FACR Method

The *best* way to solve equations of the form (19.4.28), including the constant coefficient problem (19.0.3), is a combination of Fourier analysis and cyclic reduction, the FACR method [3-6]. If at the rth stage of CR we Fourier analyze the equations of the form (19.4.32) along y, that is, with respect to the suppressed vector index, we will have a tridiagonal system in the x-direction for each y-Fourier mode:

$$\widehat{u}_{j-2^r}^k + \lambda_k^{(r)} \widehat{u}_j^k + \widehat{u}_{j+2^r}^k = \Delta^2 g_j^{(r)k} \tag{19.4.35}$$

Here $\lambda_k^{(r)}$ is the eigenvalue of $\mathbf{T}^{(r)}$ corresponding to the kth Fourier mode. For the equation (19.0.3), equation (19.4.5) shows that $\lambda_k^{(r)}$ will involve terms like $\cos(2\pi k/L) - 2$ raised to a power. Solve the tridiagonal systems for $\widehat{u}_j^k$ at the levels $j = 2^r, 2 \times 2^r, 4 \times 2^r, ..., J - 2^r$. Fourier synthesize to get the y-values on these x-lines. Then fill in the intermediate x-lines as in the original CR algorithm.

The trick is to choose the number of levels of CR so as to minimize the total number of arithmetic operations. One can show that for a typical case of a 128×128 mesh, the optimal level is $r = 2$; asymptotically, $r \to \log_2(\log_2 J)$.

A rough estimate of running times for these algorithms for equation (19.0.3) is as follows: The FFT method (in both x and y) and the CR method are roughly comparable. FACR with $r = 0$ (that is, FFT in one dimension and solve the tridiagonal equations by the usual algorithm in the other dimension) gives about a factor of two gain in speed. The optimal FACR with $r = 2$ gives another factor of two gain in speed.

CITED REFERENCES AND FURTHER READING:

Swartzrauber, P.N. 1977, *SIAM Review*, vol. 19, pp. 490–501. [1]

Buzbee, B.L, Golub, G.H., and Nielson, C.W. 1970, *SIAM Journal on Numerical Analysis*, vol. 7, pp. 627–656; see also *op. cit.* vol. 11, pp. 753–763. [2]

Hockney, R.W. 1965, *Journal of the Association for Computing Machinery*, vol. 12, pp. 95–113. [3]

Hockney, R.W. 1970, in *Methods of Computational Physics*, vol. 9 (New York: Academic Press), pp. 135–211. [4]

Hockney, R.W., and Eastwood, J.W. 1981, *Computer Simulation Using Particles* (New York: McGraw-Hill), Chapter 6. [5]

Temperton, C. 1980, *Journal of Computational Physics*, vol. 34, pp. 314–329. [6]

19.5 Relaxation Methods for Boundary Value Problems

As we mentioned in §19.0, relaxation methods involve splitting the sparse matrix that arises from finite differencing and then iterating until a solution is found.

There is another way of thinking about relaxation methods that is somewhat more physical. Suppose we wish to solve the elliptic equation

$$\mathcal{L}u = \rho \tag{19.5.1}$$

where $\mathcal{L}$ represents some elliptic operator and ρ is the source term. Rewrite the equation as a diffusion equation,

$$\frac{\partial u}{\partial t} = \mathcal{L}u - \rho \tag{19.5.2}$$

An initial distribution u *relaxes* to an equilibrium solution as $t \to \infty$. This equilibrium has all time derivatives vanishing. Therefore it is the solution of the original elliptic problem (19.5.1). We see that all the machinery of §19.2, on diffusive initial value equations, can be brought to bear on the solution of boundary value problems by relaxation methods.

Let us apply this idea to our model problem (19.0.3). The diffusion equation is

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} - \rho \tag{19.5.3}$$

If we use FTCS differencing (cf. equation 19.2.4), we get

$$u_{j,l}^{n+1} = u_{j,l}^n + \frac{\Delta t}{\Delta^2}\left(u_{j+1,l}^n + u_{j-1,l}^n + u_{j,l+1}^n + u_{j,l-1}^n - 4u_{j,l}^n\right) - \rho_{j,l}\Delta t \tag{19.5.4}$$

