6.1 Introduction

Systems of linear equations frequently arise in atmospheric science. The systems can involve thousands or even millions of unknowns, which must be solved for simultaneously. They can be solved by a wide variety of methods, which are discussed in standard texts on numerical analysis. The solution of linear systems is conceptually simple, but may nevertheless present challenges in practice. The main issue is how to minimize the amount of computational work that must be done to obtain the solution, while at the same time minimizing the amount of storage required. For the problems that arise in atmospheric science, and considering the characteristics of modern computers, maximizing computational speed is often more of a concern than minimizing storage.

One source of linear systems is boundary-value problems. These involve spatial derivatives and/or integrals, but no time derivatives and/or integrals. They can and do frequently arise in one, two, or three dimensions, in the atmospheric sciences. Two-dimensional linear boundary-value problems occur quite often in atmospheric science. Here is an example: Consider a two-dimensional flow. Let $\zeta$ and $\delta$ be the vorticity and divergence, respectively. We can define a stream function, $\psi$, and a velocity potential, $\chi$, by

$$V_r = k \times \nabla \psi,$$

and

$$V_d = \nabla \chi,$$

respectively. Here $k$ is the unit vector perpendicular to the plane of the motion, and $V_r$ and $V_d$ are the rotational and divergent parts of the wind vector, respectively, so that

$$V = V_r + V_d.$$

The vorticity and divergence then satisfy
\[ \zeta = \nabla^2 \psi, \]  
(4)

and

\[ \delta = \nabla^2 \chi, \]  
(5)

respectively. Suppose that we are given the distributions of \( \zeta \) and \( \delta \), and want to determine the wind vector. This can be done by first solving the two boundary-value problems represented by (4)-(5), with suitable boundary conditions, then using (1)-(2) to obtain \( \mathbf{V}_r \) and \( \mathbf{V}_d \), and finally using Eq. (3) to obtain the total horizontal wind vector.

A second example is the solution of the anelastic pressure equation, in which the pressure field takes whatever shape is needed to prevent the divergence of the mass flux vector from becoming non-zero.

Further examples arise from implicit time-differencing combined with space-differencing, e.g., for the diffusion equation or the shallow-water equations.

### 6.2 Solution of one-dimensional boundary-value problems

As a simple one-dimensional example, consider

\[ \frac{d^2 q(x)}{dx^2} = f(x), \]

(6)
on a periodic domain, where \( f(x) \) is a given periodic function of \( x \). Solution of (6) requires two boundary conditions. One of these can be the condition of periodicity, which we have already specified. We assume that a second boundary condition is also given, e.g., the average of \( q \) over the domain may be prescribed.

The exact solution of (6) can be obtained by expanding \( q(x) \) and \( f(x) \) in an infinite Fourier series. The individual Fourier modes will satisfy

\[ -k^2 \hat{q}_k = \hat{f}_k, \]

(7)

which can readily be solved for the \( \hat{q}_k \), provided that the wave number \( k \) is not zero. The value of \( \hat{q}_0 \), i.e., the domain average of \( q \), must be obtained directly from the second boundary condition mentioned above. The full solution for \( q(x) \) can be obtained by Fourier-summing the \( \hat{q}_k \).
This method to find the exact solution of (6) can be adapted to obtain an approximate numerical solution, simply by truncating the expansions of \( q(x) \) and \( f(x) \) after a finite number of modes. This is called the “spectral” method. Like everything else, the spectral method has both strengths and weaknesses. It will be discussed in a later chapter.

