Chapter 3

Iterative solution of linear systems

3.1 Iterative solution of elliptic PDEs

Consider the elliptic model equation

$$\phi_{xx} + \phi_{yy} = f(x, y) \tag{3.1}$$

defined for $0 \leq x \leq L_x$, $0 \leq y \leq L_x$ and subject to homogeneous boundary conditions ($\phi = 0$ on the boundary). Approximating (3.1) using second-order accurate central differences on a rectangular grid with $\Delta x = L_x/(I+1)$, $\Delta y = L_y/(J+1)$ gives

$$\frac{1}{\Delta x^2}\left(\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}\right) + \frac{1}{\Delta y^2}\left(\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}\right) = f_{i,j}, \tag{3.2}$$

at the inner points $i = 1, \dots, I$; $j = 1, \dots, J$. Equation (3.2) represents a system of linear algebraic equations for the unknown array $\boldsymbol{\phi}$ that can be written, in vector form, as

$$A\boldsymbol{\phi} = \mathbf{f}, \tag{3.3}$$

with

$$\boldsymbol{\phi} = \left(\phi_{1,1}, \dots, \phi_{I,1}, \dots, \phi_{i,j}, \dots, \phi_{1,J}, \dots, \phi_{I,J}\right)^T, \tag{3.4}$$

and

$$A = \left(\begin{array}{ccccccccccccc}
\bullet & \bullet & 0 & \dots & 0 & \bullet & 0 & \dots & \dots & \dots & \dots & \dots & 0 \\
\bullet & \bullet & \bullet & 0 & \dots & 0 & \bullet & 0 & & & & & \vdots \\
0 & \bullet & \bullet & \bullet & 0 & \dots & 0 & \bullet & 0 & & & & \vdots \\
0 & 0 & \ddots & \ddots & \ddots & 0 & \dots & 0 & \ddots & 0 & & & 0 \\
0 & \dots & 0 & \ddots & \ddots & \ddots & 0 & \dots & 0 & \ddots & 0 & & 0 \\
\bullet & 0 & \dots & 0 & \bullet & \bullet & \bullet & 0 & \dots & 0 & \bullet & 0 & 0 \\
0 & \ddots & 0 & \dots & 0 & \ddots & \ddots & \ddots & 0 & \dots & 0 & \ddots & 0 \\
0 & 0 & \bullet & 0 & \dots & 0 & \bullet & \bullet & \bullet & 0 & \dots & 0 & \bullet \\
\vdots & & 0 & \ddots & 0 & \dots & 0 & \ddots & \ddots & \ddots & 0 & \dots & 0 \\
\vdots & & & 0 & \ddots & 0 & \dots & 0 & \ddots & \ddots & \ddots & 0 & 0 \\
\vdots & & & & 0 & \bullet & 0 & \dots & 0 & \bullet & \bullet & \bullet & 0 \\
\vdots & & & & & 0 & \bullet & 0 & \dots & 0 & \bullet & \bullet & \bullet \\
0 & \dots & \dots & \dots & \dots & \dots & 0 & \bullet & 0 & \dots & 0 & \bullet & \bullet
\end{array}\right), \tag{3.5}$$

in which $\bullet$ denotes a non-zero elements.

Inversion of matrix A is very time consuming. Although the matrix is sparse, it is very large (its dimensions are $IJ \times IJ$ in two dimensions, $IJK \times IJK$ in three; its bandwidth is I in two dimensions, $I \times J$ in three). The number of operations required for the direct inversion of a matrix of this form (by Gauss elimination, for example) scales with the size of the matrix times the square of the bandwidth. This yields I^2 operations per grid point in two dimensions, $I^2 J^2$ in three dimensions. The total number of operations, therefore, is proportional to $I^3 J$ or $I^3 J^3 K$ respectively in two and three dimensions. These operation counts are too high for practical applications, except for very small I and J. Direct inversion of matrix A is, therefore, not feasible for most cases of interest, and the system of equations (3.3) is generally solved by iterative methods of the form

$$A_1 \boldsymbol{\phi}^{n+1} = A_2 \boldsymbol{\phi}^n + \mathbf{d}. \tag{3.6}$$

To determine the suitability of an iterative scheme, the following criteria will be assumed:

1. For large n, $\boldsymbol{\phi}^n$ must approach the exact solution of (3.3), $\boldsymbol{\phi}$:

$$\lim_{n\to\infty} \boldsymbol{\phi}^n = \boldsymbol{\phi}. \tag{3.7}$$

2. Convergence should be rapid.

3. A_1 should be easy to invert (to decrease the cost of the inversion), and A_2 should be sparse (to decrease the amount of memory storage required.

Since, in general, I and J are of the same order, and the present analysis is only intended to give order-of-magnitude indications on the efficiency of iterative schemes, in the following the assumption will be made that $I = J$ and that $\Delta x = \Delta y$. The extension of the results to the general cases is left to the interested reader.

3.2 The Jacobi method

Consider the diagonal part D of matrix A, its strictly lower triangular part L and its strictly upper triangular part U. The Jacobi method takes $A_1 = D$, $A_2 = -(L+U)$:[1]

$$D\boldsymbol{\phi}^{n+1} = -(L+U)\boldsymbol{\phi}^n + \mathbf{f}. \tag{3.8}$$

Since D^{-1} is computed easily, $\boldsymbol{\phi}$ can be evaluated explicitly. Equation (3.8) can be written, at each i,j point, as

$$\phi_{i,j}^{n+1} = \frac{1}{4}\left(\phi_{i-1,j}^n + \phi_{i+1,j}^n + \phi_{i,j-1}^n + \phi_{i,j+1}^n\right) - \frac{\Delta x^2}{4} f_{i,j}, \tag{3.9}$$

in which all terms on the right-hand side are known. Notice that two arrays are required at each iteration level: $\boldsymbol{\phi}^n$ and $\boldsymbol{\phi}^{n+1}$. Defining

$$\Delta\boldsymbol{\phi} = \boldsymbol{\phi}^{n+1} - \boldsymbol{\phi}^n, \tag{3.10}$$

Equation (3.8) can be rewritten as

$$\Delta\boldsymbol{\phi} = D^{-1}\left[A\boldsymbol{\phi}^n - \mathbf{f}\right] = -D^{-1} r^n, \tag{3.11}$$

where $r^n = \mathbf{f} - A\boldsymbol{\phi}^n$ is the *residual* at the n-th iteration; this expression yields

$$\boldsymbol{\phi}^{n+1} = \left[\mathcal{I} - D^{-1}A\right]\boldsymbol{\phi}^n + \mathbf{b}, \tag{3.12}$$

where $\mathcal{I}$ is the identity matrix and $\mathbf{b} = D^{-1}\mathbf{f}$.

To prove convergence of the Jacobi method, define the error at iteration n:

$$\boldsymbol{\epsilon}^n = \boldsymbol{\phi}^n - \boldsymbol{\phi}; \tag{3.13}$$

[1]Note that, although often $A_1 - A_2 = A$, this requirement is not necessary. The SOR iteration scheme, for instance, does not satisfy this condition.

If the method converges, for large n, $\boldsymbol{\phi}^n$ must be equal to $\boldsymbol{\phi}^{n+1}$ and to $\boldsymbol{\phi}$. Substituting first $\boldsymbol{\phi}^n$ and then $\boldsymbol{\phi}$ into (3.12) yields

$$\begin{aligned} \boldsymbol{\phi}^n &= \left[\mathcal{I} - D^{-1}A\right] \boldsymbol{\phi}^{n-1} + \mathbf{b} \qquad (3.14) \\ \boldsymbol{\phi} &= \left[\mathcal{I} - D^{-1}A\right] \boldsymbol{\phi} + \mathbf{b}. \qquad (3.15) \end{aligned}$$

Subtracting (3.15) from (3.14) one obtains an equation for the error $\boldsymbol{\epsilon}^n$:

$$\boldsymbol{\epsilon}^n = \left[\mathcal{I} - D^{-1}A\right] \boldsymbol{\epsilon}^{n-1}; \qquad (3.16)$$

whose solution is of the form

$$\boldsymbol{\epsilon}^n = \sum_{i,j} C_{i,j} \left(\lambda_{i,j}\right)^n \boldsymbol{\psi}_{i,j}, \qquad (3.17)$$

in which $\lambda_{i,j}$ and $\boldsymbol{\psi}_{i,j}$ are respectively the eigenvalues and the associated eigenvectors of the Jacobi matrix $[\mathcal{I} - D^{-1}A]$. From (3.17) it is clear that the Jacobi method will converge if $\Re(\lambda_{i,j}) < 1$ for all i and j. The error component that decreases most slowly is the one associated with the eigenvalue that has the largest real part (in absolute value). This eigenvalue is the *spectral radius* of the matrix:

$$\rho[\mathcal{I} - D^{-1}A] = \max_{i,j} |\lambda_{i,j}| . \qquad (3.18)$$

The spectral radius ρ, in the context of iterative methods, is also called the *convergence factor*, and the quantity $-\log(\rho)$ is the *convergence rate.*

For the Jacobi method, the eigenvalues are

$$\lambda_{i,j} = \frac{1}{2}\left[\cos\frac{i\pi}{I+1} + \cos\frac{j\pi}{J+1}\right], \qquad (3.19)$$

and the spectral radius is

$$\begin{aligned} \rho[\mathcal{I} - D^{-1}A] &= \cos(\pi \Delta x) \\ &\simeq 1 - (\pi\Delta x)^2/2 \\ &\simeq e^{-(\pi\Delta x)^2/2}; \qquad (3.20) \end{aligned}$$

since $\rho < 1$, the Jacobi method converges with convergence rate $(\pi\Delta x)^2/2$. Figure 3.1 shows the solution of the one-dimensional problem

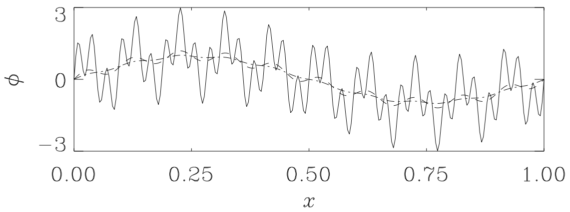

Figure 3.1: Solution of (3.21) using the Jacobi method. ——— Initial condition; - - - 50 iterations; —·— 100 iterations.

$$\phi'' = 0; \qquad \phi(x, 0) = \sin 2\pi x + \sin 20\pi x + \sin 64\pi x, \qquad (3.21)$$

and $I = 256$. It can be seen that, although the short wavelength oscillation has been damped significantly, the lower wavenumber modes are still significant.

A useful measure of the convergence rate is the number of iterations N required to reduce the error to some small number δ. It is easy to show that

$$N \sim \frac{2}{(\pi\Delta x)^2} \ln(1/\delta). \qquad (3.22)$$

The number of iterations required for convergence is $N \sim I^2$, and the number of operations, equal to the product of the number of points by the number of iterations, proportional to I^4, can be larger than that required by the direct inversion.

3.3 The Gauss-Seidel (GS) method.

The Gauss-Seidel method takes $A_1 = L + D$ and $A_2 = U$. This yields

$$(L + D)\boldsymbol{\phi}^{n+1} = -U\boldsymbol{\phi}^n + \mathbf{f}, \qquad (3.23)$$

or, in Δ-form,

$$\Delta\boldsymbol{\phi} = -(L+D)^{-1}A\boldsymbol{\phi}^n + \mathbf{b}, \tag{3.24}$$

which yields

$$\boldsymbol{\phi}^{n+1} = \left[\mathcal{I} - (L+D)^{-1}A\right]\boldsymbol{\phi}^n + \mathbf{b}, \tag{3.25}$$

where $\mathbf{b} = (L+D)^{-1}\mathbf{f}$. Since the matrix $L+D$ is lower triangular, it can also be inverted easily, and $\phi_{i,j}^{n+1}$ evaluated explicitly. At point i,j, equation (3.25) reduces to

$$\phi_{i,j}^{n+1} = \frac{1}{4}\left(\phi_{i-1,j}^{n+1} + \phi_{i+1,j}^{n} + \phi_{i,j-1}^{n+1} + \phi_{i,j+1}^{n}\right) - \frac{\Delta x^2}{4} f_{i,j}, \tag{3.26}$$

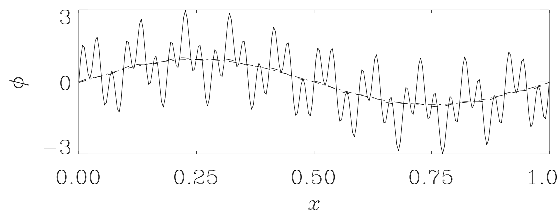

Figure 3.2: Solution of (3.21) using the Gauss-Seidel method. —— Initial condition; - - - 50 iterations; —·— 100 iterations.

in which every term on the right-hand side is known either from previous iterations ($\phi_{i+1,j}^{n}$ and $\phi_{i,j+1}^{n}$), or from previous calculations at iteration level $n+1$ ($\phi_{i-1,j}^{n+1}$ and $\phi_{i,j-1}^{n+1}$). Notice that (3.26) requires only one array to be defined at each iteration level: the new values of $\boldsymbol{\phi}$ replace the old ones as soon as they are calculated. The GS method is, therefore, simpler to program than the Jacobi method. The spectral radius for the Gauss-Seidel method is given by

$$\begin{aligned}\rho\left[\mathcal{I} - (L+D)^{-1}A\right] &= \cos^2(\pi\Delta x) \\ &\simeq 1 - (\pi\Delta x)^2 \simeq e^{-(\pi\Delta x)^2},\end{aligned} \tag{3.27}$$

and the convergence rate is $(\pi\Delta x)^2$; the number of iterations required to reduce the error is:

$$N \sim \frac{1}{(\pi\Delta x)^2}\ln(1/\delta); \tag{3.28}$$

the GS method, therefore, converges twice as fast as the Jacobi method due to the fact that more information at iteration level $n+1$ is used in (3.26) than in (3.9); such convergence rate is still not sufficient for practical applications. Figure 3.2 shows the solution of the test problem (3.21) using the Gauss-Seidel method. The solution is closer to the converged value than with the Jacobi scheme, and the short wavelength fluctuations have been damped even more effectively; the level of convergence of the low wavenumber component is, however, still unsatisfactory.

