}^{0}$ if the spectral radius $\rho(\boldsymbol{G})$ of $\boldsymbol{G}$ is less than one.

Rate of convergence

Assume that the iteration matrix $\boldsymbol{G}$ has m linearly independent eigenvectors and eigenvalues λ_k and that

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \ldots > \lambda_m$$

The error vector[†] $\boldsymbol{e}^{(j)}$ can be expressed as

$$\boldsymbol{e}^{(j)} = \lambda_1^j \left(c_1 \boldsymbol{v_1} + (\frac{\lambda_2}{\lambda_1})^j c_2 \boldsymbol{v_2} + \ldots + (\frac{\lambda_m}{\lambda_1})^j c_m \boldsymbol{v_m} \right).$$

For large j this shows that

$$\boldsymbol{e}^{(j)} = \lambda_1^j c_1 \boldsymbol{v_1};$$

similarly

$$\boldsymbol{e}^{(j+1)} = \lambda_1^{j+1} c_1 \boldsymbol{v_1},$$

so

$$\boldsymbol{e}^{(j+1)} = \lambda_1 \boldsymbol{e}^{(j)}.$$

If the $i-$th component of $\boldsymbol{e}^{(j)}$ is denoted by $e_i^{(j)}$ it is seen that

$$\frac{|e_i^{(j)}|}{|e_i^{(j+1)}|} = \frac{1}{\lambda_1} = \frac{1}{\rho(\boldsymbol{G})}$$

For large j,

$$\boldsymbol{e}^{(j)} \approx \lambda_1 \boldsymbol{e}^{(j-1)},$$

therefore

$$\boldsymbol{e}^{(j+\alpha)} \approx \lambda_1 \boldsymbol{e}^{(j+\alpha-1)} \approx \ldots \approx \lambda_1^\alpha \boldsymbol{e}^{(j)}, \quad \alpha = 1, 2, \ldots$$

As an example, take $\boldsymbol{e}_i^{(j)} = 10^{-2}$ and $\boldsymbol{e}_i^{(j+1)} = 10^{-4}$. Then $\boldsymbol{e}_i^{(j)} / \boldsymbol{e}_i^{(j+1)} = 10^2$, and the logarithm in base 10 is 2. Hence, since

$$\log_{10}(1/p) = -\log p$$

is an indication of number of decimal digits by which the error decreases per iteration. If we want to reduce the size of the error by 10^{-p}, say, then the number of iterations required to achieve this will be the least value of α for which

$$|\lambda_1^\alpha| = \rho^\alpha \leq 10^{-p}.$$

Taking logs and recalling that $\log \rho$ is negative for a convergent iteration leads to

$$\alpha \geq \frac{p}{-\log_{10} \rho}.$$

In terms of the matrix, $\boldsymbol{B} = \boldsymbol{D} + \boldsymbol{L}$ where $\boldsymbol{L}$ is the lower triangular part of $\boldsymbol{A}$. The spectral radii of the Jacobi and Gauss-Seidel matrices can be found. In fact, for large $M \equiv (N-1)^2$ it can be shown that

$$\rho_J = \cos\left(\frac{\pi}{N}\right) \approx 1 - \frac{\pi^2}{2N^2}, \quad \rho_G = \rho_J^2 \approx 1 - \frac{\pi^2}{N^2} \tag{12.18}$$

[†] slight change in notation by use of the () brackets!

This agrees with the factor of 4 error reduction we found using the toy-matrix problem used. 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

$$\alpha = \frac{p \log 10}{-\log \rho} \approx 0.467 p N^2 \text{ or } 0.233 p N^2 \tag{12.19}$$

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

12.2 Successive Over-Relaxation

Consider Eqn. 12.13, we may represent the scheme in terms of the residual vector

$$\boldsymbol{r}^j = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{U}^j,$$

which is the amount by which the j-th estimate fails to satisfy the equation. We then have that

$$\boldsymbol{U}^{j+1} = \boldsymbol{U}^j + \boldsymbol{B}^{-1}\boldsymbol{r}^j \text{ or } \boldsymbol{r}^{j+1} = C\boldsymbol{B}^{-1}\boldsymbol{r}^j. \tag{12.20}$$

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 **over-relaxation** scheme

$$\boldsymbol{U}^{j+1} = \boldsymbol{U}^j + \omega\boldsymbol{B}^{-1}\boldsymbol{r}^j. \tag{12.21}$$

In terms of the code we modify (12.10) to read

$$U^{j+1}_{mn} = (1-\omega)U^j_{mn} + \frac{\omega}{4}\left[U^j_{mn+1} + U^{j+1}_{mn-1} + U^j_{m+1n} + U^{j+1}_{m-1n} - h^2 f_{mn}\right]. \tag{12.22}$$

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.