Suppose, however, that the problem posed by (6) arises in a large numerical model, in which the functions \( q(x) \) and \( f(x) \) appear in many complicated equations, perhaps including time-dependent partial differential equations which are solved (approximately) through the use of spatial and temporal finite differences. In that case, the requirement of consistency with the other equations of the model may dictate that the spatial derivatives in (6) be approximated by a finite-difference method, such as

\[
\frac{q_{i+1} - 2q_i + q_{i-1}}{d^2} = f_i.
\]

(8)

Here \( d \) is the grid spacing. We have used centered second-order spatial differences in (8). Assuming a periodic, wave-like solution for \( q_i \), and correspondingly expanding \( f_i \), we obtain, in the usual way,

\[
-k^2 \hat{q}_k \left( \frac{\sin (kd/2)}{kd/2} \right)^2 = \hat{f}_k.
\]

(9)

Note the similarity between (9) and (7). Clearly (9) can be solved to obtain each of the \( \hat{q}_k \), except \( \hat{q}_0 \), and the result will be consistent with the finite-difference approximation (8). This example illustrates that Fourier solution methods can be used even in combination with finite-difference approximations. For each \( k \), the factor of \(-k^2 \left( \frac{\sin (kd/2)}{kd/2} \right)^2 \) in (9) can be evaluated once and stored for use later in the simulation. This is advantageous if (9) must be solved on each of many time steps. Fourier methods may or may not be applicable, depending on the geometry of the domain.

The Fourier method outlined above can produce solutions quickly, because of the existence of fast algorithms for computing Fourier transforms (not discussed here but readily available in various scientific subroutine packages). It is easy to see that the method can be extended to two or three dimensions. The Fourier method is not applicable when the problem involves spatially variable coefficients, or when the grid is nonuniform, or when the geometry of the problem is not compatible with Fourier expansion.
There are other ways to solve (8). It can be regarded as a system of linear equations, in which the unknowns are the \( q_i \). The matrix of coefficients for this particular problem turns out to be “tri-diagonal.” This means that the only non-zero elements of the matrix are the diagonal elements and those directly above and below the diagonal, as in the simple 6 x 6 problem shown below:

\[
\begin{bmatrix}
d_1 & a_2 & 0 & 0 & 0 & b_6 \\
b_1 & d_2 & a_3 & 0 & 0 & 0 \\
0 & b_2 & d_3 & a_4 & 0 & 0 \\
0 & 0 & b_3 & d_4 & a_5 & 0 \\
0 & 0 & 0 & b_4 & d_5 & a_6 \\
a_1 & 0 & 0 & 0 & b_5 & d_6
\end{bmatrix}
\begin{bmatrix}
q_1 \\
q_2 \\
q_3 \\
q_4 \\
q_5 \\
q_6
\end{bmatrix}
= 
\begin{bmatrix}
f_1 \\
f_2 \\
f_3 \\
f_4 \\
f_5 \\
f_6
\end{bmatrix}.
\]

(10)

Here each element of the 6 x 6 matrix is labeled with a single subscript, indicating its column number. The names “\(d\),” “\(a\),” and “\(b\)” denote “diagonal,” “above-diagonal,” and “below-diagonal” elements, respectively. The solution of tri-diagonal linear systems is very fast and easy. For instance, the first of the six equations represented by (10) can be solved for \( q_1 \) as a function of \( q_2 \) and \( q_6 \), provided that \( d_1 \neq 0 \). This solution can be used to eliminate \( q_1 \) in the five remaining equations. The (modified version of the) second equation can then be solved for \( q_2 \) as a function of \( q_3 \) and \( q_6 \), and this solution can be used to eliminate \( q_2 \) from the remaining four equations. Continuing in this way, we can ultimately obtain a single equation for the single unknown \( q_6 \). Once the value of \( q_6 \) has been determined, we can obtain the other unknowns by back-substitution. In case \( d_1 = 0 \) (assumed not to be true in the preceding discussion), we can immediately solve the first equation for \( q_2 \) in terms of \( q_6 \), provided that \( a_2 \) is not also equal to zero.

It should be clear that the amount of arithmetic needed to implement the tri-diagonal solver described above is simply proportional to the number of unknowns. The issue of “scaling” deals with the change in the amount of work required as the problem size increases. The algorithm described above, for use with tri-diagonal problems, scales very well. Highly optimized tri-diagonal solvers are available in standard software libraries. Because tri-diagonal systems are easy to deal with, it is good news when a problem can be expressed in terms of a tri-diagonal system. Naturally, tri-diagonal methods are not an option when the matrix is not tri-diagonal.

