Computational Electronics

Numerical Analysis

Prepared by:

Dragica Vasileska
Associate Professor

Arizona State University

Numerical Solution of Algebraic Equations

The solution of linear systems of algebraic equations is an important subject of linear algebra [1], and the computational considerations needed for computer implementation are usually treated in some detail in introductory numerical methods courses. This section simply represents a quick review or overview of the subject - it is not intended as a complete treatise on this topic. Students with little or no background in this area are referred to one of many good numerical methods texts that treat the subject in more detail. The numerical solution of large systems of algebraic equations is a direct consequence of the Finite Difference method for solving ordinary differential equations (ODEs) or partial differential equations (PDEs). Recall that the goal in these techniques is to break the continuous differential equation into a coupled set of algebraic difference equations for each finite volume or node in the system. When one has only a single independent variable (the ODE case), this process can easily lead to several hundred simultaneous equations that need to be solved. For multiple independent variables (the PDE case), systems with hundreds of thousands of equations are common. Thus, in general, we need to be able to solve large systems of linear equations of the form $Ax = b$ as part of the solution algorithm for general Finite Difference methods.

There are two general schemes for solving linear systems: Direct Elimination Methods and Iterative Methods. All the direct methods are, in some sense, based on the standard Gauss Elimination technique, which systematically applies row operations to transform the original system of equations into a form that is easier to solve. In particular, this section overviews an algorithm for implementation of the basic Gauss Elimination scheme and it also highlights the LU Decomposition method which, although functionally equivalent to the Gauss Elimination method, does provide some additional flexibility for computer implementation. Thus, the LU decomposition method is often the preferred direct solution method for low to medium sized systems (usually less than 200-300 equations).

For large systems, iterative methods (instead of direct elimination methods) are almost always used. This switch is required from accuracy considerations (related to round-off errors), from memory limitations for physical storage of the equation constants, from considerations for treating nonlinear problems, and from overall efficiency concerns. There are several specific iterative schemes that are in common use, but most methods build upon the base Gauss Seidel

method, usually with some acceleration scheme to help convergence. Thus, our focus in this brief overview is on the basic Gauss Seidel scheme and on the use of Successive Over Relaxation (SOR) to help accelerate convergence. We also give a brief introduction to the incomplete-lower-upper (ILU) decomposition method and a short tutorial to the multigrid method for solving 2D and 3D problems.

Direct Methods

1. Gauss Elimination Method

The Gauss Elimination Method forms the basis for all elimination techniques. The basic idea is to modify the original equations, using legal row operations, to give a simpler form for actual solution. The basic algorithm can be broken into two stages:

1. Forward Elimination (put equations in upper triangular form)
2. Back Substitution (solve for unknown solution vector)

To see how this works, consider the following system of equations:

$$\begin{aligned} &a_{11}x_1 + a_{12}x_2 + \ldots a_{1N}x_N = b_1 \\ &a_{21}x_1 + a_{22}x_2 + \ldots a_{2N}x_N = b_2 \\ &\vdots \\ &a_{N1}x_1 + a_{N2}x_2 + \ldots a_{NN}x_N = b_N \end{aligned} \qquad (1)$$

Now, with reference to this system of N equations and N unknowns, the Forward Elimination Step (with partial pivoting) becomes:

Step 0: Create an augmented matrix, $\tilde{A} = [Ab]$

Step 1: Determine the coefficient in the i^{th} column with the largest absolute value and interchange rows such that this element is the pivot element (i = 1, 2, 3, to N-1)

Step 2: Normalize the pivot equation (i.e. divide by the i,i element)

Step 3 Multiply normalized eqn. i by the j,i element of eqn. j

Step 4 Subtract the resultant equation in Step 3 from eqn. j

repeat Steps 3 and 4 for $j = i$+1 to N

go to Step 1 for next $i = i$+1 to N-1

and the Back Substitution Step is given by:

Step 5 $x_N = b'_N / a'_{NN}$

Step 6 $$x_i = \left(b_i' - \sum_{j=i+1}^{N} a_{ij}' x_j \right) / a_{ii}'$$

repeat for $i = N\text{-}1, N\text{-}2$, to 1

Note: The primes here indicate that the coefficients at this stage are different from the original coefficients.