3.4 The Gauss-Seidel method with Line Relaxation (GSLR).

Further improvement in the convergence rate can be obtained if one of the remaining terms evaluated at iteration n in (3.26) is instead evaluated at iteration $n+1$ instead. One can write, for example,

$$\phi_{i,j}^{n+1} = \frac{1}{4}\left(\phi_{i-1,j}^{n+1} + \phi_{i+1,j}^{n} + \phi_{i,j-1}^{n+1} + \phi_{i,j+1}^{n+1}\right) - \frac{\Delta x^2}{4} f_{i,j}; \tag{3.29}$$

At each i-level, $\phi_{i+1,j}^{n}$ is known from the calculation of the previous iteration level, and $\phi_{i-1,j}^{n+1}$ is known either from the boundary condition, or from the calculation of the previous i-level. One can, therefore, rewrite (3.29) as

$$-\phi_{i,j-1}^{n+1} + 4\phi_{i,j}^{n+1} - \phi_{i,j+1}^{n+1} = \left(\phi_{i-1,j}^{n+1} + \phi_{i+1,j}^{n}\right) - \Delta x^2 f_{i,j} \text{ for } j = 1,\ldots,J. \tag{3.30}$$

Since the right-hand side of (3.30) is known, this equation represents a system of J linear algebraic equations of the form

$$B\boldsymbol{\phi}_i^{n+1} = \mathbf{d}, \tag{3.31}$$

where

$$\boldsymbol{\phi}_i^{n+1} = \left(\phi_{i,1},\ldots,\phi_{i,j},\ldots,\phi_{i,J}\right)^T, \tag{3.32}$$

and

$$B = \begin{pmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 4 & -1 \\ & & & -1 & 4 \end{pmatrix}. \tag{3.33}$$

Since the matrix B is tridiagonal, the system of equations (3.31) can be solved easily by direct inversion using the Thomas algorithm.

The convergence of the GSLR method is faster than that of the GS method, but more operations per grid point are required. This method is particularly useful when the solution varies more rapidly in one direction than in the other, and more grid points are required in the former direction, which should be treated implicitly.

3.5 Successive Over-Relaxation methods (SOR).

Successive Over-Relaxation methods attempt to use the past iteration history to accelerate convergence. An iteration method evaluates the value of ϕ at the new iteration from the value at the old iteration level:

$$\boldsymbol{\phi}^{n+1} = \left[\mathcal{I} - M^{-1}A\right]\boldsymbol{\phi}^n + \mathbf{b}, \tag{3.34}$$

where the form of the matrix M depends on the iteration method chosen. With relaxation methods, one first computes a provisional value $\widetilde{\boldsymbol{\phi}}$ that satisfies

$$\widetilde{\boldsymbol{\phi}} = \left[\mathcal{I} - M^{-1}A\right]\boldsymbol{\phi}^n + \mathbf{b}, \tag{3.35}$$

then evaluates $\boldsymbol{\phi}^{n+1}$ from an extrapolation of $\boldsymbol{\phi}^n$ and $\widetilde{\boldsymbol{\phi}}$:

$$\boldsymbol{\phi}^{n+1} = \widetilde{\boldsymbol{\phi}} + \alpha\left(\widetilde{\boldsymbol{\phi}} - \boldsymbol{\phi}^n\right). \tag{3.36}$$

If the relaxation parameter ω is defined as $\omega = 1 + \alpha$, (3.36) can be rewritten as

$$\boldsymbol{\phi}^{n+1} = \omega\widetilde{\boldsymbol{\phi}} + (1-\omega)\boldsymbol{\phi}^n \tag{3.37}$$

or, using (3.35)

$$\boldsymbol{\phi}^{n+1} = \left\{\omega\left[\mathcal{I} - M^{-1}A\right] + (1-\omega)\mathcal{I}\right\}\boldsymbol{\phi}^n + \omega\mathbf{b}. \tag{3.38}$$

Equation (3.38) is the general form of a relaxation scheme. Three cases are possible: $\omega > 1$ (over-relaxation); $\omega < 1$, (under-relaxation), and $\omega = 1$ (no relaxation).

If λ' is an eigenvalue of $[\mathcal{I} - M^{-1}A]$, it can be shown that $\lambda = \omega\lambda' + (1-\omega)$ is an eigenvalue of $\left\{\omega\left[\mathcal{I} - M^{-1}A\right] + (1-\omega)\mathcal{I}\right\}$. It is also easy to see that, for a given iteration method, the optimum relaxation parameter ω_{opt} is the one that gives $|\lambda_{min}| = |\lambda_{max}|$.

For the Jacobi method, $\lambda'_{max} = -\lambda'_{min} = \cos(\pi\Delta x)$; from the point of view of the relaxation, the Jacobi method cannot be accelerated by relaxation. For $\omega \geq 1$, λ_{min} becomes less than -1, and the numerical solution diverges. For $\omega < 1$, the numerical solution converges to the exact one, but at a rate slower than that of the Jacobi method (under-relaxing the Jacobi scheme can, however, lead to rapid damping of the high-frequencies). For $\omega = 1$ the convergence rate is $(\pi\Delta x)^2/2$.

Applying relaxation to the Gauss-Seidel method yields

$$\widetilde{\boldsymbol{\phi}} = \left[\mathcal{I} - (L+D)^{-1}A\right]\boldsymbol{\phi}^n + \mathbf{b}, \tag{3.39}$$

and

$$\boldsymbol{\phi}^{n+1} = \left\{\omega\left[\mathcal{I} - (L+D)^{-1}A\right] + (1-\omega)\mathcal{I}\right\}\boldsymbol{\phi}^n + \omega\mathbf{b}. \tag{3.40}$$

Since $\lambda'_{min} \simeq 0$ and $\lambda'_{max} = \cos(\pi\Delta x)$, the maximum and eigenvalues of the Gauss-Seidel method with relaxation are

$$\begin{cases} \lambda_{min} &= (1-\omega) \\ \lambda_{max} &= \omega\lambda'_{max} + (1-\omega), \end{cases} \tag{3.41}$$

and

$$\omega_{opt} = \frac{2}{2 - \lambda'_{max}}. \tag{3.42}$$

The spectral radius of the Gauss-Seidel relaxation matrix is, therefore,

$$\rho = |\lambda_{max}| = \frac{\lambda'_{max}}{2 - \lambda'_{max}}; \tag{3.43}$$

since $\lambda'_{max} \simeq 1 - (\pi \Delta x)^2$, $\rho \simeq 1 - 2(\pi \Delta x)^2$, and the convergence rate is $2(\pi \Delta x)^2$; this method converges twice as fast as the standard Gauss-Seidel scheme, four times as fast as the Jacobi method.

Further improvement in the convergence rate can be obtained if the over-relaxation is performed as soon as a new value of $\widetilde{\boldsymbol{\phi}}$ is calculated, rather than after the entire array $\widetilde{\boldsymbol{\phi}}$ has been evaluated. This method is known as SOR. Its matrix form can be derived by writing

$$D\widetilde{\boldsymbol{\phi}} = -L\boldsymbol{\phi}^{n+1} - U\boldsymbol{\phi}^n + \mathbf{f}, \tag{3.44}$$

and

$$D\boldsymbol{\phi}^{n+1} = \omega D\widetilde{\boldsymbol{\phi}} + (1 - \omega)D\boldsymbol{\phi}^n; \tag{3.45}$$

substituting (3.44) into (3.45) yields

$$\boldsymbol{\phi}^{n+1} = (D + \omega L)^{-1} \left[(1 - \omega)D - \omega U\right] \boldsymbol{\phi}^n + \omega \left(D + \omega L\right)^{-1} \mathbf{f}. \tag{3.46}$$

The corresponding expression for $\phi_{i,j}^{n+1}$ is

$$\phi_{i,j}^{n+1} = \frac{\omega}{4} \left[\phi_{i-1,j}^{n+1} + \phi_{i+1,j}^{n} + \phi_{i,j-1}^{n+1} + \phi_{i,j+1}^{n} - \Delta x^2 f_{i,j}\right] + (1 - \omega)\phi_{i,j}^{n}. \tag{3.47}$$

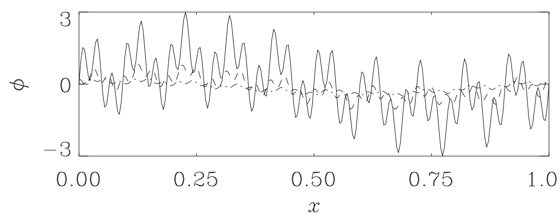

Figure 3.3: Solution of (3.21) using the SOR method (ω = 1.948). ——— Initial condition; - - - 50 iterations; —·— 100 iterations.

The eigenvalues of the SOR matrix are

$$\lambda = \frac{1}{4} \left[\omega \lambda_J \pm \sqrt{\omega^2 \lambda_J^2 - 4(\omega - 1)}\right]^2 \tag{3.48}$$

(where λ_J are the eigenvalues of the Jacobi matrix), and the optimum relaxation parameter is

$$\omega_{opt} = \frac{2}{1 + \sqrt{1 - \lambda_J^2}}. \tag{3.49}$$

The spectral radius of the SOR matrix is

$$\rho = \frac{1 - \sqrt{1 - \lambda_J^2}}{1 + \sqrt{1 - \lambda_J^2}} \simeq 1 - 2\pi \Delta x. \tag{3.50}$$

The convergence rate of SOR is $2\pi \Delta x$, much better than that of the methods examined previously. The number of iterations required to reduce the error to δ is

$$N \sim \frac{1}{2\pi \Delta x} \ln(1/\delta), \tag{3.51}$$

proportional to $1/\Delta x \sim I$, while, for the methods seen before $N \sim 1/\Delta x^2 \sim I^2$. The total number of operations, therefore, is proportional to I^3 rather than I^4. This very significant improvement of the convergence rate explains the popularity of this method for engineering applications. Figure 3.3 shows the improved convergence rate in the solution of the test problem. Notice that in this case the high wavenumber modes are the least damped ones. To obtain better damping of the high-frequency modes, Chebychev acceleration can be applied (see [9]), in which the relaxation parameter ω is initially set to 1 to damp the high-frequency components of the error, and then slowly changed at each sweep in such a way that its optimum value is approached asymptotically.

SOR can also be applied to Gauss-Seidel Line Relaxation (SLOR) with even more rapid convergence rate (although at the expense of increased CPU time). Line relaxation is recommended if sharp gradients are expected in one direction.

For most cases of engineering interest it is not possible to evaluate the optimal relaxation parameter exactly (as in the sample problem discussed here). In general, ω approaches 2 as the mesh is refined. It is usually better to choose ω greater than ω_{opt} rather than smaller than ω_{opt}, since the convergence rate decays more rapidly for $\omega < \omega_{opt}$ than for $\omega > \omega_{opt}$.