We could, of course, solve the linear system by other methods that are discussed in introductory texts, such as Cramer’s Rule or matrix inversion or Gaussian elimination. These “classical” methods work, but they are very inefficient compared to the Fourier and tri-diagonal methods discussed above. For each of the classical methods, the amount of arithmetic needed to
find the solution is proportional to the square of the number of unknowns. If the number of unknowns is large, the methods are prohibitively expensive.

Finally, we could solve (8) by a relaxation method. Here the idea is to make an “initial guess” for \( q_i \), then refine the guess successively, until a “sufficiently good” approximation to the exact solution of (8) is obtained. Relaxation methods were invented during the 1940s (e.g., Southwell, 1940, 1946; Allen, 1954), and are now very widely used (e.g., Strang, 2007). Several relaxation methods are discussed below.

### 6.3 Jacobi relaxation

Starting from this point, most of the discussion in this chapter is a condensed version of that found in the paper by Fulton et al. (1986).

As an example of a boundary-value problem, consider

\[
-\nabla^2 q = f \quad \text{on a two-dimensional domain, and} \\
q = g \quad \text{on the boundary of the domain.}
\]

(11)

We consider \( f \) and \( g \) to be known.

As a simple, concrete example, we approximate (11) on a grid with uniform spacing \( d \), and \( N \) grid points in each direction. Using second-order centered differences as an example, we write:

\[
d^{-2} \left( 4q_{j,k} - q_{j-1,k} - q_{j+1,k} - q_{j,k-1} - q_{j,k+1} \right) = f_{j,k} \quad \text{for } 0 < (j, k) < N , \\
q_{j,k} = g_{j,k} , \quad \text{for } j = 0, N \quad \text{and } k = 0, N
\]

(12)

We now explore relaxation methods for the solution of (12). Relaxation methods are iterative, i.e., they start with an initial guess for the solution, and obtain successively better approximations to the solution by repeatedly executing a sequence of steps. Each pass through the sequence of steps is called a “sweep.”

We need a notation that allows us to distinguish approximate solutions from exact solutions. Here by “exact” solution we mean an exact solution to the finite-difference problem posed in (12). We use a “hat” to denote the approximate solution to the finite-difference problem, i.e., we let \( \hat{q}_{j,k} \) denote an approximation to \( q_{j,k} \).

The simplest relaxation method is called Jacobi relaxation or simultaneous relaxation. The Jacobi method defines the new value \( \hat{q}^{\text{new}}_{j,k} \) by applying (12) with the new value at the point \((j,k)\) and the “old” values at the neighboring points, i.e.,
\[ d^2 \left( 4q_{j,k}^{\text{new}} - q_{j-1,k} - q_{j+1,k} - q_{j,k-1} - q_{j,k+1} \right) = f_{j,k}, \]  

or

\[ q_{j,k}^{\text{new}} = \frac{1}{4} \left( d^2 f_{j,k} + q_{j-1,k} + q_{j+1,k} + q_{j,k-1} + q_{j,k+1} \right). \]

With this approach, we compute \( q_{j,k}^{\text{new}} \) at all interior points using (13), and then replace the “old” approximate solution by the new one. This procedure is repeated until convergence is deemed adequate. Conditions for convergence are discussed briefly below.

Suppose that your first guess is that \( q_{j,k} \) is uniform across the entire grid. Then, on the first sweep, the four values of \( \hat{q} \) on the right-hand side of (14) will all be the same number, and those four terms alone will try to make \( q_{j,k}^{\text{new}} \) the same number again. It is true that the \( d^2 f_{j,k} \) term prevents this, but its effect is usually small in a single sweep, because \( h < 1 \). As a result, it can take many sweeps for the iteration to converge. The finer the grid, the smaller the value of \( d^2 f_{j,k} \), and the more sweeps it takes. We return to this point later.