2. The LU Decomposition Method

The Gauss Elimination Method has the disadvantage that all right-hand sides (i.e. all the *b* vectors of interest for a given problem) must be known in advance for the elimination step to proceed. The LU Decomposition Method outlined here has the property that the matrix modification (or decomposition) step can be performed independent of the right hand side vector. This feature is quite useful in practice - therefore, the LU Decomposition Method is usually the Direct Scheme of choice in most applications.

To develop the basic method, let us break the coefficient matrix into a product of two matrices,

$$A = LU \tag{2}$$

where L is a lower triangular matrix and U is an upper triangular matrix. Now, the original system of equations $Ax = b$, becomes

$$LUx = b \tag{3}$$

This expression can be broken into two problems,

$$Ly = b, \quad Ux = y \tag{4}$$

The rationale behind this approach is that the two systems given in Eq. (4) are both easy to solve; one by forward substitution and the other by back substitution. In particular, because L is a lower triangular matrix, the expression $Ly = b$ can be solved with a simple forward substitution step. Similarly, since U has upper triangular form, $Ux = y$ can be evaluated with a simple back substitution algorithm.

Thus the key to this method is the ability to find two matrices L and U that satisfy Eq. (4). Doing this is referred to as the Decomposition Step and there are a variety of algorithms available. Three specific approaches are as follows:

- Doolittle Decomposition:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ l_{21} & 1 & 0 & 0 \\ l_{31} & l_{32} & 1 & 0 \\ l_{41} & l_{42} & l_{43} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \tag{5}$$

Because of the specific structure of the matrices, a systematic set of formulae for the components of *L* and *U* results.

- Crout Decomposition:

$$\begin{bmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{bmatrix} \begin{bmatrix} 1 & u_{12} & u_{13} & u_{14} \\ 0 & 1 & u_{23} & u_{24} \\ 0 & 0 & 1 & u_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \tag{6}$$

The evaluation of the components of *L* and *U* is done in a similar fashion as above.

- Cholesky Factorization:

For symmetric, positive definite matrices, where $A = A^T$ and $x^T Ax > 0$ for all $x \neq 0$ then,

$$U = L^T \quad \text{and} \quad A = LL^T \tag{7}$$

and a simple set of expressions for the elements of *L* can be obtained (as above). Once the elements of *L* and *U* are available (usually stored in a single *N*×*N* matrix), Matlab's standard equation solver (using the backslash notation, x = A\b), uses several variants of the basic LU Decomposition method depending on the form of the original coefficient matrix (see the Matlab help files for details).

3. LU Decomposition in 1D

The LU decomposition method is very trivial for 1D problems where the discretization of the ODE or the PDE leads to a three point stencil and a tridiagonal matrix *A*. It is easy to show, that the system of equations that we need to solve, and for the purpose of clarity, we denote by *Ax = f*, in matrix form reads

$$\begin{bmatrix} a_1 & c_1 & 0 & 0 & \cdots & 0 \\ b_2 & a_2 & c_2 & 0 & \dots & 0 \\ 0 & b_3 & a_3 & c_3 & \dots & 0 \\ \vdots & & & & & \\ \vdots & & & & & \\ 0 & 0 & 0 & \dots & b_n & a_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ \vdots \\ f_n \end{bmatrix}. \tag{8}$$

The solution of this problem can be represented as a two-step procedure that is explained below.