We have already alluded above that the rate of convergence of the methods is related to the spectral radius – ideally we want to choose an optimum value of $\omega = \omega_o$ that minimises the spectral radius of the iteration matrix. A general formula for an arbitrary set of linear equations is difficult to find, The following result, which we do not prove (see Varga[‡]) for the Jacobi iteration matrix, can be shown to be

$$\omega_o = \frac{2}{1 + \sqrt{1 - \rho^2(\boldsymbol{B})}},$$

where $\rho(\boldsymbol{B})$ is the spectral radius of the Jacobi iteration matrix – here the matrix $\boldsymbol{A}$ is real and symmetric and has **block** tridiagonal form, with non-zero off diagonal elements of the Jacobi iteration matrix $\boldsymbol{B}$.

In general the optimum ω varies with N. But the optimal ω brings a very marked improvement to $\rho \approx 1 - 2\pi/N$ and $\alpha \approx 0.37 pN$. The **Successive Over-Relaxation** (**SOR**) method (12.22) requires $O(N)$ iterations not $O(N^2)$. But can we do better still?

[‡]Varga, R. S. (1992) *Matrix Iterative Analysis*, Prentice-Hall, New-Jersey

Results for the toy-problem with just $N = 4$ (see Eqn. 11.3) are shown in Fig. 12.1. Observe the G-S method halves the number of iterations required, to achieve a near machine precision residual $\boldsymbol{r}^j = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{U}^j$ value. This is in almost perfect agreement with the theoretical result given by expressions (12.19). The SOR method gives still better convergence (nearly halving the iteration count again), after having found the optimal $\omega_o \approx 1.07$.

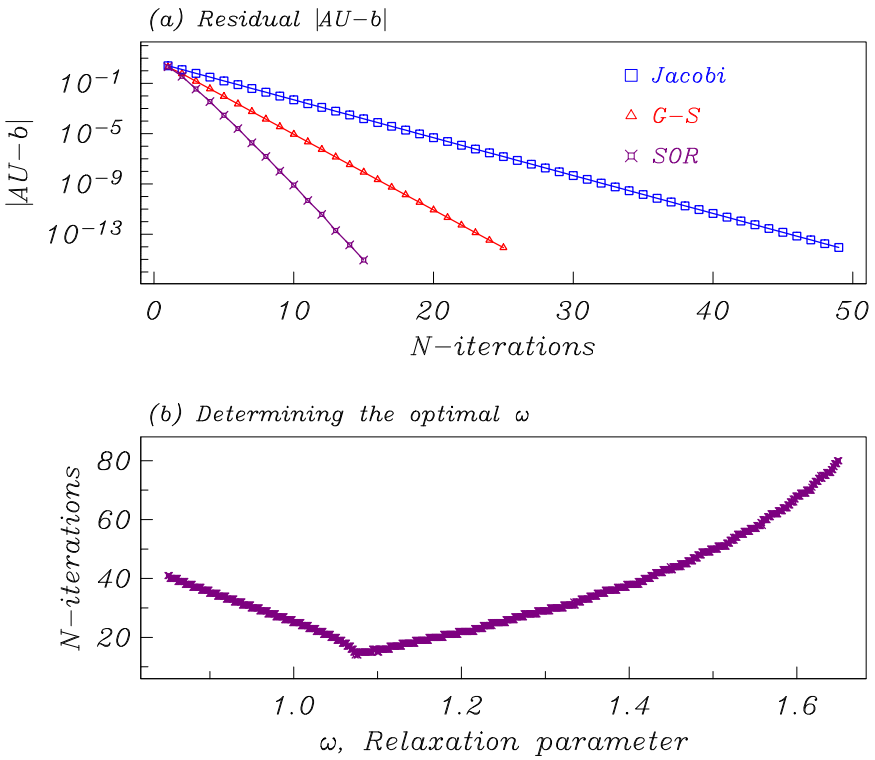

Figure 12.1: (a) Iterations required by the Jacobi, Gauss-Seidel and SOR methods. (b) Variation of the iteration count on varying the SOR relaxation parameter ω in Eqn. 12.22. The SOR computation in the upper figure is by setting the optimal $\omega_o \approx 1.07$ identified from plot (b).

13 Introduction to Multigrid Methods

Last lecture we investigated the behaviour of our 3 iterative methods on different grids and with different error wavenumbers. Essentially we found:

1. Jacobi and Gauss-Seidel quickly smoothed out the high wavenumbers (oscillations on a small scale). However, they were very slow to iron out the low wavenumbers (long waves). After a few iterations the error and residuals were dominated by these low wavenumbers. Thus even if the solution to the entire problem varies on a short length-scale, the errors after a few iterations vary on a large scale.