Let the error of a given approximation be denoted by

\[ \varepsilon_{j,k} \equiv q_{j,k} - q_{j,k}. \]

Here again \( q_{j,k} \) is the exact solution of the finite-difference system. Consider one sweep of Jacobi relaxation. Using (15) to eliminate all values of \( \hat{q} \) (14), we find that

\[ \varepsilon_{j,k}^{\text{new}} + q_{j,k} = \frac{1}{4} \left[ d^2 f_{j,k} + (\varepsilon_{j-1,k} + \varepsilon_{j+1,k} + \varepsilon_{j,k-1} + \varepsilon_{j,k+1}) + (q_{j-1,k} + q_{j+1,k} + q_{j,k-1} + q_{j,k+1}) \right]. \]

In view of (12), this can be simplified to

\[ \varepsilon_{j,k}^{\text{new}} = \frac{1}{4} \left( \varepsilon_{j-1,k} + \varepsilon_{j+1,k} + \varepsilon_{j,k-1} + \varepsilon_{j,k+1} \right). \]

Eq. (17) shows that the new error (after the sweep) is the average of the current errors (before the sweep) at the four surrounding points.

Suppose that the error field consists of a checkerboard pattern of 1’s and -1’s. Suppose further that point \((j,k)\) has a “current” error of +1, i.e., \( \varepsilon_{j,k} = 1 \). For our assumed checkerboard pattern of 1’s and -1’s, the iterations will converge to the exact solution in only one sweep. However, if the error field is a regular grid of 1’s and -1’s, it will take many sweeps for convergence.
error pattern, it follows that the errors at the neighboring points referenced on the right-hand side of (17) are all equal to -1. At the end of the sweep we will have $\varepsilon_{j,k} = -1$. Then, on the next iteration, we will again obtain $\varepsilon_{j,k} = 1$. You should be able to see that the checkerboard error pattern “flips sign” from one iteration to the next. The checkerboard error can never be reduced to zero by Jacobi iteration.

Here is a better way to analyze the problem. First, rewrite (17) as

$$
\varepsilon_{j,k}^{\text{new}} = \varepsilon_{j,k} + \frac{1}{4} \left( \varepsilon_{j-1,k} + \varepsilon_{j+1,k} + \varepsilon_{j,k-1} + \varepsilon_{j,k+1} - 4\varepsilon_{j,k} \right).
$$

(18)

The quantity in parentheses in (18) is an “increment” which, when added to the “old” error, $\varepsilon_{j,k}$, gives the new error, $\varepsilon_{j,k}^{\text{new}}$. Eq. (18) looks like time differencing, and we can use von Neumann’s method to analyze the decrease in the error from one sweep to the next. First, write

$$
\varepsilon_{j,k} = \hat{\varepsilon} e^{i(jld+km\delta)},
$$

(19)

where $l$ and $m$ are the wave numbers in the $x$ and $y$ directions, respectively. We also define an “amplification factor” by

$$
\varepsilon_{j,k}^{\text{new}} = \lambda \varepsilon_{j,k}.
$$

(20)

Substituting (19) and (20) into (18), we find that

$$
\lambda = 1 + \frac{1}{4} \left( e^{i(j-1)ld} + e^{i(j+1)ld} + e^{i(k-1)m\delta} + e^{i(k+1)m\delta} - 4 \right)
$$

$$
= \frac{1}{2} \left[ \cos(ld) + \cos(md) \right].
$$

(21)

If $\lambda$ is negative, the sign of the error will oscillate from one sweep to the next. To have rapid, monotonic convergence, we want $\lambda$ to be positive and considerably less than one. Eq. (21) shows that for “long” modes, with $ld \ll 1$ and $md \ll 1$, $\lambda$ is just slightly less than one. This means that the long modes are slowly damped. For the checkerboard, of wavelength $ld = md = \pi$, we get $\lambda = -1$. This corresponds to the oscillation already discussed above. The error goes to zero after a single sweep for $ld = md = \pi / 2$, corresponding to a wavelength (in both directions) of $4d$. In short, the $2d$ error never goes away, but the $4d$ error is eliminated after a single sweep.
A strategy to overcome the checkerboard problem is to “under-relax.” To understand this approach, we first re-write (14) as

\[ \hat{q}_{j,k}^{\text{new}} = \hat{q}_{j,k} + \left[ \frac{1}{4} \left( d^2 f_{j,k} + \hat{q}_{j-1,k} + \hat{q}_{j+1,k} + \hat{q}_{j,k-1} + \hat{q}_{j,k+1} \right) - \hat{q}_{j,k} \right]. \]  