Step 1: Decompose the coefficient matrix *A* into a product of lower and upper triangular matrices:

$$A = LU = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ \beta_2 & 1 & 0 & 0 & \dots & 0 \\ 0 & \beta_3 & 1 & 0 & \dots & 0 \\ \vdots & & & & & \\ \vdots & & & & & \\ 0 & 0 & 0 & \dots & \beta_n & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 & c_1 & 0 & 0 & \cdots & 0 \\ 0 & \alpha_2 & c_2 & 0 & \dots & 0 \\ 0 & 0 & \alpha_3 & c_3 & \dots & 0 \\ \vdots & & & & & \\ \vdots & & & & & \\ 0 & 0 & 0 & \dots & 0 & \alpha_n \end{bmatrix} \tag{9}$$

From the equality of the two matrices, we have:

$$\alpha_1 = a_1; \quad \beta_k = b_k / \alpha_{k-1}; \quad \alpha_k = a_k - \beta_k c_{k-1}; \quad k = 2, 3, \dots, n \tag{10}$$

Step 2: Solve the system of equations $LUx = f$, by first solving $Lg = f$ using forward substitution, and then solving $Ux = g$ using backward substitution. Then, the solution of $Lg = f$ is represented as:

$$g_1 = f_1; \quad g_k = f_k - \beta_k g_{k-1}; \quad k = 2, 3, \dots, n, \tag{11}$$

And the solution of $Ux = g$ as:

$$x_n = g_n / \alpha_n; \quad x_k = [g_k - c_k x_{k+1}] / \alpha_k; \quad k = n-1, n-2, \dots, 2, 1 \tag{12}$$

Iterative Methods

For large systems of equations, an iterative solution scheme for the unknown vector can always be written in the form

$$x^{p+1} = Bx^P + c \tag{13}$$

where *B* is the iteration matrix, *c* is a constant vector and *p* is an iteration counter. Convergence of this scheme is guaranteed if the largest eigenvalue of the iteration matrix is less that unity, where ρ = spectral radius = $|\lambda|_{\max}$. Therefore, if $\rho < 1$ the iterative scheme will converge. If

$\rho << 1$, the iterative scheme converges very rapidly. If $\rho \approx 1$ but less than unity, the scheme will be slowly converging. The iteration algorithm will diverge if the spectral radius is greater than unity. Convergence is tested during the iterative process by computing the largest relative change from one iteration to the next, and comparing the absolute value of this result with some desired tolerance. If the maximum relative change is less than the desired accuracy, then the process is terminated. If this condition is not satisfied, then another iteration is performed.

1. The Gauss Seidel Method

Let us take the original system of equations given by $Ax = b$ and convert it into the classical Gauss Seidel iterative scheme. To do this, let us break the original matrix into three specific components, or

$$A = L + D + U \tag{14}$$

where the three matrices on the right hand side, in sequence, are strictly lower triangular, diagonal, and strictly upper triangular matrices. Now, substituting this into the original expression gives

$$(L + D)x + Ux = b \tag{15}$$

or

$$(L + D)x = b - Ux \tag{16}$$

If we premultiply by $(L + D)^{-1}$ and notice that the solution vector appears on both sides of the equation, we can write the equation in an iterative form as

$$x^{p+1} = -(L + D)^{-1}Ux^{p} + (L + D)^{-1}b \tag{17}$$

Clearly this is in the standard form for iterative solutions as defined in Eq. (13), where the iteration matrix is given by

$$B = -(L + D)^{-1}U \tag{18}$$

and the constant vector is written as

$$c = (L + D)^{-1}b \tag{19}$$

This form of the iteration strategy is useful for the study of the convergence properties of model problems. It is, however, not particularly useful as a program algorithm for code implementation. For actual implementation on the computer, one writes these equations differently, never having to formally take the inverse as indicated above. In practice, Eq. (17) is written in iterative form as

$$Dx^{p+1} = b - Lx^{p+1} - Ux^{p} \tag{20}$$

or

$$x^{p+1} = D^{-1}\left(b - Lx^{p+1} - Ux^{p}\right) \tag{21}$$

This specific form is somewhat odd at first glance, since x^{p+1} appears on both sides of the equation. This is justified because of the special form of the strictly lower triangular matrix, *L*. This can be seen more clearly if the matrix equations are written using discrete notation. In discrete form Eq. (21) can be expanded as

$$x_i^{p+1} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{p+1} - \sum_{j=i+1}^{N} a_{ij}x_j^{p}\right) \tag{22}$$

where the diagonal elements of D^{-1} are simply $1/a_{ii}$ and the limits associated with the summations account for the special structure of the *L* and *U* matrices.