3.6 Alternating Direction Implicit (ADI) relaxation schemes

Alternating Direction Implicit (ADI) schemes solve the time-dependent problem

$$\frac{d\boldsymbol{\phi}}{dt} + \nabla^2 \boldsymbol{\phi} = \mathbf{f}; \tag{3.52}$$

as $t \to \infty$, the time derivative vanishes and $\boldsymbol{\phi}$ satisfies the standard Poisson equation (3.1). Applying the spatial discretization and a simple time-advancement scheme yields

$$\frac{\boldsymbol{\phi}^{n+1} - \boldsymbol{\phi}^n}{2\tau} = -A\boldsymbol{\phi} + \mathbf{f}, \tag{3.53}$$

where 2τ is a fictitious time step that (as will be shown later) plays the role of a relaxation parameter. The matrix A is then split into two operators (three in 3D), one for each coordinate direction: $A = A_x + A_y$. For a standard centered scheme, for instance, $A_x = \frac{1}{\Delta x^2} B(1, -2, 1)$. The time advancement is then split into two (or three) half-steps:

$$\begin{aligned}
\boldsymbol{\phi}^* - \boldsymbol{\phi}^n &= -\tau \left(A_x \boldsymbol{\phi}^* + A_y \boldsymbol{\phi}^n - \mathbf{f} \right) && (3.54)\\
\boldsymbol{\phi}^{n+1} - \boldsymbol{\phi}^* &= -\tau \left(A_x \boldsymbol{\phi}^* + A_y \boldsymbol{\phi}^{n+1} - \mathbf{f} \right), && (3.55)
\end{aligned}$$

which can be rewritten as

$$\begin{aligned}
\left[\mathcal{I} + \tau A_x \right] \boldsymbol{\phi}^* &= \left[\mathcal{I} - \tau A_y \right] \boldsymbol{\phi}^n + \tau \mathbf{f} && (3.56)\\
\left[\mathcal{I} + \tau A_y \right] \boldsymbol{\phi}^{n+1} &= \left[\mathcal{I} - \tau A_x \right] \boldsymbol{\phi}^* + \tau \mathbf{f}; && (3.57)
\end{aligned}$$

the operator $[\mathcal{I} + \tau A_x]$ is tridiagonal, and therefore can easily be inverted to calculate $\boldsymbol{\phi}^*$. The other operator, $[\mathcal{I} + \tau A_y]$ is banded tridiagonal; however, if one computes the transpose of $\boldsymbol{\phi}^{n+1}$, it becomes tridiagonal. Both A_x and A_y are positive definite, and convergence of the ADI scheme can be shown if $\tau > 0$.

It is common to vary τ at each time-step (or iteration) to damp recursively the various frequency components of the error. Various recipes to achieve this can be found in Refs. [9, 5]. In general, τ is chosen to span the expected range of eigenvalues over a cycle of k iterations. The convergence rate that can be achieved after each cycle is proportional to $\Delta x^{1/k}$, resulting in a number of operations proportional to I^2, an improvement by one order of magnitude over the SOR method.

3.7 Multigrid methods.

The principle of multigrid methods is very simple: it is easy to find relaxation methods that produce approximations to the solution of an elliptic equation with *smooth error.* If the error is smooth, corrections to these approximations can be efficiently calculated on *coarse grids.* A method that implements a few relaxation steps on a fine grid and corrections to the iterative solution calculated on coarser grids is called a *multigrid method.* These methods require a total number of operations proportional to the number of grid points, $N \sim I^2$, and can be very efficient elliptic equation solvers.

3.7.1 Basics of multigrid: the Coarse-Grid Correction (CGC).

Let $Lu = f$ represent an elliptic partial differential equation such as (3.1), and let

$$L^h u^h = f^h \tag{3.58}$$

be its representation on a discrete grid with $\Delta x = \Delta y = h$. The Coarse-Grid Correction method is a two-grid algorithm based on the following sequence:

1. Obtain U^h (an approximation to u^h on the fine grid h) by performing n_1 iterations with some relaxation scheme.
2. Define the error on the fine grid
$$\epsilon^h = u^h - U^h; \tag{3.59}$$
by applying the operator L^h to both sides of Eq. (3.59) the relationship that the error must satisfy (*Defect Equation*) is obtained:
$$L^h \epsilon^h = r^h \tag{3.60}$$
where $r^h = f^h - L^h U^h$, the residual on the fine grid, is known. Since U^h is known, if (3.60) could be solved for ϵ^h, it would be possible to find u^h from (3.59).
3. Transfer the residual r^h to a coarse grid (with grid spacing $H = 2h$ in both directions) by some *restriction operator* I_h^H such that $r^H = I_h^H r^h$.
4. Solve the equation governing the error on the coarse grid [*Coarse Grid Defect Equation* (CGDE)]
$$L^H \epsilon^H = r^H, \tag{3.61}$$
using a method to be specified later.
5. Transfer the error ϵ^H from the coarse grid to the fine grid by some interpolation operator I_H^h: $\epsilon^h = I_H^h \epsilon^H$.
6. Update $U^h = U^h + \epsilon^h$.
7. Perform n_2 iterations with the relaxation scheme to eliminate errors introduced by the interpolation.
8. If the solution is not accurate enough, go back to step 1.

The Coarse Grid Defect Equation (3.61) is not, in general, a good approximation of the defect equation (3.60) unless the error ϵ^h is smooth. The error ϵ^h has components that can be resolved on the coarse grid and others that cannot be resolved. Figure 3.4 illustrates the *visible* (or *low frequency*) and *invisible* (or *high frequency*) error components on a 4-point grid. All error components with wavelength less than or equal to $2H$ (and wavenumber greater than or equal to $I\pi/2$) are invisible on the coarse mesh. If the *invisible* components have significant amplitude, the CGDE will yield an inaccurate estimate of the error.

The way to solve the CGDE also has not been specified yet. Although any iterative scheme can be used, (SOR, for example) one can also employ CGC again, with a coarser grid still (with spacing $2H$). Techniques that perform this procedure recursively are called *multigrid methods*.

3.7.2 The restriction and interpolation operators.

Various choices are possible for the restriction operator I_h^H. The simplest possible one is the *injection* or *unweighted restriction* (see Figure 3.5 for the node-numbering convention):

$$U_{i,j}^H = \left(I_h^H U^h\right)_{i,j} = U_{2i,2j}^h, \tag{3.62}$$

which can be written in matrix form as

$$U_{i,j}^H = \left(I_h^H U^h\right)_{i,j} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} U^h \quad . \tag{3.63}$$

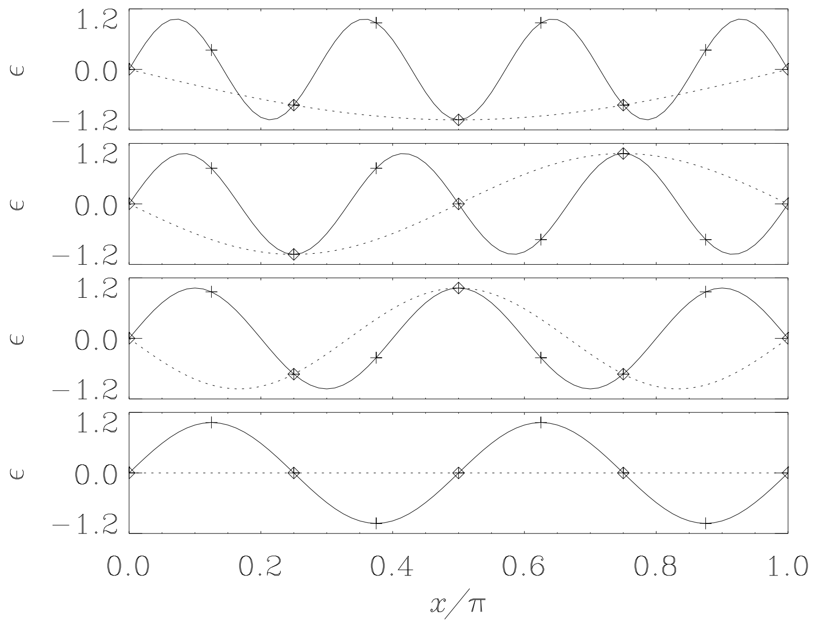

Figure 3.4: Visible and invisible error components. + 8-point grid; ∘ 4-point grid. From top to bottom: $\sin(-x)$ and $\sin(7x)$; $\sin(-2x)$ and $\sin(6x)$; $\sin(-3x)$ and $\sin(5x)$; $\sin(-0x)$ and $\sin(4x)$.

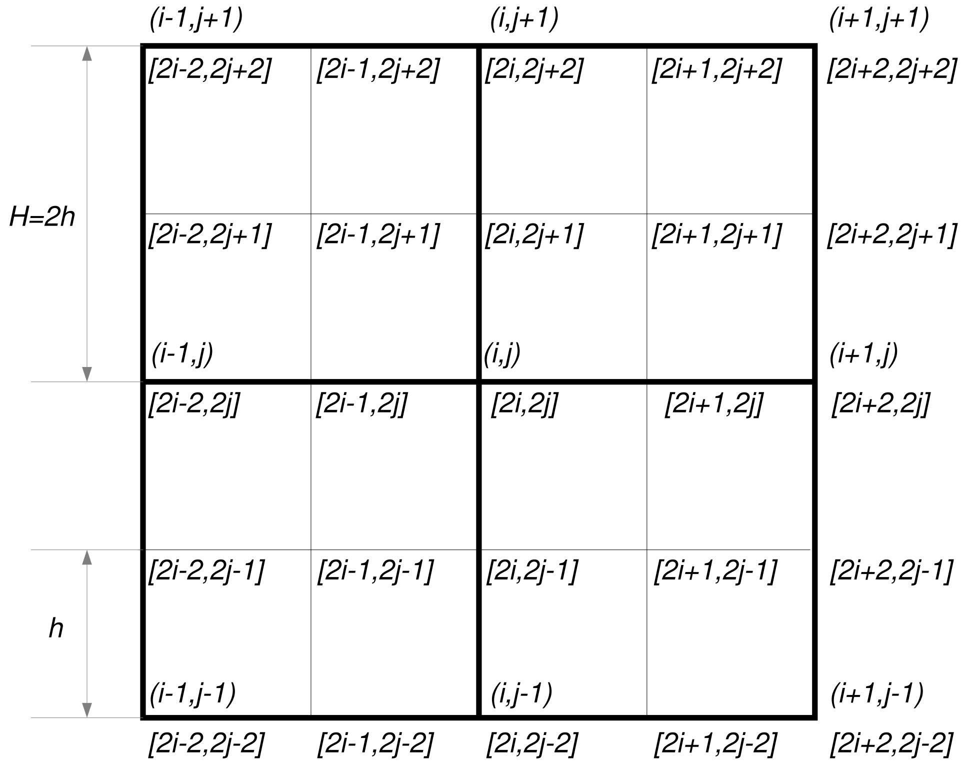

Figure 3.5: Node numbering on the coarse (parentheses) and fine (square brackets) grids.

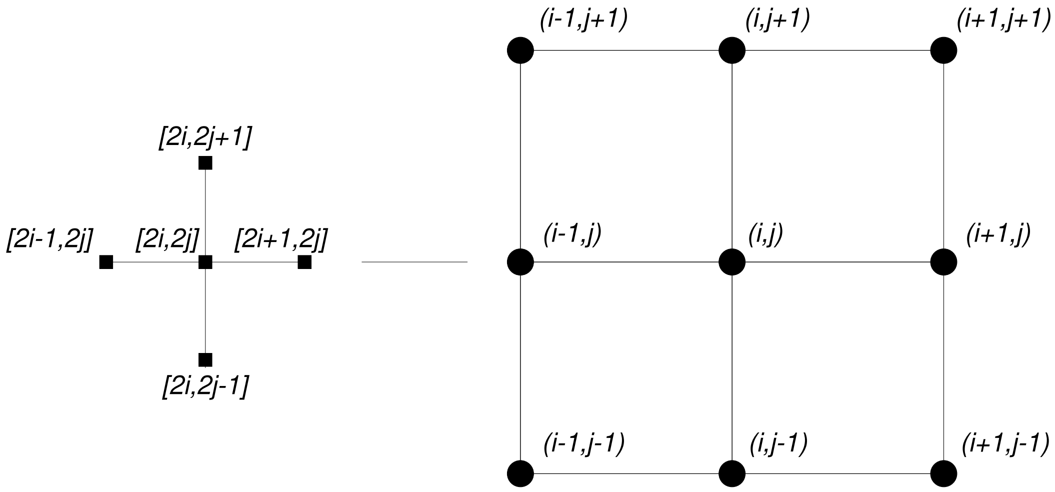

Figure 3.6: Coarse and fine grid stencils.