2. These low wavenumber modes do not require a lot of points to resolve them well. Furthermore, on coarser grids they would be damped more quickly. This suggests that it might be worth swapping between more than one grid. This is the idea behind **multigrid methods.**

3. The over-relaxation methods (SOR), although their overall convergence was much faster than Jacobi/GS, did not smooth the solution at all well. At all stages, the error vector, though small in magnitude, consisted of a mixture of wavenumbers. This does not lend itself well to the idea of swapping between grids, and so surprisingly, we will use our 2nd best method (Gauss-Seidel) on the various grids.

Two grid strategy
Suppose we have a course grid C and a fine grid F containing twice as many points in each direction. We wish to solve $A_F\mathbf{u}_F = \mathbf{b}_F$ on the fine grid. The corresponding approximation on C is $A_c\mathbf{u}_c = \mathbf{b}_c$.

(a) We begin with a few Gauss-Seidel sweeps on F to get rid of the short waves, obtaining an approximation $\mathbf{u}_F$.

(b) Next we calculate the residuals $\mathbf{r}_F = A\mathbf{u}_F - \mathbf{b}$.

(c) Now we transfer to the coarser grid C using a restriction operator $\mathcal{R} : \mathrm{F} \to \mathrm{C}$, to find $\mathbf{r}_c = \mathcal{R}\mathbf{r}_F$.

(d) On the coarser grid we calculate the error $\mathbf{z_c}$ quickly, where $A_c\mathbf{z_c} = \mathbf{r_c}$.

(e) We now use an interpolation operator P to map the error $\mathbf{z_c}$ to $\mathbf{z_F} = \mathcal{P}\mathbf{z_c}$.

(f) We then modify our fine estimate by the calculated error, so that $\mathbf{u}_F + \mathbf{z_F}$ is the new solution estimate.

(g) We then take a few more sweeps using Gauss-Seidel, in case any short wave errors have crept in. This completes a cycle. If we wish we can repeat the process using our new estimate.

Before we can implement this scheme, we have to decide how to **interpolate** from a coarse mesh onto a finer one, and also, how best to **restrict** our solution on a finer mesh to a less

accurate coarse mesh. If we do this linearly, we can represent these transformations by rectangular matrices, $\mathcal{P}$ and $\mathcal{R}$. Let these have dimensions $\mu \times \nu$ and $\nu \times \mu$ respectively. As each grid point on the coarse mesh is also a point on the fine mesh, the simplest idea would be to use the same value at the new point. But it might be a good idea to use a linear combination of all the neighbouring points. We want to minimize the errors introduced by transfer between grids. Ideally, we would like $\mathcal{PR}$ and $\mathcal{RP}$ to be identity matrices but that would overconstrain the problem. It is a good idea to require the interpolation operator $\mathcal{P}$ and our restriction operator $\mathcal{R}$ to be adjoints with respect to the scalar product, $\{u, v\}$. If we denote the coarse and fine meshes by C and F, and let u_c and v_F be any vectors in the respective meshes. Then

$$\{\mathbf{u}_c, \mathcal{R}\mathbf{v}_F\} = \{\mathcal{P}\mathbf{u}_c, \mathbf{v}_F\} \quad \Longrightarrow \quad \mathcal{R} = P^T/4, \tag{13.1}$$

allowing for a factor of 4 from the halving of the steplengths (the vector $\mathbf{v}_F$ is 4 times the length of $\mathbf{u}_c$.) Note effectively we are replacing the matrix A_c by $\mathcal{R}\mathsf{A}_F\mathcal{P}$ – this is known as the Galerkin condition.

The interpolation or prolongation operator shown in Figure (13.1a) is easiest to choose, as we have less information to work with

$$\begin{aligned} u_F(2i, 2j) &= u_c(i, j) \\ u_F(2i+1, 2j) &= \tfrac{1}{2}(u_c(i, j) + u_c(i+1, j)) \\ u_F(2i, 2j+1) &= \tfrac{1}{2}(u_c(i, j) + u_c(i, j+1)) \\ u_F(2i+1, 2j+1) &= \tfrac{1}{4}(u_c(i, j) + u_c(i+1, j) + u_c(i, j+1) + u_c(i+1, j+1)) \end{aligned} \quad . \tag{13.2}$$

Using (13.1), the appropriate restriction operator, Figure (13.1b), is then $\mathbf{u}_c = \mathcal{R}\mathbf{u}_F$ where

$$\begin{aligned} u_c(i, j) = &\tfrac{1}{4}u_F(2i, 2j) + \ldots \\ &+ \tfrac{1}{8}(u_F(2i+1, 2j) + u_F(2i, 2j+1) + u_F(2i-1, 2j) + u_F(2i, 2j-1)) + \ldots \\ &+ \tfrac{1}{16}(u_F(2i+1, 2j+1) + u_F(2i-1, 2j+1) + u_F(2i-1, 2j-1) + u_F(2i+1, 2j-1)) \end{aligned} \quad . \tag{13.3}$$

This keeps the errors under control when flipping between grids.