2. The Successive Over-Relaxation (SOR) Method

To improve the rate of convergence, one might consider using a weighted average of the results of the two most recent estimates to obtain the next best guess of the solution. If the solution is converging, this might help extrapolate to the real solution more quickly. This idea is the basis of the SOR method. In particular, let α be some weight factor with a value between 0 and 2. Now, let's compute the next value of x^{p+1} to use in the Gauss-Seidel method as a linear combination of the current value, x^{p+1}, and the previous solution, x^{p}, as follows:

$$x^{p+1}|_{new} = \alpha x^{p+1} + (1-\alpha)x^{p} \quad \text{with} \quad 0 < \alpha < 2 \tag{23}$$

Note that if α is unity, we simply get the standard Gauss Seidel method (or whatever base iterative scheme is in use). When α is greater that unity, the system is said to be over-relaxed, indicating that the latest value, x^{p+1}, is being weighted more heavily (weight for x^{p} is negative). If, however, α is less than one, the system is under-relaxed, this time indicating that the previous solution, x^{p}, is more heavily weighted (positive weight values). The idea, of course, is to choose the relaxation parameter to improve convergence (reduce the spectral radius). This is most often done in a trial and error fashion for certain classes of problems (experience helps here). Some more advanced codes do try to estimate this quantity as part of the iterative calculation, although this is not particularly easy.

3. Incomplete LU decomposition for 2D and 3D problems

Within incomplete factorization schemes [2] for 2D problems, the matrix A is decomposed into a product of lower (L) and upper (U) triangular matrices, each of which has four non-zero diagonals in the same locations as the ones of the original matrix A. The unknown elements of the L and U matrices are selected in such a way that the five diagonals common to both A and $A'= LU$ are identical and the four superfluous diagonals represent the matrix N, i.e., $A'=A+N$. Thus, rather than solving the original system of equations $Ax=b$, one solves the modified system $LUx=b+Nx$, by solving successively the matrix equations $LV=b+Nx$ and $V=Ux$, where V is an auxiliary vector. It is important to note that the four superfluous terms of N affect the rate of convergence of the ILU method. Stone [3] suggested the introduction of partial cancellation, which minimizes the influence of these additional terms and accelerates the rate of convergence of the ILU method. By using a Taylor series expansion, the superfluous terms appearing in A' are partially balanced by subtracting approximately equal terms.

4. Multigrid Method

The multi-grid method represents an improvement over the SOR and ILU methods in terms of iterative techniques available for solving large systems of equations [4]. The basic principle behind the multi-grid method is to reduce different Fourier components of the error on grids with different mesh sizes. Most iterative techniques work by quickly eliminating the high-frequency Fourier components, while the low-frequency ones are left virtually unchanged. The result is a convergence rate that is initially fast, but slows down dramatically as the high-frequency components disappear. The multi-grid method utilizes several grids, each with consecutively coarser mesh sizes. Each of these grids acts to reduce a different Fourier component of the error, therefore increasing the rate of convergence with respect to single grid based methods, such as an SOR.

Practical multigrid methods were first introduced in the 1970s by Brandt [5]. These methods can solve elliptic PDEs discretized on N grid points in $O(N)$ operations. The "rapid" direct elliptic solvers discussed in Ref. [6] 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 introduction to the subject here. In this approach, the method obtains successive solutions on finer and finer grids. You can stop the solution either at a pre-specified fineness, or you can monitor the truncation error due to the discretization, quitting only when it is tolerably small.