(22)

This simply says that \( \hat{q}_{j,k}^{\text{new}} \) is equal to \( \hat{q}_{j,k} \) plus an “increment.” For the checkerboard error, the increment given by Jacobi relaxation tries to reduce the error, but it is “too large,” and so “overshoots;” this is why the sign of \( \hat{q}_{j,k}^{\text{new}} \) flips from one iteration to the next. We can reduce the increment by multiplying it by a factor less than one, which we will call \( \omega \), i.e., we replace (18) by

\[ \hat{q}_{j,k}^{\text{new}} = \hat{q}_{j,k} + \omega \left[ \frac{1}{4} \left( d^2 f_{j,k} + \hat{q}_{j-1,k} + \hat{q}_{j+1,k} + \hat{q}_{j,k-1} + \hat{q}_{j,k+1} \right) - \hat{q}_{j,k} \right]. \]  

(23)

where \(-1 < \omega < 1\). For \( \omega = 0 \), a sweep does nothing. For \( \omega = 1 \), (23) reverts to (22). With the use of (23), (18) is replaced by

\[ \hat{q}_{j,k}^{\text{new}} = \hat{q}_{j,k} (1 - \omega) + \frac{\omega}{4} \left( d^2 f_{j,k} + \hat{q}_{j-1,k} + \hat{q}_{j+1,k} + \hat{q}_{j,k-1} + \hat{q}_{j,k+1} \right). \]  

(24)

Substitution of (15) into (24), with the use of (12), gives

\[ \varepsilon_{j,k}^{\text{new}} = \varepsilon_{j,k} (1 - \omega) + \frac{\omega}{4} \left( \varepsilon_{j-1,k} + \varepsilon_{j+1,k} + \varepsilon_{j,k-1} + \varepsilon_{j,k+1} \right). \]  

(25)

From (25), you should be able to see that, if we choose \( \omega = 0.5 \), the checkerboard error will be destroyed in a single pass. This demonstrates that under-relaxation can be useful with the Jacobi algorithm. On the other hand, using \( \omega = 0.5 \) makes the long modes converge even more slowly than with \( \omega = 1 \). This suggests that the optimal value of \( \omega \) satisfies \( \frac{1}{2} < \omega < 1 \).

Suppose that, on a particular sweep, the error is spatially uniform over the grid. Then, according to (17), the error will never change under Jacobi relaxation, and this is true even with under-relaxation, as can be seen from (25). This is not really a problem, however, because, as discussed earlier, when solving a problem of this type the average over the grid has to be determined by a boundary condition. For example, if the appropriate boundary condition can be applied at the time of formulating the first guess, then the domain-mean error will be zero even before the relaxation begins.
Note, however, that if the error field is spatially smooth (but not uniform), it will change only a little on each sweep. This is a clue that the “large-scale” part of the error is reduced only slowly, while the smaller-scale part of the error is reduced more rapidly. Once the small-scale part of the error has been removed, the remaining error is smooth.

The slow convergence of the long modes determines how many iterations are needed to reduce the overall error to an acceptable level. The reason that the long modes converge slowly is that, as you can see from the algorithm, each sweep shares information only among grid cells that are immediate neighbors. Information travels across $N$ grid cells only after $N$ sweeps. As a result, many sweeps are needed for information to travel across a large grid.

For a given domain size, convergence is slower (i.e., more sweeps are needed) when the grid spacing is finer. This seems fair, since a finer grid can hold more information.

Given that the long modes are the ones that are most accurately resolved on the grid, it is ironic that they “cause trouble” by limiting the speed of convergence.

For errors of intermediate spatial scale, Jacobi relaxation works reasonably well.