Other possible choices are the *half-weighted restriction*

$$\begin{aligned} U^H_{i,j} &= \left(I^H_h U^h\right)_{i,j} = \frac{1}{2} U^h_{2i,2j} + \frac{1}{8}\left(U^h_{2i+1,2j} + U^h_{2i-1,2j} + U^h_{2i,2j+1} + U^h_{2i,2-1}\right) \\ &= \frac{1}{8}\begin{pmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{pmatrix} U^h, \end{aligned} \tag{3.64}$$

and the *full-weighted restriction*

$$U^H_{i,j} = \left(I^H_h U^h\right)_{i,j} = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix} U^h \quad . \tag{3.65}$$

The most commonly used interpolation operator I^h_H is the *bilinear interpolation*

$$\left\{ \begin{array}{lclcl} U^h_{2i,2j} &=& \left(I^h_H U^H\right)_{2i,2j} &=& U^H_{i,j} \\ U^h_{2i+1,2j} &=& \left(I^h_H U^H\right)_{2i+1,2j} &=& \frac{1}{2}\left(U^H_{i,j} + U^H_{i+1,j}\right) \\ U^h_{2i,2j+1} &=& \left(I^h_H U^H\right)_{2i,2j+1} &=& \frac{1}{2}\left(U^H_{i,j} + U^H_{i+1,j}\right) \\ U^h_{2i+1,2j+1} &=& \left(I^h_H U^H\right)_{2i+1,2j+1} &=& \frac{1}{4}\left(U^H_{i,j} + U^H_{i+1,j} + U^H_{i+1,j+1} + U^H_{i,j+1}\right). \end{array} \right. \tag{3.66}$$

It is recommended to define I^H_h as the adjoint operator of I^h_H: $I^H_h = c(I^h_H)^T$. If the bilinear interpolation is chosen for I^h_H, then $I^H_h = (I^h_H)^T$ is the full-weighted restriction.

3.7.3 The coarse-grid operator.

The coarse-grid operator L^H should be chosen consistently with the interpolation operator; this can be done by taking

$$\begin{aligned} L^H U^H &= I^H_h\left(L^h U^h\right) = I^H_h L^h\left(I^h_H U^H\right) \\ &= \left(I^H_h L^h I^h_H\right) U^H \end{aligned} \tag{3.67}$$

$$\Rightarrow L^H = I^H_h L^h I^h_H \tag{3.68}$$

This choice is often impractical, since the stencil of L^H may not correspond to that of L^h, depending on the choice of restriction and interpolation operators. If, for example, bilinear interpolation and half-weighted restriction are used, the standard 5-point stencil used on the fine grid maps into a 9-point stencil (Figure 3.6). Usually, L^H is chosen as the same operator as L^h, applied on the coarse grid.

3.7.4 The relaxation method.

As seen before, the CGDE is a good approximation of the defect equation (3.60) only in the error is smooth. Thus, the main purpose of the relaxation method used to obtain U^h should be to damp high frequency errors. The optimum relaxation scheme for a multigrid method is not the one that gives the fastest convergence, but the one that has the best smoothing properties.

Consider, for example, the Jacobi method with under-relaxation (damped Jacobi). It was found, in Section 6.4, that over-relaxation of the Jacobi method leads to amplification of the error, while under-relaxation gives slower convergence than the standard Jacobi method, whose convergence rate is in any case unsatisfactory. Although neither the Jacobi nor the damped Jacobi method is a very efficient iterative matrix inversion method, the damped Jacobi method can be an effective smoothing technique for a multigrid method. A characteristic of the Jacobi method, which was evident in Figure 3.1, is the fact that it damps the high-frequency components of the error faster than the low-frequency ones. This property can be exploited to make the method an efficient relaxation scheme for multigrid algorithms.

To apply the damped Jacobi technique in a multigrid method the relaxation scheme must be optimized to damp *high frequencies* very effectively. High frequencies are defined as those frequencies that are invisible on the coarser grid, *i.e.*, corresponding to eigenvectors with wavenumber $i\pi$, $j\pi$ such that $\max(|i\Delta x|,\ |j\Delta y|) \geq I/2$ (Figure 3.7). The eigenvalues of the Jacobi method are given by (3.19):

$$\lambda'_{i,j} = \frac{1}{2}\left[\cos(i\pi\Delta x) + \cos(j\pi\Delta y)\right]$$

while its eigenvectors are

$$\psi_{i,j} = 2\sin(i\pi x)\sin(j\pi y); \qquad (3.69)$$

for the damped Jacobi method the eigenvalues are

$$\lambda_{i,j} = \frac{\omega}{2}\left[\cos(i\pi\Delta x) + \cos(j\pi\Delta y)\right] + (1-\omega) \qquad (3.70)$$

and the eigenvectors are also given by (3.69).

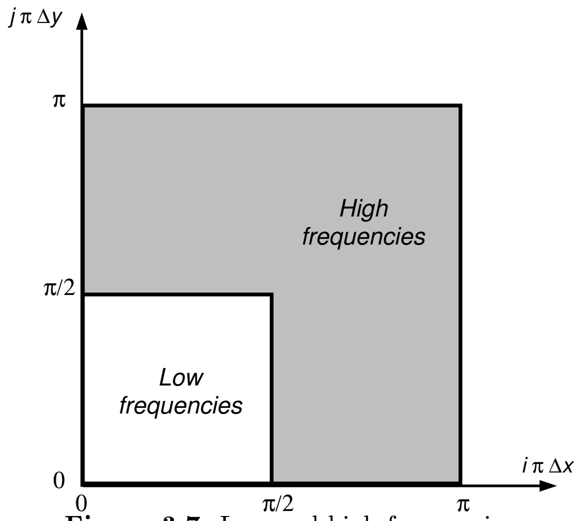

Figure 3.7: Low and high frequencies.

Over the range in Figure 4, $\lambda_{i,j}$ decreases as j increases for fixed i, and it also decreases as i increases for fixed j. The maximum eigenvalue must, therefore, be found at one of the corners, where the values of λ are:

$$\begin{cases} \lambda(\pi/2,0) &= \frac{\omega}{2}(\cos\pi/2+1)+(1-\omega) &= 1-\omega/2 \\ \lambda(\pi,0) &= \frac{\omega}{2}(\cos\pi+1)+(1-\omega) &= 1-\omega \\ \lambda(0,\pi/2) &= \frac{\omega}{2}(1+\cos\pi/2)+(1-\omega) &= 1-\omega/2 \\ \lambda(\pi/2,\pi/2) &= \frac{\omega}{2}2\cos\pi/2+(1-\omega) &= 1-\omega \\ \lambda(0,\pi) &= \frac{\omega}{2}(1+\cos\pi)+(1-\omega) &= 1-\omega \\ \lambda(\pi,\pi) &= \frac{\omega}{2}2\cos\pi+(1-\omega) &= 1-2\omega. \end{cases} \qquad (3.71)$$

Thus, $\lambda_{min} = 1-2\omega$ and $\lambda_{max} = 1-\omega/2$ (since $\omega \leq 1$) and $\omega_{opt} = 4/5$.

Next, we define the *smoothing parameter* μ as the maximum eigenvalue (in absolute value) over the high frequencies:

$$\mu = \max_{1/2\leq\max(|ih|,\ |jh|)\leq 1} |\lambda_{i,j}|\,. \qquad (3.72)$$

Small values of μ indicate good damping characteristics of a relaxation scheme. For the damped Jacobi method, $\mu = 3/5$ if ω_{opt} is used. Although the damped Jacobi method converges very slowly, it effectively smooths out high frequency oscillations, and is an acceptable relaxation scheme for multigrid methods.

Other relaxation schemes can be analyzed in a similar manner. It can be shown, for example, that, for Gauss-Seidel relaxation, $\mu = 1/2$; for Gauss-Seidel line relaxation (GSLR), $\mu = 1/\sqrt{5}$.

Table 3.1: Logic of an FMG program

$h = K\Delta x$	Coarsest grid
$U^h = \left(L^h\right)^{-1} f^h$	Direct solution on coarsest grid
do while $h > \Delta x$:	Nested iteration
$h = h/2$	
$U^h = I^h_{2h} U^{2h}$	
multigrid$(h, n_1, n_2, f^h, U^h, L^h)$	
enddo	
$r^h = L^h U^h - f^h$	Evaluate defect on fine grid
do while $\lVert r^h \rVert < \delta$:	Execute until defect is small
multigrid$(h, n_1, n_2, f^h, U^h, L^h)$	
$r^h = L^h U^h - f^h$	Update defect on fine grid
enddo	
end	
procedure multigrid$(h, n_1, n_2, f^h, U^h, L^h)$	
if $h = Kh$ then	
$U^h = \left(L^h\right)^{-1} f^h$	Direct solution on coarsest grid
else	
do n_1 times: relax(U^h, f^h)	Eliminate high frequency errors
$U^{2h} = 0$	
$f^{2h} = I^{2h}_h (f^h - L^h U^h)$	Calculate defect on coarse grid
multigrid$(2h, n_1, n_2, f^{2h}, U^{2h}, L^{2h})$	Solve CGDE
$U^{2h} = U^h + I^h_{2h} U^{2h}$	
do n_2 times: relax(U^h, f^h)	Eliminate high frequency errors
endif	
end	

3.7.5 The Full Multigrid (FMG) Method.

The Full Multigrid (FMG) Method is a generalization of the CGC algorithm. In a FMG scheme a sequence of grids with spacing $\Delta x,\ 2\Delta x,\ 4\Delta x, \ldots, K\Delta x$ is used. At every grid level the CGDE is solved by a CGC scheme a coarser mesh. Direct inversion is used on the coarsest mesh to solve

$$L^{K\Delta x} u^{K\Delta x} = f^{K\Delta x}. \tag{3.73}$$

In a FMG scheme, furthermore, nested iteration is used to obtain the initial guess U^h on the fine grid. Nested iteration allows a more accurate evaluation of the initial guess on a given grid h by solving the problem $L^{2h}u^{2h} = f^{2h}$ on a coarser grid and interpolating the result on the finer one. The logic of a multigrid program with K grid levels is shown in Table 2.

3.7.6 Work estimates.

In general, multigrid methods require extra storage. In two dimensions, the storage required for the $2h$ grid is $1/4$ that required for the h grid. The total storage required will, therefore, be $(1+1/4+1/16+\ldots)I^2 \simeq 4I^2/3$. The storage required is only 33% greater than that required by a one-grid algorithm. In three dimensions the comparison is even better: only $8/7$ of the original storage is required.

It can be shown that the total work for one CGC iteration is proportional to the number of grid points, I^2. The total number of operations required to reduce the error to δ will be proportional to $I^2 \ln(1/\delta)$. For

Table 3.2: Work estimates for iterative methods

Method	Work
Jacobi	$\sim I^4 \ln I$
Gauss-Seidel	$\sim I^4 \ln I$
SOR (with ω_{opt})	$\sim I^3 \ln I$
ADI	$\sim I^2 \ln I$
Multigrid (CGC)	$\sim I^2 \ln I$
Multigrid (FMG)	$\sim I^2$

a FMG method the number of operations is reduced to I^2.

3.8 The conjugate gradient method

3.8.1 The quadratic form

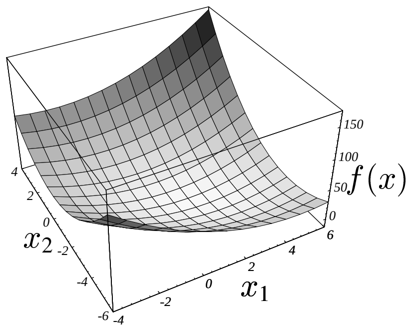

Figure 3.8: Graph of a quadratic form $f(\mathbf{x})$. The minimum point on this surface is the solution to $A\mathbf{x} = \mathbf{b}$. From [12].

Let A be a positive definite, symmetric matrix. Solution of

$$A\boldsymbol{\phi} = \mathbf{b} \tag{3.74}$$

is equivalent to minimization of the quadratic form (Fig. 3.8)

$$f = \frac{1}{2}\boldsymbol{\phi} \cdot A\boldsymbol{\phi} - \mathbf{b} \cdot \boldsymbol{\phi} + C. \tag{3.75}$$

Proof:

$$f = \frac{1}{2}\phi_i A_{ij} \phi_j - b_i \phi_i + C \tag{3.76}$$

$$\begin{aligned} \frac{\partial f}{\partial \phi_k} &= \frac{1}{2}\left(A_{kj}\phi_j + \phi_i A_{ij}\right) - b_k \\ &= \frac{1}{2}\left(A_{kj} + A_{jk}\right)\phi_j - b_k \\ &= A_{kj}\phi_j - b_k; \end{aligned} \tag{3.77}$$

requiring that $\partial f / \partial x_k = 0$ is equivalent to requiring that

$$A_{kj}\phi_j = b_k, \qquad Q.E.D. \tag{3.78}$$

If A is positive definite, $\partial f / \partial \phi_k = 0$ is a minimum, otherwise it can be something less desirable (Fig. 3.9).