From One-Grid, through Two-Grid, to Multigrid

The key idea of the multigrid method can be understood by considering the simplest case of a two-grid method. Suppose we are trying to solve the linear elliptic problem

$$Lu = f \tag{24}$$

where L is some linear elliptic operator and f is the source term. When one discretizes Eq. (24) on a uniform grid with mesh size h, the resulting set of linear algebraic equations arises

$$L_h u_h = f_h \tag{25}$$

Let $\tilde{u}_h$ denote some approximate solution to Eq. (25). We will use the symbol u_h to denote the exact solution to the difference equations. Then the *error* in $\tilde{u}_h$ or the *correction* is

$$v_h = u_h - \tilde{u}_h \tag{26}$$

The *residual* or *defect* is

$$d_h = L_h \tilde{u}_h - f_h \tag{27}$$

Since L_h is linear, the error satisfies

$$L_h v_h = -d_h \tag{28}$$

At this point we need to make an approximation to L_h in order to find v_h. The classical iteration methods, such as Jacobi or Gauss-Seidel, do this by finding, at each stage, an approximate solution of the equation

$$\hat{L}_h \hat{v}_h = -d_h \tag{29}$$

where $\hat{L}_h$ is a "simpler" operator than L_h. For example, $\hat{L}_h$ is the diagonal part of L_h for Jacobi iteration, or the lower triangle for Gauss-Seidel iteration. The next approximation is generated by

$$\tilde{u}_h^{new} = \tilde{u}_h + \hat{v}_h \tag{30}$$

Now consider, as an alternative, a completely different type of approximation for L_h, one in which we "coarsify" rather than "simplify." That is, we form some appropriate approximation L_H of L_h on a coarser grid with mesh size H (we will always take $H = 2h$, but other choices are possible). The residual equation is now approximated by

$$L_H v_H = -d_H \tag{31}$$

Since L_H has smaller dimension, this equation will be easier to solve than Eq. (28). To define the defect d_H on the coarse grid, we need a *restriction operator* R that restricts d_h to the coarse grid:

$$d_H = R d_h \, . \tag{32}$$

The restriction operator is also called the *fine-to-coarse operator* or the *injection operator*. Once we have a solution $\tilde{v}_H$ to Eq. (31), we need a *prolongation operator* P that prolongates or interpolates the correction to the fine grid:

$$\tilde{v}_H P \tilde{v}_H \tag{33}$$

The prolongation operator is also called the *coarse-to-fine operator* or the *interpolation operator*. Both R and P are chosen to be linear operators. Finally the approximation $\tilde{u}_h$ can be updated:

$$\tilde{u}_h^{new} = \tilde{u}_h + \tilde{v}_h \tag{34}$$

One step of this *coarse-grid correction scheme* is thus:

- Compute the defect on the fine grid from Eq. (27).
- Restrict the defect by Eq. (32).
- Solve Eq. (31) exactly on the coarse grid for the correction.
- Interpolate the correction to the fine grid by Eq. (33).
- Compute the next approximation by Eq. (34).

Let us contrast the advantages and disadvantages of relaxation and the coarse-grid correction scheme. Consider the error v_h expanded into a discrete Fourier series. Call the components in the lower half of the frequency spectrum the *smooth components* and the high-frequency components the *nonsmooth components*. We have seen that relaxation becomes very slowly convergent in the limit $h \to 0$, i.e., when there are a large number of mesh points. The reason turns out to be that the smooth components are only slightly reduced in amplitude on each iteration. However, many relaxation methods reduce the amplitude of the nonsmooth components by large factors on each iteration: They are good *smoothing operators*. For the two-grid iteration, on the other hand, components of the error with wavelengths $2H$ are not even

representable on the coarse grid and so cannot be reduced to zero on this grid. But it is exactly these high-frequency components that can be reduced by relaxation on the fine grid! This leads us to combine the ideas of relaxation and coarse-grid correction:

- Pre-smoothing: Compute $\bar{u}_h$ by applying $\nu_1 \geq 0$ steps of a relaxation method to $\tilde{u}_h$.
- Coarse-grid correction: As above, using $\bar{u}_h$ to give $\bar{u}_h^{new}$.
- Post-smoothing: Compute $\tilde{u}_h^{new}$ by applying $\nu_2 \geq 0$ steps of the relaxation method to $\bar{u}_h^{new}$.