Recall from (19.2.6) that FTCS differencing is stable in one spatial dimension only if $\Delta t/\Delta^2 \leq \frac{1}{2}$. In two dimensions this becomes $\Delta t/\Delta^2 \leq \frac{1}{4}$. Suppose we try to take the largest possible timestep, and set $\Delta t = \Delta^2/4$. Then equation (19.5.4) becomes

$$u_{j,l}^{n+1} = \frac{1}{4}\left(u_{j+1,l}^n + u_{j-1,l}^n + u_{j,l+1}^n + u_{j,l-1}^n\right) - \frac{\Delta^2}{4}\rho_{j,l} \tag{19.5.5}$$

Thus the algorithm consists of using the average of u at its four nearest-neighbor points on the grid (plus the contribution from the source). This procedure is then iterated until convergence.

This method is in fact a classical method with origins dating back to the last century, called *Jacobi's method* (not to be confused with the Jacobi method for eigenvalues). The method is not practical because it converges too slowly. However, it is the basis for understanding the modern methods, which are always compared with it.

Another classical method is the *Gauss-Seidel* method, which turns out to be important in multigrid methods (§19.6). Here we make use of updated values of u on the right-hand side of (19.5.5) as soon as they become available. In other words, the averaging is done "in place" instead of being "copied" from an earlier timestep to a later one. If we are proceeding along the rows, incrementing j for fixed l, we have

$$u_{j,l}^{n+1} = \frac{1}{4}\left(u_{j+1,l}^n + u_{j-1,l}^{n+1} + u_{j,l+1}^n + u_{j,l-1}^{n+1}\right) - \frac{\Delta^2}{4}\rho_{j,l} \tag{19.5.6}$$

This method is also slowly converging and only of theoretical interest when used by itself, but some analysis of it will be instructive.

Let us look at the Jacobi and Gauss-Seidel methods in terms of the matrix splitting concept. We change notation and call **u** "**x**," to conform to standard matrix notation. To solve

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{19.5.7}$$

we can consider splitting **A** as

$$\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U} \tag{19.5.8}$$

where **D** is the diagonal part of **A**, **L** is the lower triangle of **A** with zeros on the diagonal, and **U** is the upper triangle of **A** with zeros on the diagonal.

In the Jacobi method we write for the rth step of iteration

$$\mathbf{D} \cdot \mathbf{x}^{(r)} = -(\mathbf{L} + \mathbf{U}) \cdot \mathbf{x}^{(r-1)} + \mathbf{b} \tag{19.5.9}$$

For our model problem (19.5.5), **D** is simply the identity matrix. The Jacobi method converges for matrices **A** that are "diagonally dominant" in a sense that can be made mathematically precise. For matrices arising from finite differencing, this condition is usually met.

What is the rate of convergence of the Jacobi method? A detailed analysis is beyond our scope, but here is some of the flavor: The matrix $-\mathbf{D}^{-1} \cdot (\mathbf{L} + \mathbf{U})$ is the *iteration matrix* which, apart from an additive term, maps one set of **x**'s into the next. The iteration matrix has eigenvalues, each one of which reflects the factor by which the amplitude of a particular eigenmode of undesired residual is suppressed during one iteration. Evidently those factors had better all have modulus < 1 for the relaxation to work at all! The rate of convergence of the method is set by the rate for the slowest-decaying eigenmode, i.e., the factor with largest modulus. The modulus of this largest factor, therefore lying between 0 and 1, is called the *spectral radius* of the relaxation operator, denoted ρ_s.

The number of iterations r required to reduce the overall error by a factor 10^{-p} is thus estimated by

$$r \approx \frac{p \ln 10}{(-\ln \rho_s)} \tag{19.5.10}$$

In general, the spectral radius ρ_s goes asymptotically to the value 1 as the grid size J is increased, so that more iterations are required. For any given equation, grid geometry, *and boundary condition*, the spectral radius can, in principle, be computed analytically. For example, for equation (19.5.5) on a $J \times J$ grid with Dirichlet boundary conditions on all four sides, the asymptotic formula for large J turns out to be

$$\rho_s \simeq 1 - \frac{\pi^2}{2J^2} \tag{19.5.11}$$

The number of iterations r required to reduce the error by a factor of 10^{-p} is thus

$$r \simeq \frac{2pJ^2 \ln 10}{\pi^2} \simeq \frac{1}{2} pJ^2 \tag{19.5.12}$$

In other words, the number of iterations is proportional to the number of mesh points, J^2. Since 100×100 and larger problems are common, it is clear that the Jacobi method is only of theoretical interest.