3.8.2 Steepest Descent

To find the minimum, one can use the method of *Steepest Descent*: choose a starting guess $\boldsymbol{\phi}^{(0)}$ and move by some amount $\alpha^{(0)}$ in the direction of the negative gradient:

$$-\frac{\partial f}{\partial \phi_k} = -A_{kj}\phi_j^{(0)} + b_k = r_k^{(0)}, \tag{3.79}$$

where $\mathbf{r}^{(n)}$ is the *residual* at the n-th iteration. The new value of $\boldsymbol{\phi}$ is, therefore, given by:

$$\boldsymbol{\phi}^{(1)} = \boldsymbol{\phi}^{(0)} + \alpha^{(0)}\mathbf{r}^{(0)}. \tag{3.80}$$

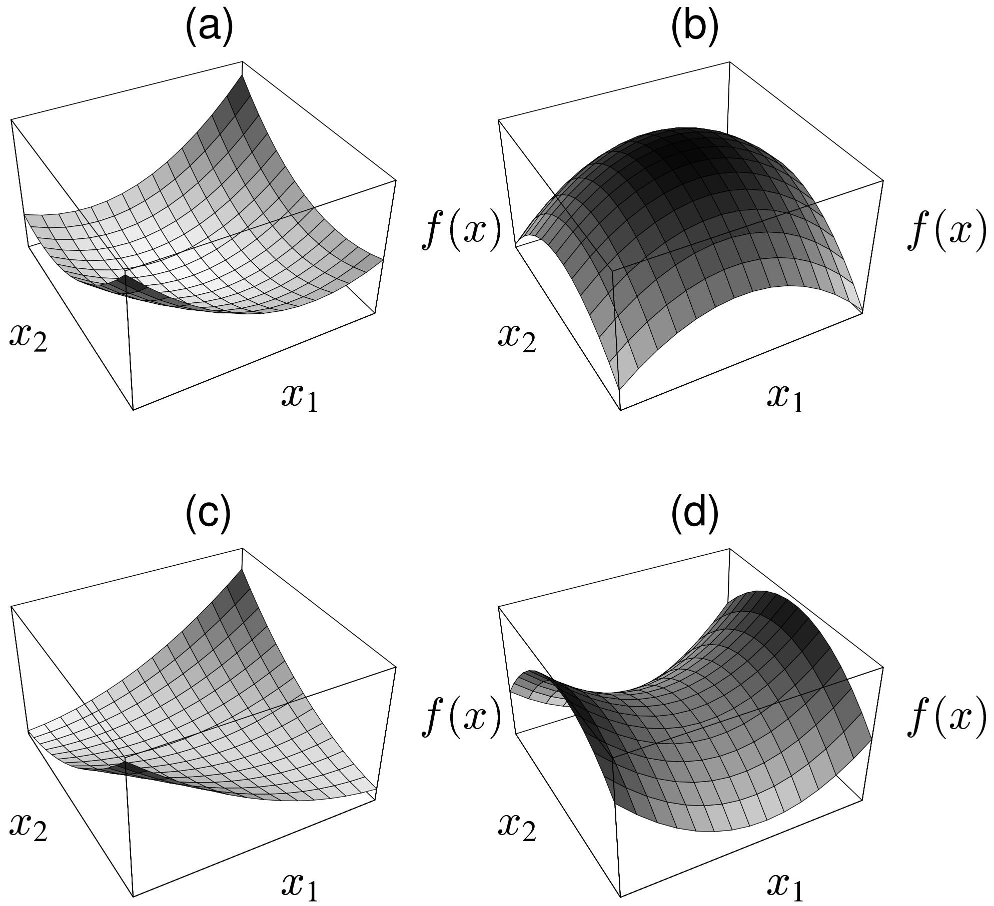

Figure 3.9: Quadratic form for (a) a positive-definite matrix; (b) a negative-definite matrix; (c) a singular positive-indefinite matrix (a line that runs through the bottom of the valley is the set of solutions); (d) an indefinite matrix (the solution is a saddle point). From [12].

To choose the optimal $\alpha^{(0)}$ (*i.e.*, to decide how far we should go in the $\mathbf{r}^{(0)}$ direction), find the local minimum of f along the line by setting $df/d\alpha^{(0)} = 0$:

$$\frac{d}{d\alpha^{(0)}}\left[f\left(\phi_j^{(1)}\right)\right] = \frac{\partial f}{\partial \phi_j^{(1)}}\frac{\partial \phi_j^{(1)}}{\partial \alpha^{(0)}} = \underbrace{\left[\phi_k^{(1)}A_{kj} - b_j\right]}_{=-r_j{}^{(1)}} r_j{}^{(0)} = -r_j{}^{(1)}r_j{}^{(0)} = 0; \tag{3.81}$$

thus, to optimize the first (and subsequent) iteration, we must reach a point where the new residual is orthogonal to the old one (i.e., to the direction in which we are moving). Returning to (3.81):

$$\begin{aligned}
\left[b_j - \phi_k^{(1)}A_{kj}\right] r_j{}^{(0)} &= 0,\\
\left[b_j - \left\{\phi_k^{(0)} + \alpha^{(0)}r_k{}^{(0)}\right\} A_{kj}\right] r_j{}^{(0)} &= 0,\\
\left[b_j - \phi_k^{(0)}A_{kj} - \alpha^{(0)}r_k{}^{(0)}A_{kj}\right] r_j{}^{(0)} &= 0,\\
\left[r_j{}^{(0)} - \alpha^{(0)}r_k{}^{(0)}A_{kj}\right] r_j{}^{(0)} &= 0,\\
r_j{}^{(0)}r_j{}^{(0)} - \alpha^{(0)}r_k{}^{(0)}A_{kj}r_j{}^{(0)} &= 0;
\end{aligned}$$

which yields

$$\alpha^{(0)} = \frac{r_j{}^{(0)}r_j{}^{(0)}}{r_k{}^{(0)}A_{kj}r_j{}^{(0)}} = \frac{\mathbf{r}^{(0)}\cdot\mathbf{r}^{(0)}}{\mathbf{r}^{(0)}\cdot A\mathbf{r}^{(0)}}, \tag{3.82}$$

with similar results for the following iterations. This generates the following recursive sequence for the method of Steepest Descent:

$$\begin{aligned}
&1. \quad \alpha^{(n)} = \frac{\mathbf{r}^{(n)}\cdot\mathbf{r}^{(n)}}{\mathbf{r}^{(n)}\cdot A\mathbf{r}^{(n)}}, && (3.83)\\
&2. \quad \boldsymbol{\phi}^{(n+1)} = \boldsymbol{\phi}^{(n)} + \alpha^{(n)}\mathbf{r}^{(n)}, && (3.84)\\
&3a. \quad \mathbf{r}^{(n+1)} = \mathbf{r}^{(n)} - \alpha^{(n)}A\mathbf{r}^{(n)} \qquad \text{or} && (3.85)\\
&3b. \quad \mathbf{r}^{(n+1)} = \mathbf{b} - A\boldsymbol{\phi}^{(n+1)} && (3.86)
\end{aligned}$$

which should be iterated until $||\mathbf{r}^{(n)}|| < \delta$ (with δ a small number). Notice that updating the residual according to step $3a$ decreases the number of vector products necessary. Since roundoff error tends to accumulate, however, it is desirable to recompute periodically the residual according to the definition (3.86) rather than the update (3.85). Figure 3.10 shows the convergence for an quadratic form shown before, and an initial guess $[-2,-2]^T$.

To find the convergence rate, express the error at iteration n in terms of the eigenvalues, λ_j, and normalized eigenvectors, $\boldsymbol{\psi}_j$, of the matrix A:

$$\boldsymbol{\varepsilon}^{(n)} = \boldsymbol{\phi}^{(n)} - \boldsymbol{\phi} = \sum_j C_j(\lambda_j)^n\boldsymbol{\psi}_j = \sum_j \xi_j^{(n)}\boldsymbol{\psi}_j. \tag{3.87}$$

Since the error satisfies $-A\boldsymbol{\varepsilon}^{(n)} = \mathbf{r}^{(n)}$, we have

$$\mathbf{r}^{(n)} = -\sum_j \xi_j^{(n)}A\boldsymbol{\psi}_j = -\sum_j \xi_j^{(n)}\lambda_j\boldsymbol{\psi}_j. \tag{3.88}$$

Also,

$$\boldsymbol{\varepsilon}^{(n+1)} = \boldsymbol{\varepsilon}^{(n)} + \alpha^{(n)}\mathbf{r}^{(n)} = \boldsymbol{\varepsilon}^{(n)} + \frac{\mathbf{r}^{(n)}\cdot\mathbf{r}^{(n)}}{\mathbf{r}^{(n)}\cdot A\mathbf{r}^{(n)}}; \tag{3.89}$$

however, from (3.88),

$$\mathbf{r}^{(n)}\cdot\mathbf{r}^{(n)} = \left(-\sum_j \xi_j^{(n)}\lambda_j\boldsymbol{\psi}_j\right)\cdot\left(-\sum_k \xi_k^{(n)}\lambda_k\boldsymbol{\psi}_k\right),$$

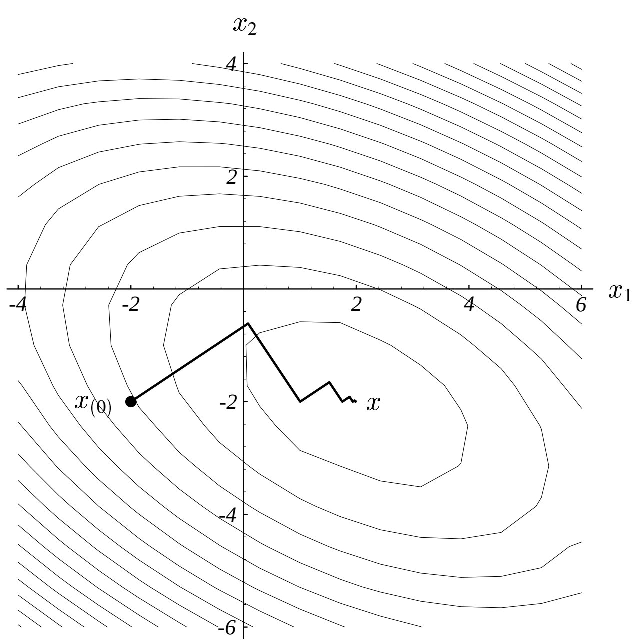

Figure 3.10: The method of Steepest Descent. From [12].

$$
\begin{aligned}
&= \sum_j \sum_k \xi_j^{(n)} \xi_k^{(n)} \lambda_j \lambda_k \boldsymbol{\psi}_j \cdot \boldsymbol{\psi}_k, \\
&= \sum_j {\xi_j^{(n)}}^2 {\lambda_j}^2 \qquad \text{by the orthonormality of the eigenvectors;} \qquad (3.90)
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{r}^{(n)} \cdot A\mathbf{r}^{(n)} &= \left(-\sum_j \xi_j^{(n)} \lambda_j \boldsymbol{\psi}_j\right) \cdot \left(-\sum_k \xi_k^{(n)} \lambda_k A \boldsymbol{\psi}_k\right), \\
&= \sum_j {\xi_j^{(n)}}^2 {\lambda_j}^3; \qquad (3.91)
\end{aligned}
$$

substituting (3.90) and (3.91) into (3.89) gives

$$
\boldsymbol{\varepsilon}^{(n+1)} = \boldsymbol{\varepsilon}^{(n)} + \frac{\sum_j {\xi_j^{(n)}}^2 {\lambda_j}^2}{\sum_k {\xi_k^{(n)}}^2 {\lambda_k}^3} \mathbf{r}^{(n)}. \qquad (3.92)
$$

If all the eigenvalues are equal,

$$
\boldsymbol{\varepsilon}^{(n+1)} = \boldsymbol{\varepsilon}^{(n)} + \frac{1}{\lambda} \left(-\sum_j \xi_j^{(n)} \lambda \boldsymbol{\psi}_j\right) = \boldsymbol{\varepsilon}^{(n)} - \boldsymbol{\varepsilon}^{(n)} = 0; \qquad (3.93)
$$

which implies that SD converges in a single iteration. If there is a range of eigenvalues, however, the convergence is slowed down (Fig. 3.11). This is especially true if $f(x)$ forms a narrow valley, the solution may start bouncing back and forth between the sides of the valley and not go very far towards convergence.