It is only a short step from the above two-grid method to a multigrid method. Instead of solving the coarse-grid defect Eq. (31) exactly, we can get an approximate solution of it by introducing an even coarser grid and using the two-grid iteration method. If the convergence factor of the two-grid method is small enough, we will need only a few steps of this iteration to get a good enough approximate solution. We denote the number of such iterations by γ. Obviously, we can apply this idea recursively down to some coarsest grid. There the solution is found easily, for example by direct matrix inversion or by iterating the relaxation scheme to convergence. One iteration of a multigrid method, from finest grid to coarser grids and back to finest grid again, is called a *cycle*. The exact structure of a cycle depends on the value of γ, the number of two-grid iterations at each intermediate stage. The case $\gamma = 1$ is called a V-cycle, while $\gamma = 2$ is called a W-cycle (see Fig. 1). These are the most important cases in practice. Note that once more than two grids are involved, the pre-smoothing steps after the first one on the finest grid need an initial approximation for the error v. This should be taken to be zero.

Smoothing, Restriction, and Prolongation Operators

The most popular smoothing method, and the one you should try first, is Gauss-Seidel, since it usually leads to a good convergence rate. The exact form of the Gauss-Seidel method depends on the ordering chosen for the mesh points. For typical second-order elliptic equations like our model problem, it is usually best to use red-black ordering, making one pass through the mesh updating the “even” points (like the red squares of a checkerboard) and another pass updating the “odd” points (the black squares). When quantities are more strongly coupled along one dimension than another, one should relax a whole line along that dimension simultaneously. Line relaxation for nearest-neighbor coupling involves solving a tridiagonal system, and so is still efficient. Relaxing odd and even lines on successive passes is called zebra relaxation and is

usually preferred over simple line relaxation. Note that SOR should not be used as a smoothing operator. The over-relaxation destroys the high-frequency smoothing that is so crucial for the multigrid method.

A succint notation for the prolongation and restriction operators is to give their symbol. The symbol of P is found by considering v_H to be 1 at some mesh point (x,y), zero elsewhere, and then asking for the values of Pv_H. The most popular prolongation operator is simple bilinear interpolation. It gives nonzero values at the 9 points (x, y), $(x + h, y)$, . . . , $(x - h, y - h)$, where the values are $1, \frac{1}{2}, \ldots, \frac{1}{4}$.