The Gauss-Seidel method, equation (19.5.6), corresponds to the matrix decomposition

$$(\mathbf{L}+\mathbf{D})\cdot\mathbf{x}^{(r)} = -\mathbf{U}\cdot\mathbf{x}^{(r-1)}+\mathbf{b} \tag{19.5.13}$$

The fact that $\mathbf{L}$ is on the left-hand side of the equation follows from the updating in place, as you can easily check if you write out (19.5.13) in components. One can show [1-3] that the spectral radius is just the square of the spectral radius of the Jacobi method. For our model problem, therefore,

$$\rho_s \simeq 1-\frac{\pi^2}{J^2} \tag{19.5.14}$$

$$r \simeq \frac{pJ^2 \ln 10}{\pi^2} \simeq \frac{1}{4}pJ^2 \tag{19.5.15}$$

The factor of two improvement in the number of iterations over the Jacobi method still leaves the method impractical.

Successive Overrelaxation (SOR)

We get a better algorithm — one that was the standard algorithm until the 1970s — if we make an *overcorrection* to the value of $\mathbf{x}^{(r)}$ at the rth stage of Gauss-Seidel iteration, thus anticipating future corrections. Solve (19.5.13) for $\mathbf{x}^{(r)}$, add and subtract $\mathbf{x}^{(r-1)}$ on the right-hand side, and hence write the Gauss-Seidel method as

$$\mathbf{x}^{(r)} = \mathbf{x}^{(r-1)} - (\mathbf{L}+\mathbf{D})^{-1}\cdot[(\mathbf{L}+\mathbf{D}+\mathbf{U})\cdot\mathbf{x}^{(r-1)}-\mathbf{b}] \tag{19.5.16}$$

The term in square brackets is just the residual vector $\xi^{(r-1)}$, so

$$\mathbf{x}^{(r)} = \mathbf{x}^{(r-1)} - (\mathbf{L}+\mathbf{D})^{-1}\cdot\xi^{(r-1)} \tag{19.5.17}$$

Now *overcorrect*, defining

$$\mathbf{x}^{(r)} = \mathbf{x}^{(r-1)} - \omega(\mathbf{L}+\mathbf{D})^{-1}\cdot\xi^{(r-1)} \tag{19.5.18}$$

Here ω is called the *overrelaxation parameter*, and the method is called *successive overrelaxation* (SOR).

The following theorems can be proved [1-3]:

- The method is convergent only for $0<\omega<2$. If $0<\omega<1$, we speak of *underrelaxation*.
- Under certain mathematical restrictions generally satisfied by matrices arising from finite differencing, only overrelaxation ($1<\omega<2$) can give faster convergence than the Gauss-Seidel method.
- If ρ_{Jacobi} is the spectral radius of the Jacobi iteration (so that the square of it is the spectral radius of the Gauss-Seidel iteration), then the *optimal* choice for ω is given by

$$\omega = \frac{2}{1+\sqrt{1-\rho_{\text{Jacobi}}^2}} \tag{19.5.19}$$

- For this optimal choice, the spectral radius for SOR is

$$\rho_{\rm SOR} = \left(\frac{\rho_{\rm Jacobi}}{1 + \sqrt{1 - \rho_{\rm Jacobi}^2}} \right)^2 \tag{19.5.20}$$

As an application of the above results, consider our model problem for which $\rho_{\rm Jacobi}$ is given by equation (19.5.11). Then equations (19.5.19) and (19.5.20) give

$$\omega \simeq \frac{2}{1 + \pi/J} \tag{19.5.21}$$

$$\rho_{\rm SOR} \simeq 1 - \frac{2\pi}{J} \qquad \text{for large} \quad J \tag{19.5.22}$$

Equation (19.5.10) gives for the number of iterations to reduce the initial error by a factor of 10^{-p},

$$r \simeq \frac{pJ \ln 10}{2\pi} \simeq \frac{1}{3} pJ \tag{19.5.23}$$

Comparing with equation (19.5.12) or (19.5.15), we see that optimal SOR requires of order J iterations, as opposed to of order J^2. Since J is typically 100 or larger, this makes a tremendous difference! Equation (19.5.23) leads to the mnemonic that 3-figure accuracy ($p = 3$) requires a number of iterations equal to the number of mesh points along a side of the grid. For 6-figure accuracy, we require about twice as many iterations.