If we define the "energy norm"

$$
||\mathbf{g}||_A^2 = \mathbf{g} \cdot A\mathbf{g}, \qquad (3.94)
$$

it is possible to find the following bound for the convergence rate:

$$
||\boldsymbol{\varepsilon}^{(n)}||_A^2 \leq \left(\frac{\kappa - 1}{\kappa + 1}\right)^n ||\boldsymbol{\varepsilon}^{(0)}||_A^2, \qquad (3.95)
$$

where $\kappa = |\lambda_{max}|/|\lambda_{min}|$ is the condition number. In typical problems, $\kappa \sim o(N)$, and convergence is very slow (Fig. 3.12).

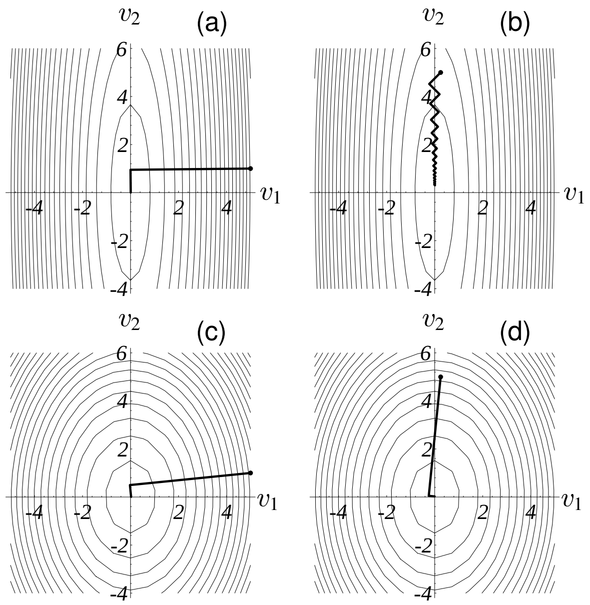

Figure 3.11: Convergence of the method of Steepest Descent. (a) and (b): large κ; (c) and (d): small κ. From [12].

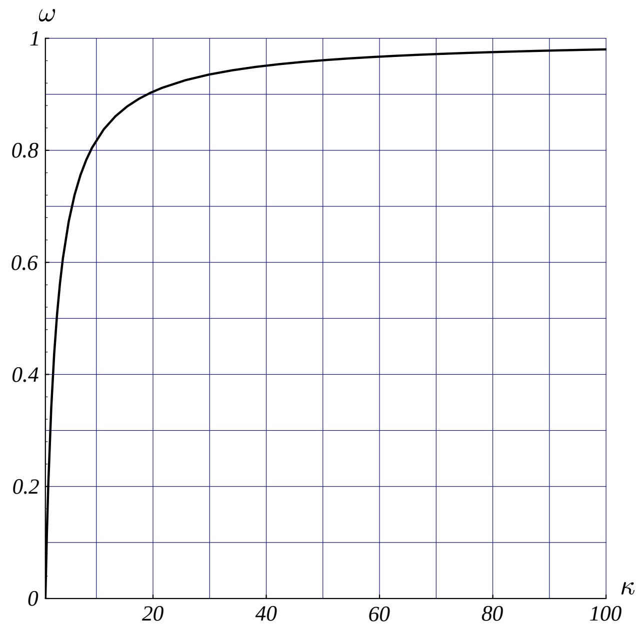

Figure 3.12: Convergence of the method of Steepest Descent as a function of the condition number κ. From [12].

3.8.3 Conjugate directions

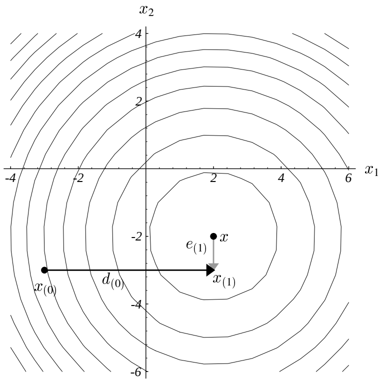

Figure 3.13: The method of orthogonal directions. From [12].

As seen before, SD fails if κ is large, and the surface f forms a narrow valley; SD just goes from one side of the valley to the other, going over the same directions over and over again. An improvement is obtained if, instead of always moving in the direction of the negative gradient, we move in an arbitrary direction $\mathbf{d}^{(n)}$, with the only constraint that $\mathbf{d}^{(n)}$ is orthogonal to all other directions: $\mathbf{d}^{(n)} \cdot \mathbf{d}^{(l)} = 0$ unless $l = n$. Suppose that we represent the initial error, $\boldsymbol{\varepsilon}^{(0)}$, in terms of its components along the directions $\mathbf{d}^{(n)}$ (which are a linearly independent set, therefore suitable as a basis; the representation is not independent of the set $\mathbf{d}^{(n)}$ chosen, but in all cases one requires no more than N components to describe $\boldsymbol{\varepsilon}^{(0)}$). We can move in each direction by the optimal amount, $\boldsymbol{\phi}^{(n+1)} = \boldsymbol{\phi}^{(n)} + \alpha^{(n)}\mathbf{d}^{(n)}$, by forcing $\mathbf{d}^{(n)}$ to be orthogonal to $\boldsymbol{\varepsilon}^{(n+1)}$; the procedure will converge in N steps at most, since each iteration annihilates one component of the error. Although each step is not optimal (unlike SD), the whole procedure is.

To determine $\alpha^{(n)}$, exploit the property that $\mathbf{d}^{(n)} \cdot \boldsymbol{\varepsilon}^{(n+1)} = 0$:

$$\mathbf{d}^{(n)} \cdot \left[\boldsymbol{\varepsilon}^{(n)} + \alpha^{(n)}\mathbf{d}^{(n)}\right] = 0, \tag{3.96}$$

which yields the optimum $\alpha^{(n)}$ as

$$\alpha^{(n)} = -\frac{\mathbf{d}^{(n)} \cdot \boldsymbol{\varepsilon}^{(n)}}{\mathbf{d}^{(n)} \cdot \mathbf{d}^{(n)}}. \tag{3.97}$$

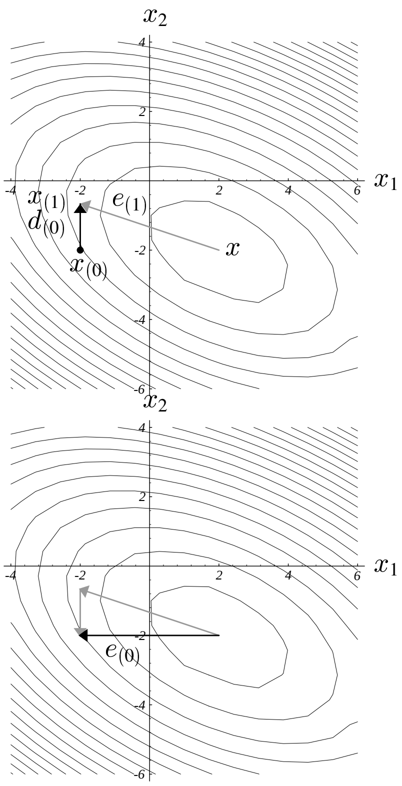

Figure 3.14: The method of conjugate directions. From [12].

This expression is, of course, useless, since $\boldsymbol{\varepsilon}^{(n)}$ is unknown. We can obtain a useful one if we require that the directions $\mathbf{d}^{(n)}$, instead of being orthogonal, be A-orthogonal (or *conjugate*): $\mathbf{d}^{(n)} \cdot A\mathbf{d}^{(l)} = 0$ unless $l = n$. This constraint gives $\mathbf{d}^{(n)} \cdot A\boldsymbol{\varepsilon}^{(n+1)} = 0$, which can be rewritten as:

$$\begin{aligned} \mathbf{d}^{(n)} \cdot A\left[\boldsymbol{\varepsilon}^{(n)} + \alpha^{(n)}\mathbf{d}^{(n)}\right] &= 0, & (3.98)\\ \mathbf{d}^{(n)} \cdot A\boldsymbol{\varepsilon}^{(n)} &= -\alpha^{(n)}\mathbf{d}^{(n)} \cdot A\mathbf{d}^{(n)}, & (3.99)\\ \mathbf{d}^{(n)} \cdot \left[-\mathbf{r}^{(n)}\right] &= -\alpha^{(n)}\mathbf{d}^{(n)} \cdot A\mathbf{d}^{(n)}, & (3.100) \end{aligned}$$

from which $\alpha^{(n)}$ can be obtained:

$$\alpha^{(n)} = \frac{\mathbf{r}^{(n)} \cdot \mathbf{d}^{(n)}}{\mathbf{d}^{(n)} \cdot A\mathbf{d}^{(n)}}. \tag{3.101}$$

To prove that this method gives a solution in N steps, express the initial error in terms of the basis vectors $\mathbf{d}^{(n)}$:

$$\boldsymbol{\varepsilon}^{(0)} = \sum_{n=0}^{N-1} \delta_n\, \mathbf{d}^{(n)}, \tag{3.102}$$

where δ_n are constants. Multiplying both sides by $\mathbf{d}^{(l)} \cdot A$ yields

$$\mathbf{d}^{(l)} \cdot A\boldsymbol{\varepsilon}^{(0)} = \sum_{n=0}^{N-1} \delta_n\, \mathbf{d}^{(l)} \cdot A\mathbf{d}^{(n)} = \delta_n\, \mathbf{d}^{(l)} \cdot A\mathbf{d}^{(l)}, \tag{3.103}$$

by the conjugation of the search directions; the expression above yields

$$\delta_n = \frac{\mathbf{d}^{(n)} \cdot A\boldsymbol{\varepsilon}^{(0)}}{\mathbf{d}^{(n)} \cdot A\mathbf{d}^{(n)}} = \frac{\mathbf{d}^{(n)} \cdot A\left[\boldsymbol{\varepsilon}^{(0)} + \sum_{l=0}^{n-1} \alpha^{(l)}\mathbf{d}^{(l)}\right]}{\mathbf{d}^{(n)} \cdot A\mathbf{d}^{(n)}} = \frac{\mathbf{d}^{(n)} \cdot A\boldsymbol{\varepsilon}^{(n)}}{\mathbf{d}^{(n)} \cdot A\mathbf{d}^{(n)}}; \tag{3.104}$$

since, from (3.99),

$$\alpha^{(n)} = -\frac{\mathbf{d}^{(n)} \cdot A\varepsilon^{(n)}}{\mathbf{d}^{(n)} \cdot A\mathbf{d}^{(n)}} = -\delta_n, \tag{3.105}$$

we can rewrite (3.103) as

$$\varepsilon^{(0)} = -\sum_{n=0}^{N-1} \alpha^{(n)}\mathbf{d}^{(n)}; \tag{3.106}$$

observing that

$$\begin{aligned}
\varepsilon^{(n)} &= \varepsilon^{(n-1)} + \alpha^{(n-1)}\mathbf{d}^{(n-1)} \\
&= \varepsilon^{(n-2)} + \alpha^{(n-2)}\mathbf{d}^{(n-2)} + \alpha^{(n-1)}\mathbf{d}^{(n-1)} \\
&= \varepsilon^{(0)} + \sum_{l=0}^{n-1} \alpha^{(l)}\mathbf{d}^{(l)} \\
&= -\sum_{l=0}^{N-1} \alpha^{(l)}\mathbf{d}^{(l)} + \sum_{l=0}^{n-1} \alpha^{(l)}\mathbf{d}^{(l)} \\
&= -\sum_{l=n}^{N-1} \alpha^{(l)}\mathbf{d}^{(l)} = \sum_{l=n}^{N-1} \delta_n\mathbf{d}^{(l)}.
\end{aligned} \tag{3.107}$$

we conclude that each iteration cuts away one component of the error, and in N iterations we have convergence.