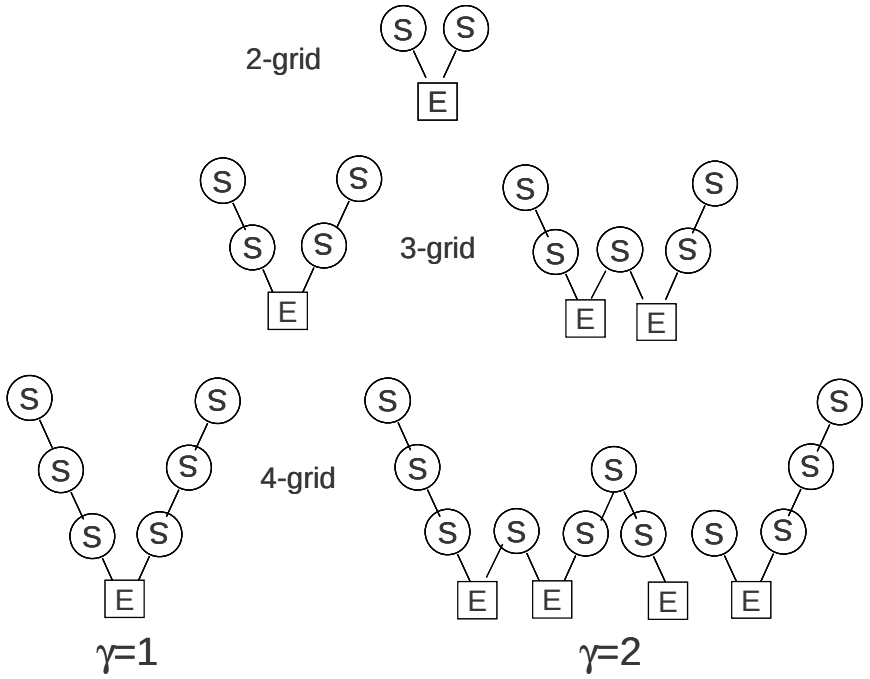

Figure 1 Structure of multigrid cycles. S denotes smoothing, while E denotes exact solution on the coarsest grid. Each descending line \ denotes restriction ® and each ascending line / denotes prolongation (P). The finest grid is at the top level of each diagram. For the V-cycles (γ=1) the E step is replaced by one 2-grid iteration each time the number of grid levels increases by one. For the W-cycles (γ=2) each E step gets replaced by two 2-grid iterations.

Its symbol is therefore

$$\begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix}. \tag{35}$$

The symbol of R is defined by considering v_h to be defined everywhere on the fine grid, and then asking what is Rv_h at (x, y) as a linear combination of these values. The simplest possible choice for R is *straight injection*, which means simply filling each coarse-grid point with the value from the corresponding fine-grid point. Its symbol is "[1]." However, difficulties can arise in practice with this choice. It turns out that a safe choice for R is to make it the adjoint operator to P. Then, take P to be bilinear interpolation, and choose $u_H = 1$ at (x, y), zero elsewhere. Then take P to be bilinear interpolation, and choose $u_H = 1$ at (x, y), zero elsewhere. Finally, the symbol of R is

$$\begin{bmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{bmatrix} \tag{36}$$

Note the simple rule: The symbol of R is 1/4 the transpose of the matrix defining the symbol of P. This rule is general whenever $R = P^{\dagger}$ and $H = 2h$. The particular choice of R in Eq. (36) is called *full weighting*. Another popular choice for R is *half weighting*, "halfway" between full weighting and straight injection. Its symbol is

$$\begin{bmatrix} 0 & \frac{1}{8} & 0 \\ \frac{1}{8} & \frac{1}{2} & \frac{1}{8} \\ 0 & \frac{1}{8} & 0 \end{bmatrix} \tag{37}$$

A similar notation can be used to describe the difference operator L_h. For example, the standard differencing of the model problem, Eq. (29), is represented by the *five-point difference star*

$$L_h = \frac{1}{h^2}\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{38}$$

If you are confronted with a new problem and you are not sure what P and R choices are likely to work well, here is a safe rule: Suppose m_p is the order of the interpolation P (i.e., it interpolates polynomials of degree m_p - 1 exactly). Suppose m_r is the order of R, and that R is the adjoint of some P (not necessarily the P you intend to use). Then if m is the order of the differential

operator L_h, you should satisfy the inequality $m_p + m_r > m$. For example, bilinear interpolation and its adjoint, full weighting, for Poisson's equation satisfy $m_p + m_r = 4 > m = 2$.

Of course the P and R operators should enforce the boundary conditions for your problem. The easiest way to do this is to rewrite the difference equation to have homogeneous boundary conditions by modifying the source term if necessary. Enforcing homogeneous boundary conditions simply requires the P operator to produce zeros at the appropriate boundary points. The corresponding R is then found by $R = P^{\dagger}$.

References

1 G. Strang, *Linear Algebra and Its Applications* (Academic Press, New York, second edition, 1980)

2 G. V. Gadiyak and M. S. Obrecht, *Simulation of Semiconductor Devices and Processes: Proceedings of the Second International Conference* (1986) 147.

3 H. L. Stone, SIAM J. Numer. Anal., 5 (1968) 536.

4 W. Hackbush, *Multi-Grid Methods and Applications* (Springer-Verlag, Berlin, 1985).

5 A. Brandt, "Multi-level adaptive technique (MLAT): The Multi-grid Method", *IBM Research Report RD-6026*, IBM T. J. Watson Research Center, Yorktown Heights, Ney York, 1976.

6 R.E. Bank and D.J. Rose, "An O(N*N) method for solving constant coefficient boundary value problems in two dimensions," SIAM Journal of Numerical Analysis, Vol. 12, pp. 529-540 (1975).