How do we choose ω for a problem for which the answer is not known analytically? That is just the weak point of SOR! The advantages of SOR obtain only in a fairly narrow window around the correct value of ω. It is better to take ω slightly too large, rather than slightly too small, but best to get it right.

One way to choose ω is to map your problem approximately onto a known problem, replacing the coefficients in the equation by average values. Note, however, that the known problem must have the same grid size and boundary conditions as the actual problem. We give for reference purposes the value of $\rho_{\rm Jacobi}$ for our model problem on a rectangular $J \times L$ grid, allowing for the possibility that $\Delta x \neq \Delta y$:

$$\rho_{\rm Jacobi} = \frac{\cos \dfrac{\pi}{J} + \left(\dfrac{\Delta x}{\Delta y} \right)^2 \cos \dfrac{\pi}{L}}{1 + \left(\dfrac{\Delta x}{\Delta y} \right)^2} \tag{19.5.24}$$

Equation (19.5.24) holds for homogeneous Dirichlet or Neumann boundary conditions. For periodic boundary conditions, make the replacement $\pi \to 2\pi$.

A second way, which is especially useful if you plan to solve many similar elliptic equations each time with slightly different coefficients, is to determine the optimum value ω empirically on the first equation and then use that value for the remaining equations. Various automated schemes for doing this and for "seeking out" the best values of ω are described in the literature.

While the matrix notation introduced earlier is useful for theoretical analyses, for practical implementation of the SOR algorithm we need explicit formulas.

Consider a general second-order elliptic equation in x and y, finite differenced on a square as for our model equation. Corresponding to each row of the matrix **A** is an equation of the form

$$a_{j,l}u_{j+1,l} + b_{j,l}u_{j-1,l} + c_{j,l}u_{j,l+1} + d_{j,l}u_{j,l-1} + e_{j,l}u_{j,l} = f_{j,l} \tag{19.5.25}$$

For our model equation, we had $a = b = c = d = 1, e = -4$. The quantity f is proportional to the source term. The iterative procedure is defined by solving (19.5.25) for $u_{j,l}$:

$$u*_{j,l} = \frac{1}{e_{j,l}}\left(f_{j,l} - a_{j,l}u_{j+1,l} - b_{j,l}u_{j-1,l} - c_{j,l}u_{j,l+1} - d_{j,l}u_{j,l-1}\right) \tag{19.5.26}$$

Then $u^{\text{new}}_{j,l}$ is a weighted average

$$u^{\text{new}}_{j,l} = \omega u*_{j,l} + (1-\omega)u^{\text{old}}_{j,l} \tag{19.5.27}$$

We calculate it as follows: The residual at any stage is

$$\xi_{j,l} = a_{j,l}u_{j+1,l} + b_{j,l}u_{j-1,l} + c_{j,l}u_{j,l+1} + d_{j,l}u_{j,l-1} + e_{j,l}u_{j,l} - f_{j,l} \tag{19.5.28}$$

and the SOR algorithm (19.5.18) or (19.5.27) is

$$u^{\text{new}}_{j,l} = u^{\text{old}}_{j,l} - \omega\frac{\xi_{j,l}}{e_{j,l}} \tag{19.5.29}$$

This formulation is very easy to program, and the norm of the residual vector $\xi_{j,l}$ can be used as a criterion for terminating the iteration.

Another practical point concerns the order in which mesh points are processed. The obvious strategy is simply to proceed in order down the rows (or columns). Alternatively, suppose we divide the mesh into "odd" and "even" meshes, like the red and black squares of a checkerboard. Then equation (19.5.26) shows that the odd points depend only on the even mesh values and vice versa. Accordingly, we can carry out one half-sweep updating the odd points, say, and then another half-sweep updating the even points with the new odd values. For the version of SOR implemented below, we shall adopt odd-even ordering.