The conjugate directions can be found by the process known as *Gramm-Schmidt Conjugation*: consider a set of linearly independent vectors $\mathbf{u}^{(n)}$; let:

$$\mathbf{d}^{(0)} = \mathbf{u}^{(0)}; \qquad \mathbf{d}^{(n)} = \mathbf{u}^{(n)} + \sum_{l=0}^{n-1} \beta_{nl}\, \mathbf{d}^{(l)}, \tag{3.108}$$

where the constants β_{nl} are defined only for $l < n$. By orthogonality,

$$\mathbf{d}^{(l)} \cdot A\mathbf{d}^{(n)} = \left[\mathbf{u}^{(l)} + \sum_{j=0}^{l-1} \beta_{lj}\, \mathbf{d}^{(j)}\right] \cdot A\mathbf{d}^{(n)} = \mathbf{u}^{(l)} \cdot A\mathbf{d}^{(n)} + \sum_{j=0}^{l-1} \beta_{lj}\, \mathbf{d}^{(j)} \cdot A\mathbf{d}^{(n)}; \tag{3.109}$$

three cases can be distinguished:

$$\begin{aligned}
0 &= \mathbf{u}^{(l)} \cdot A\mathbf{d}^{(n)} && \text{for } l < n; \\
\mathbf{d}^{(n)} \cdot A\mathbf{d}^{(n)} &= \mathbf{u}^{(n)} \cdot A\mathbf{d}^{(n)} && \text{for } l = n; \\
0 = \mathbf{u}^{(l)} \cdot A\mathbf{d}^{(n)} &+ \beta_{ln}\, \mathbf{d}^{(n)} \cdot A\mathbf{d}^{(n)} && \text{for } l > n;
\end{aligned}$$

the third of the above yields

$$\beta_{ln} = -\frac{\mathbf{u}^{(l)} \cdot A\mathbf{d}^{(n)}}{\mathbf{d}^{(n)} \cdot A\mathbf{d}^{(n)}}. \tag{3.110}$$

3.8.4 Conjugate Gradients

The limitation of this process (the fact that to determine $\mathbf{d}^{(n)}$ one requires the previous $n-1$ directions) can be avoided by an intelligent choice of the basis directions $\mathbf{u}^{(n)}$: let $\mathbf{u}^{(n)} = \mathbf{r}^{(n)}$. Then

$$\beta_{ln} = -\frac{\mathbf{r}^{(l)} \cdot A\mathbf{d}^{(n)}}{\mathbf{d}^{(n)} \cdot A\mathbf{d}^{(n)}}; \tag{3.111}$$

the use of the residuals as search directions has some interesting properties: first of all notice that, from (3.107)

$$-\mathbf{d}^{(n)} \cdot A\varepsilon^{(m)} = \mathbf{d}^{(n)} \cdot \mathbf{r}^{(m)} = -\sum_{l=m}^{N-1} \delta_l \mathbf{d}^{(n)} \cdot A\mathbf{d}^{(l)} \tag{3.112}$$

and, hence, $\mathbf{d}^{(n)} \cdot \mathbf{r}^{(m)} = 0$ for $m > n$. Also, from (3.108),

$$\mathbf{d}^{(n)} \cdot \mathbf{r}^{(m)} = \mathbf{u}^{(n)} \cdot \mathbf{r}^{(m)} + \sum_{l=0}^{n-1} \beta_{nl} \mathbf{d}^{(l)} \cdot \mathbf{r}^{(m)}. \tag{3.113}$$

However, for $m > n$ both the LHS and the summation are zero, and we conclude that $\mathbf{u}^{(n)} \cdot \mathbf{r}^{(m)} = 0$ for $m > n$. If the residual are the search directions, they are orthogonal to each other: $\mathbf{r}^{(n)} \cdot \mathbf{r}^{(m)} = 0$ for $m \neq n$.

Notice, furthermore, that

$$\begin{aligned}
&\mathbf{r}^{(n)} = -A\boldsymbol{\varepsilon}^{(n)} = -A\left[\boldsymbol{\varepsilon}^{(n-1)} + \alpha^{(n-1)}\mathbf{d}^{(n-1)}\right] = \mathbf{r}^{(n-1)} - \alpha^{(n-1)}A\mathbf{d}^{(n-1)} \\
&\mathbf{r}^{(l)} \cdot \mathbf{r}^{(n+1)} = \mathbf{r}^{(l)} \cdot \mathbf{r}^{(n)} - \alpha^{(n)}\mathbf{r}^{(l)} \cdot A\mathbf{d}^{(n)} \qquad \text{which yields} \\
&\alpha^{(n)}\mathbf{r}^{(l)} \cdot A\mathbf{d}^{(n)} = \mathbf{r}^{(l)} \cdot \mathbf{r}^{(n)} - \mathbf{r}^{(l)} \cdot \mathbf{r}^{(n+1)}.
\end{aligned}$$

For the only cases of interest ($l > n$), since the residuals are orthogonal,

$$\begin{aligned}
\text{for } l = n+1 \qquad & \alpha^{(n)}\mathbf{r}^{(l)} \cdot A\mathbf{d}^{(n)} = \alpha^{(l-1)}\mathbf{r}^{(l)} \cdot A\mathbf{d}^{(l-1)} = -\mathbf{r}^{(l)} \cdot \mathbf{r}^{(l)}; \\
\text{otherwise} \qquad & \alpha^{(n)}\mathbf{r}^{(l)} \cdot A\mathbf{d}^{(n)} = 0.
\end{aligned}$$

Thus $\beta_{ln} \neq 0$ only for $l = n+1$; letting $\beta_{l,l-1} = \beta^{(l)}$, and recalling (3.101), we obtain:

$$\begin{aligned}
\beta^{(l)} &= \frac{1}{\alpha^{(l-1)}} \frac{\mathbf{r}^{(l)} \cdot \mathbf{r}^{(l)}}{\mathbf{d}^{(l-1)} \cdot A\mathbf{d}^{(l-1)}} \\
&= \frac{\mathbf{d}^{(l-1)} \cdot A\mathbf{d}^{(l-1)}}{\mathbf{r}^{(l-1)} \cdot \mathbf{d}^{(l-1)}} \frac{\mathbf{r}^{(l)} \cdot \mathbf{r}^{(l)}}{\mathbf{d}^{(l-1)} \cdot A\mathbf{d}^{(l-1)}} \\
&= \frac{\mathbf{r}^{(l)} \cdot \mathbf{r}^{(l)}}{\mathbf{r}^{(l-1)} \cdot \mathbf{d}^{(l-1)}}.
\end{aligned}$$

However, from (3.112),

$$\begin{aligned}
\mathbf{r}^{(n)} \cdot \mathbf{d}^{(n)} &= \mathbf{u}^{(n)} \cdot \mathbf{r}^{(n)} + \sum_{k=0}^{n-1} \beta_{nk}\mathbf{d}^{(k)} \cdot \mathbf{r}^{(n)} \\
&= \mathbf{u}^{(n)} \cdot \mathbf{r}^{(n)} - \sum_{k=0}^{n-1} \beta_{nk}\mathbf{d}^{(k)} \cdot A\boldsymbol{\varepsilon}^{(n)};
\end{aligned}$$

and, from (3.113), $\mathbf{d}^{(k)} \cdot A\boldsymbol{\varepsilon}^{(n)} = 0$ when $k < n$, by A-orthogonality. Thus, $\mathbf{r}^{(n)} \cdot \mathbf{d}^{(n)} = \mathbf{u}^{(n)} \cdot \mathbf{r}^{(n)} = \mathbf{r}^{(n)} \cdot \mathbf{r}^{(n)}$, to yield:

$$\beta^{(l)} = \frac{\mathbf{r}^{(l)} \cdot \mathbf{r}^{(l)}}{\mathbf{r}^{(l-1)} \cdot \mathbf{r}^{(l-1)}}. \tag{3.114}$$

This generates the following recursive sequence for the method of Conjugate Gradients:

$$\begin{aligned}
&0. \quad \mathbf{d}^{(0)} = \mathbf{r}^{(0)} = \mathbf{b} - A\boldsymbol{\phi}^{(0)} && (3.115) \\
&1. \quad \alpha^{(n)} = \frac{\mathbf{r}^{(n)} \cdot \mathbf{d}^{(n)}}{\mathbf{d}^{(n)} \cdot A\mathbf{d}^{(n)}}, && (3.116) \\
&2. \quad \boldsymbol{\phi}^{(n+1)} = \boldsymbol{\phi}^{(n)} + \alpha^{(n)}\mathbf{d}^{(n)}, && (3.117) \\
&3a. \quad \mathbf{r}^{(n+1)} = \mathbf{r}^{(n)} - \alpha^{(n)}A\mathbf{d}^{(n)} \qquad \text{or} && (3.118) \\
&3b. \quad \mathbf{r}^{(n+1)} = \mathbf{b} - A\boldsymbol{\phi}^{(n+1)} && (3.119) \\
&4. \quad \beta^{(n+1)} = \frac{\mathbf{r}^{(n+1)} \cdot \mathbf{r}^{(n+1)}}{\mathbf{r}^{(n)} \cdot \mathbf{r}^{(n)}} && (3.120) \\
&5. \quad \mathbf{d}^{(n+1)} = \mathbf{r}^{(n+1)} + \beta^{(n+1)}\mathbf{d}^{(n)}. && (3.121)
\end{aligned}$$

Steps 1 through 5 are repeated until convergence. As before, the residual can be updated via step $3a$, but should be periodically recomputed using step $3b$ instead.

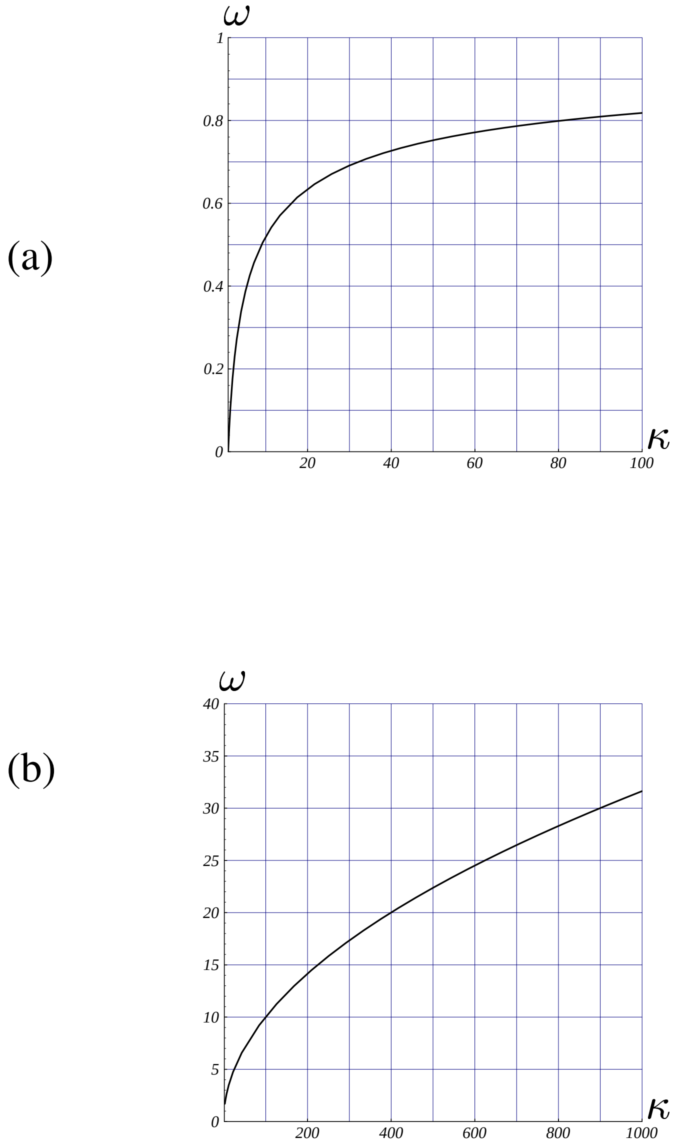

Figure 3.15: (a) Convergence of the Conjugate Gradient Method; (b) number of iterations of SD required to match one iteration of CG. From [12].

In theory, the method of Conjugate Gradients converges in less than N iterations. In practice, the accumulation of roundoff error makes it more useful as an iterative scheme; the convergence rate is bounded by

$$||\varepsilon^{(n)}||_A^2 \le \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^n ||\varepsilon^{(0)}||_A^2, \tag{3.122}$$

which is much faster than the method of Steepest Descent (Fig. 3.15).