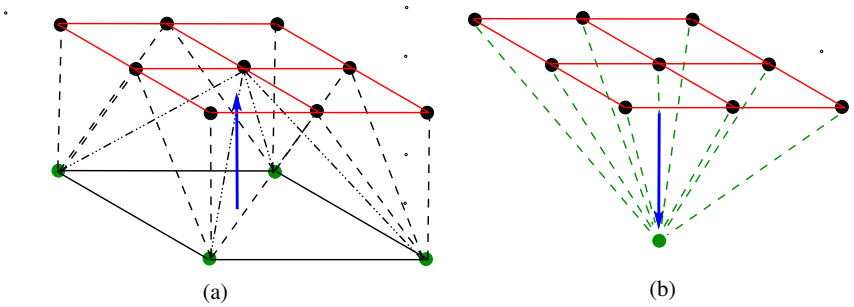

Figure 13.1: (a) Prolongation operation restoring values on a fine grid from a coarse grid. (b) Restriction operation based on full-weighting of the fine-grid values to produce a coarse grid.

13.1 More than 2 grids

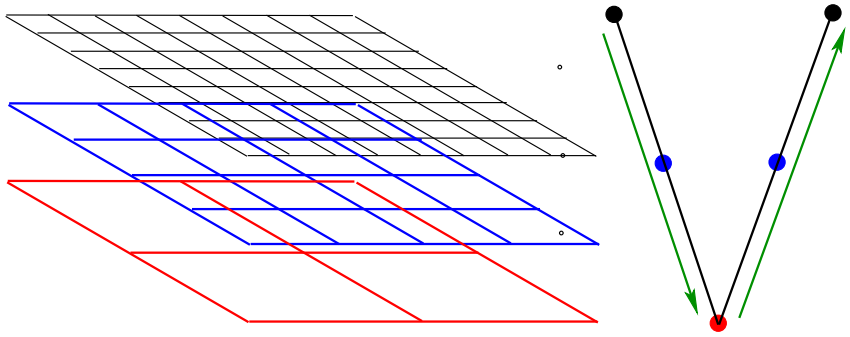

Figure 13.2: Grid hierarchy and traversal order for a V-cycle multigrid method.

The coarser grid has twice the steplength, but it may still be slow to solve for the errors as in step (d) above. But there is nothing to prevent us from performing the same algorithm again, transferring oo a still coarser grid, If our initial N is of the form 2^p+1, we may transfer all the way down to $p=1$ and then move up the chain back to the finest grid.

The program MultigridV.m performs a cycle transferring down the grids to solve for the error on a coarse grid, and then interpolating the **error** back up to the finest grid. This is known as a 'V-cycle'. It is extremely efficient, but can still be improved upon!

13.2 Full multigrid

The Multigrid algorithm starts on the fine grid with an arbitrary initial guess. Instead, we could solve the problem on a coarse grid and interpolate down to give a better initial estimate, and hence reduce the iterations required. The program FullMG.m starts on the coarsest grid and interpolates to the next grid. There it solves the problem with a 2 grid algorithm, and then interpolates again, each time using a V-cycle down to the coarsest grid. Not only is it faster, but full MG also obtains the solution on every grid as opposed to the final grid only.

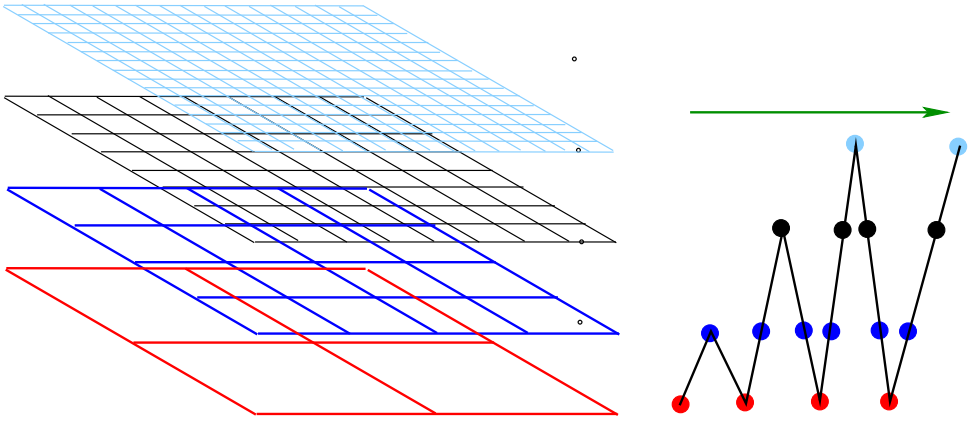

Figure 13.3: Grid hierarchy and traversal order for a full multigrid method.