The last practical point is that in practice the asymptotic rate of convergence in SOR is not attained until of order J iterations. The error often grows by a factor of 20 before convergence sets in. A trivial modification to SOR resolves this problem. It is based on the observation that, while ω is the optimum *asymptotic* relaxation parameter, it is not necessarily a good initial choice. In SOR with *Chebyshev acceleration*, one uses odd-even ordering and changes ω at each half-sweep according to the following prescription:

$$\begin{aligned}
\omega^{(0)} &= 1 \\
\omega^{(1/2)} &= 1/(1-\rho^2_{\text{Jacobi}}/2) \\
\omega^{(n+1/2)} &= 1/(1-\rho^2_{\text{Jacobi}}\omega^{(n)}/4), \qquad n = 1/2, 1, ..., \infty \\
\omega^{(\infty)} &\to \omega_{\text{optimal}}
\end{aligned} \tag{19.5.30}$$

The beauty of Chebyshev acceleration is that the norm of the error always decreases with each iteration. (This is the norm of the actual error in $u_{j,l}$. The norm of the residual $\xi_{j,l}$ need not decrease monotonically.) While the asymptotic rate of convergence is the same as ordinary SOR, there is never any excuse for not using Chebyshev acceleration to reduce the total number of iterations required.

Here we give a routine for SOR with Chebyshev acceleration.

```
#include <math.h>
#define MAXITS 1000
#define EPS 1.0e-5

void sor(double **a, double **b, double **c, double **d, double **e,
	double **f, double **u, int jmax, double rjac)
```
Successive overrelaxation solution of equation (19.5.25) with Chebyshev acceleration. a, b, c, d, e, and f are input as the coefficients of the equation, each dimensioned to the grid size [1..jmax][1..jmax]. u is input as the initial guess to the solution, usually zero, and returns with the final value. rjac is input as the spectral radius of the Jacobi iteration, or an estimate of it.
```
{
	void nrerror(char error_text[]);
	int ipass,j,jsw,l,lsw,n;
	double anorm,anormf=0.0,omega=1.0,resid;
```
Double precision is a good idea for jmax bigger than about 25.
```
	for (j=2;j<jmax;j++)
```
Compute initial norm of residual and terminate iteration when norm has been reduced by a factor EPS.
```
		for (l=2;l<jmax;l++)
			anormf += fabs(f[j][l]);          Assumes initial u is zero.
	for (n=1;n<=MAXITS;n++) {
		anorm=0.0;
		jsw=1;
		for (ipass=1;ipass<=2;ipass++) {      Odd-even ordering.
			lsw=jsw;
			for (j=2;j<jmax;j++) {
				for (l=lsw+1;l<jmax;l+=2) {
					resid=a[j][l]*u[j+1][l]
						+b[j][l]*u[j-1][l]
						+c[j][l]*u[j][l+1]
						+d[j][l]*u[j][l-1]
						+e[j][l]*u[j][l]
						-f[j][l];
					anorm += fabs(resid);
					u[j][l] -= omega*resid/e[j][l];
				}
				lsw=3-lsw;
			}
			jsw=3-jsw;
			omega=(n == 1 && ipass == 1 ? 1.0/(1.0-0.5*rjac*rjac) :
				1.0/(1.0-0.25*rjac*rjac*omega));
		}
		if (anorm < EPS*anormf) return;
	}
	nrerror("MAXITS exceeded");
}
```

The main advantage of SOR is that it is very easy to program. Its main disadvantage is that it is still very inefficient on large problems.

ADI (Alternating-Direction Implicit) Method

The ADI method of §19.3 for diffusion equations can be turned into a relaxation method for elliptic equations [1-4]. In §19.3, we discussed ADI as a method for solving the time-dependent heat-flow equation

$$\frac{\partial u}{\partial t} = \nabla^2 u - \rho \tag{19.5.31}$$

By letting $t \to \infty$ one also gets an iterative method for solving the elliptic equation

$$\nabla^2 u = \rho \tag{19.5.32}$$

In either case, the operator splitting is of the form

$$\mathcal{L} = \mathcal{L}_x + \mathcal{L}_y \tag{19.5.33}$$

where $\mathcal{L}_x$ represents the differencing in x and $\mathcal{L}_y$ that in y.

For example, in our model problem (19.0.6) with $\Delta x = \Delta y = \Delta$, we have

$$\begin{aligned} \mathcal{L}_x u &= 2u_{j,l} - u_{j+1,l} - u_{j-1,l} \\ \mathcal{L}_y u &= 2u_{j,l} - u_{j,l+1} - u_{j,l-1} \end{aligned} \tag{19.5.34}$$

More complicated operators may be similarly split, but there is some art involved. A bad choice of splitting can lead to an algorithm that fails to converge. Usually one tries to base the splitting on the physical nature of the problem. We know for our model problem that an initial transient diffuses away, and we set up the x and y splitting to mimic diffusion in each dimension.