3.8.5 Preconditioning

Preconditioning consists of solving the related problem $M^{-1}A\boldsymbol{\phi} = M^{-1}\mathbf{b}$ instead of the original one. If $\kappa(M^{-1}A) < \kappa(A)$, this is clearly a good idea. One can then rewrite the preconditioned system as

$$
\begin{array}{lll}
0a. & \mathbf{r}^{(0)} = \mathbf{b} - A\boldsymbol{\phi}^{(0)} & (3.123)\\
0b. & \mathbf{d}^{(0)} = M^{-1}\mathbf{r}^{(0)} & (3.124)\\
1. & \alpha^{(n)} = \dfrac{\mathbf{r}^{(n)} \cdot M^{-1}\mathbf{r}^{(n)}}{\mathbf{d}^{(n)} \cdot A\mathbf{d}^{(n)}}, & (3.125)\\
2. & \boldsymbol{\phi}^{(n+1)} = \boldsymbol{\phi}^{(n)} + \alpha^{(n)}\mathbf{d}^{(n)}, & (3.126)\\
3a. & \mathbf{r}^{(n+1)} = \mathbf{r}^{(n)} - \alpha^{(n)}A\mathbf{d}^{(n)} \qquad \text{or} & (3.127)\\
3b. & \mathbf{r}^{(n+1)} = \mathbf{b} - A\boldsymbol{\phi}^{(n+1)} & (3.128)\\
4. & \beta^{(n+1)} = \dfrac{\mathbf{r}^{(n+1)} \cdot M^{-1}\mathbf{r}^{(n+1)}}{\mathbf{r}^{(n)} \cdot M^{-1}\mathbf{r}^{(n)}} & (3.129)\\
5. & \mathbf{d}^{(n+1)} = M^{-1}\mathbf{r}^{(n+1)} + \beta^{(n+1)}\mathbf{d}^{(n)}. & (3.130)
\end{array}
$$

This procedure can be rewritten using an auxiliary vector $\mathbf{z}^{(n)} = M^{-1}\mathbf{r}^{(n)}$, as follows:

$$
\begin{array}{lll}
0a. & \mathbf{r}^{(0)} = \mathbf{b} - A\boldsymbol{\phi}^{(0)} & (3.131)\\
0b. & \mathbf{z}^{(0)} = M^{-1}\mathbf{r}^{(0)} & (3.132)\\
0c. & \mathbf{d}^{(0)} = \mathbf{z}^{(0)} & (3.133)\\
1. & \alpha^{(n)} = \dfrac{\mathbf{r}^{(n)} \cdot \mathbf{z}^{(n)}}{\mathbf{d}^{(n)} \cdot A\mathbf{d}^{(n)}}, & (3.134)\\
2. & \boldsymbol{\phi}^{(n+1)} = \boldsymbol{\phi}^{(n)} + \alpha^{(n)}\mathbf{d}^{(n)}, & (3.135)\\
3a. & \mathbf{r}^{(n+1)} = \mathbf{r}^{(n)} - \alpha^{(n)}A\mathbf{d}^{(n)} \qquad \text{or} & (3.136)\\
3b. & \mathbf{r}^{(n+1)} = \mathbf{b} - A\boldsymbol{\phi}^{(n+1)} & (3.137)\\
4. & \text{solve } M\mathbf{z}^{(n+1)} = \mathbf{r}^{(n+1)} & (3.138)\\
5. & \beta^{(n+1)} = \dfrac{\mathbf{r}^{(n+1)} \cdot \mathbf{z}^{(n+1)}}{\mathbf{r}^{(n)} \cdot \mathbf{z}^{(n)}} & (3.139)\\
6. & \mathbf{d}^{(n+1)} = \mathbf{z}^{(n+1)} + \beta^{(n+1)}\mathbf{d}^{(n)}. & (3.140)
\end{array}
$$

This procedure requires the solution of the system $M\mathbf{z}^{(n)} = \mathbf{r}^{(n)}$. The matrix M should, therefore, be easy to invert. Typically, it is of the form $M = LU$, where L and U are respectively a lower triangular and an upper triangular matrix.

Write the i,j element of the product $A\boldsymbol{\phi}$ as

$$(A\boldsymbol{\phi})_{i,j} = W_{i,j}\phi_{i,j-1} + S_{i,j}\phi_{i-1,j} + P_{i,j}\phi_{i,j} + N_{i,j}\phi_{i+1,j} + E_{i,j}\phi_{i,j+1}. \tag{3.141}$$

The simplest preconditionser (*Jacobi preconditioning*) uses a diagonal matrix M whose elements are the reciprocals of $P_{i,j}$. This is equivalent to stretching the quadratic form along one of the coordinate axes, which is effective in making it more cylindrical (and make CG converge in one iteration only) only if the eigenvectors are nearly aligned with the axes.

Better preconditioners are based on *incomplete factorizations* of the matrix A. A factorization $M = LU$ is said to be *incomplete* if some non-zero elements of L and U are neglected. The accuracy of the factorization (i.e., how well $M^{-1} = (LU)^{-1}$ approximates A^{-1}) determines the effectiveness of the preconditioner.

Two simple preconditioners (see [3]) are the RS5 and LU5 preconditioners. The incomplete LU5 factorization has L identical to the lower triangular part of A, and U chosen in such a way that the super-diagonal elements of LU match those of A. The incomplete RS5 factorization is similar, but the diagonal term is modified so that the row-sums of LU are equal to those of A.

Let

$$(L\boldsymbol{\phi})_{i,j} = w_{i,j}\phi_{i,j-1} + s_{i,j}\phi_{i-1,j} + p_{i,j}\phi_{i,j}; \qquad (U\boldsymbol{\phi})_{i,j} = \phi_{i,j} + n_{i,j}\phi_{i+1,j} + e_{i,j}\phi_{i,j+1}. \tag{3.142}$$

The elements of L and U are given by

$$\begin{aligned} w_{i,j} &= W_{i,j} \\ s_{i,j} &= S_{i,j} \\ p_{i,j} &= P_{i,j} - w_{i,j}n_{i,j-1} - s_{i,j}e_{i-1,j} + \alpha\left[w_{i,j}e_{i,j-1} + g_{i,j}n_{i-1,j}\right] \\ n_{i,j} &= N_{i,j}/p_{i,j} \\ e_{i,j} &= E_{i,j}/p_{i,j}. \end{aligned}$$

For $\alpha = 0$ the LU5 factorization obtains, for $\alpha = 1$ the RS5. The former is better for the high frequencies, the latter for the high ones.

3.9 Comparison of iterative schemes

Work estimates for the methods discussed in the previous sections are shown in Table 3.2. Since the minimum error that can be achieved is equal to the truncation error, δ was set equal to Δx^2, so that $\log(1/\delta) \sim \log I$.

To perform a numerical comparison of several schemes, let $u_{ex} = \exp(-16x^2)\exp(-4y^2)$, and $f = \nabla^2 u_{ex}$. The equation

$$\nabla^2 u = f \tag{3.143}$$

was solved on the square domain $0 \le x \le 1$, $0 \le y \le 1$, with boundary conditions $u(x,0) = u_{ex}(x,0)$, $u(x,1) = u_{ex}(x,1)$, $u(0,y) = u_{ex}(0,y)$, $u(1,y) = u_{ex}(1,y)$, and the initial condition $u(x,0) = 0$ for $0 < x < 1$, $0 < y < 1$.

Second-order central differences were used, and the mesh contained 64^2 points. The following schemes were used to solve (3.143):

1. SOR with Chebychev acceleration (SOR). The optimum value of ω was used.
2. Coarse-grid correction algorithm (CGC) with Gauss-Seidel relaxation. Two relaxation steps were performed before and one after the solution of the Coarse-grid defect equation.
3. Conjugate Gradient.
4. Conjugate Gradient with Jacobi Preconditioning.
5. Conjugate Gradient with LU5 Preconditioning.

The L_2 norm of the error,

$$||\epsilon||_2 = \frac{1}{IJ}\left[\sum_i \sum_j (u_{i,j} - u_{ex,i.j})^2\right]^{1/2}, \tag{3.144}$$

and of the residual

$$||r||_2 = \frac{||A\mathbf{u} - \mathbf{f}||_2}{||A||_2||\mathbf{u}||_2 + ||f||_2} \tag{3.145}$$

are shown in Fig. 3.16 as a function of the CPU time used by the various methods. Various interesting points can be made regarding this figure: first, The preconditioned CG and the CGC schemes are the most effective ones. Jacobi preconditioning does not improve the convergence rate. The residual decreases rapidly by 5 orders of magnitude while the error is still significant. This shows that stopping the iterations before the residual has reached machine zero may be dangerous.

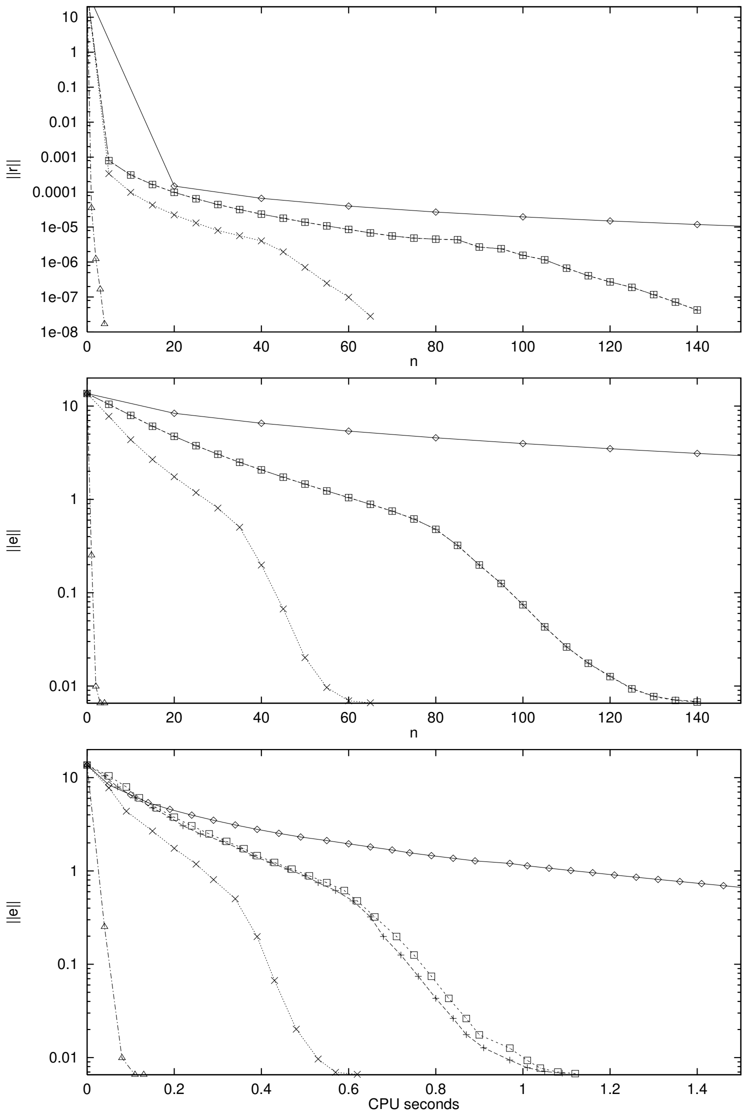

Figure 3.16: L_2 norm of the residual (top figure) and of the error (bottom two figures). $\diamond$ SOR; $+$ CG; $\square$ CG with Jacobi preconditioning; $\times$ CG with LU5 preconditioning; $\triangle$ CGC.

Bibliography

[1] D. A. Anderson, J. C. Tannehill and R. H. Pletcher. *Computational fluid mechanics and heat transfer*, 2nd Ed., Hemisphere, 1997.

[2] W.L. Briggs. *A Multigrid Tutorial*, SIAM, 1987.

[3] Canuto, C., Hussaini, M. Y., Quarteroni, A., and Zang, T. A. *Spectral methods in fluid dynamics* (Springer-Verlag, Heidelberg) (1988).

[4] J. Ferziger and M. Perić. "Computational Methods for Fluid Dynamics." Springer-Verlag, 1996.

[5] C. Hirsch. *Numerical computation of internal and external flows. Vol. 1.* Wiley, 1988.

[6] M. Khorrami, B. Singer, and R. Radeztsky Jr. "Reynolds Averaged Navier-Stokes Computations of a Flap Side-Edge Flow Field" *AIAA Paper 98-0768*, (1998).

[7] S.K. Lele. "Compact finite difference schemes with spectral-like resolution," *J. Comput. Phys.* **103**, 16, 1992.

[8] Liu, J., Piomelli, U., and Spalart, P. R. "Interaction between a spatially growing turbulent boundary layer and embedded streamwise vortices." *J. Fluid Mech.* **326**, 151 (1996).

[9] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vettering. *Numerical Recipes*, Cambridge University Press, 1986.

[10] R. D. Richtmyer. *Difference methods for initial-value problems*, Interscience Publishers Inc., New York, 1957.

[11] K. Stüben and U. Trottenberg. "Multigrid Methods: Fundamental Algorithms, Model Problem Analysis and Applications." in *Multigrid Methods* (W. Hackbusch and U. Trottenberg, eds.), Springer-Verlag, 1982.

[12] J.R. Shewchuk. "An introduction to the conjugate gradient method without the agonizing pain." School of Computer Science, Carnegie Mellon University, 1994.