Having chosen a splitting, we difference the time-dependent equation (19.5.31) implicitly in two half-steps:

$$\begin{aligned} \frac{u^{n+1/2} - u^n}{\Delta t/2} &= -\frac{\mathcal{L}_x u^{n+1/2} + \mathcal{L}_y u^n}{\Delta^2} - \rho \\ \frac{u^{n+1} - u^{n+1/2}}{\Delta t/2} &= -\frac{\mathcal{L}_x u^{n+1/2} + \mathcal{L}_y u^{n+1}}{\Delta^2} - \rho \end{aligned} \tag{19.5.35}$$

(cf. equation 19.3.16). Here we have suppressed the spatial indices (j, l). In matrix notation, equations (19.5.35) are

$$(\mathbf{L}_x + r\mathbf{1}) \cdot \mathbf{u}^{n+1/2} = (r\mathbf{1} - \mathbf{L}_y) \cdot \mathbf{u}^n - \Delta^2 \rho \tag{19.5.36}$$

$$(\mathbf{L}_y + r\mathbf{1}) \cdot \mathbf{u}^{n+1} = (r\mathbf{1} - \mathbf{L}_x) \cdot \mathbf{u}^{n+1/2} - \Delta^2 \rho \tag{19.5.37}$$

where

$$r \equiv \frac{2\Delta^2}{\Delta t} \tag{19.5.38}$$

The matrices on the left-hand sides of equations (19.5.36) and (19.5.37) are tridiagonal (and usually positive definite), so the equations can be solved by the standard tridiagonal algorithm. Given $\mathbf{u}^n$, one solves (19.5.36) for $\mathbf{u}^{n+1/2}$, substitutes on the right-hand side of (19.5.37), and then solves for $\mathbf{u}^{n+1}$. The key question is how to choose the iteration parameter r, the analog of a choice of timestep for an initial value problem.

As usual, the goal is to minimize the spectral radius of the iteration matrix. Although it is beyond our scope to go into details here, it turns out that, for the optimal choice of r, the ADI method has the same rate of convergence as SOR. The individual iteration steps in the ADI method are much more complicated than in SOR, so the ADI method would appear to be inferior. This is in fact true if we choose the same parameter r for every iteration step. However, it is possible to choose a *different r* for each step. If this is done optimally, then ADI is generally more efficient than SOR. We refer you to the literature [1-4] for details.

Our reason for not fully implementing ADI here is that, in most applications, it has been superseded by the multigrid methods described in the next section. Our advice is to use SOR for trivial problems (e.g., 20×20), or for solving a larger problem once only, where ease of programming outweighs expense of computer time. Occasionally, the sparse matrix methods of §2.7 are useful for solving a set of difference equations directly. For production solution of large elliptic problems, however, multigrid is now almost always the method of choice.

CITED REFERENCES AND FURTHER READING:

Hockney, R.W., and Eastwood, J.W. 1981, *Computer Simulation Using Particles* (New York: McGraw-Hill), Chapter 6.

Young, D.M. 1971, *Iterative Solution of Large Linear Systems* (New York: Academic Press). [1]

Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag), §§8.3–8.6. [2]

Varga, R.S. 1962, *Matrix Iterative Analysis* (Englewood Cliffs, NJ: Prentice-Hall). [3]

Spanier, J. 1967, in *Mathematical Methods for Digital Computers, Volume 2* (New York: Wiley), Chapter 11. [4]

19.6 Multigrid Methods for Boundary Value Problems

Practical multigrid methods were first introduced in the 1970s by Brandt. These methods can solve elliptic PDEs discretized on N grid points in $O(N)$ operations. The "rapid" direct elliptic solvers discussed in §19.4 solve special kinds of elliptic equations in $O(N \log N)$ operations. The numerical coefficients in these estimates are such that multigrid methods are comparable to the rapid methods in execution speed. Unlike the rapid methods, however, the multigrid methods can solve general elliptic equations with nonconstant coefficients with hardly any loss in efficiency. Even nonlinear equations can be solved with comparable speed.

Unfortunately there is not a single multigrid algorithm that solves all elliptic problems. Rather there is a multigrid technique that provides the framework for solving these problems. You have to adjust the various components of the algorithm within this framework to solve your specific problem. We can only give a brief