6.4 Gauss-Seidel relaxation

Gauss-Seidel relaxation is similar to Jacobi relaxation, except that each value is updated immediately after it is calculated. For example, suppose that we start at the lower left-hand corner of the grid, and work our way across the bottom row, then move to the left-most end of the second row from the bottom, and so on. In Gauss-Seidel relaxation, as we come to each grid point we use the “new” values of all $q$ s that have already been updated, so that (14) is replaced by

$$
\hat{q}_{j,k}^{\text{new}} = \frac{1}{4} \left( d^2 f_{j,k} + \hat{q}_{j-1,k}^{\text{new}} + \hat{q}_{j+1,k}^{\text{new}} + \hat{q}_{j,k-1}^{\text{new}} + \hat{q}_{j,k+1}^{\text{new}} \right).
$$

(26)

This immediately reduces the storage requirements, because it is not necessary to save all of the old values and all of the new values simultaneously. More importantly, it also speeds up the convergence of the iteration, relative to Jacobi relaxation.

Obviously (26) does not apply to the very first point encountered on the very first sweep, because at that stage no “new” values are available. For the first point, we will just perform a Jacobi-style update using (14). It is only for the second and later rows of points that (26) actually applies. Because values are updated as they are encountered during the sweep, the results obtained with Gauss-Seidel relaxation depend on where the sweep starts. To the extent that the final result satisfies (12) exactly, it will be independent of where the sweep starts.

For Gauss-Seidel relaxation, the error-reduction formula corresponding to (17) is
\[ \epsilon_{j,k}^{\text{new}} = \frac{1}{4} \left( \epsilon_{j-1,k}^{\text{new}} + \epsilon_{j+1,k}^{\text{new}} + \epsilon_{j,k-1}^{\text{new}} + \epsilon_{j,k+1}^{\text{new}} \right). \]  

(27)

You should be able to see that with Gauss-Seidel iteration a checkerboard error is in fact reduced on each sweep. Consider the following simple example on a 6x6 mesh. Suppose that \( f \) is identically zero, so that the solution (with periodic boundary conditions) is that \( q \) is spatially constant. We make the rather ill-considered first guess that the solution is a checkerboard:

\[ q_{j,k}^{0} = \begin{bmatrix}
1 & -1 & 1 & -1 & 1 & -1 \\
-1 & 1 & -1 & 1 & -1 & 1 \\
1 & -1 & 1 & -1 & 1 & -1 \\
-1 & 1 & -1 & 1 & -1 & 1 \\
1 & -1 & 1 & -1 & 1 & -1 \\
-1 & 1 & -1 & 1 & -1 & 1 \\
\end{bmatrix}. \]

(28)

Here the superscript zero denotes the first guess. After partially completing one sweep, doing the bottom row and the left-most three elements of the second row from the bottom, we have:

\[ q_{j,k}^{1,\text{partial}} = \begin{bmatrix}
1 & -1 & 1 & -1 & 1 & -1 \\
-1 & 1 & -1 & 1 & -1 & 1 \\
1 & -1 & 1 & -1 & 1 & -1 \\
-1 & 1 & -1 & 1 & -1 & 1 \\
-0.5 & 0.25 & -0.281 & -1 & 1 & -1 \\
1 & -0.5 & 0.625 & -0.593 & 0.602 & -0.60 \\
\end{bmatrix}. \]

(29)

Although the solution is flipping sign as a result of the sweep, the amplitude of the checkerboard is decreasing significantly. You can finish the exercise for yourself.

Inspection of (29) shows that the errors have been reduced after one (partial) sweep, but the sign of the errors has not changed. The error has moved towards zero, but not far enough. This suggests that convergence can be speeded up by multiplying the increment by a factor greater than one, i.e., by “over-relaxation.” By analogy with (24), we replace (26) by

\[ \hat{q}_{j,k}^{\text{new}} = \hat{q}_{j,k} (1 - \omega) + \frac{\omega}{4} \left( d^2 f_{j,k} + \hat{q}_{j-1,k}^{\text{new}} + \hat{q}_{j+1,k}^{\text{new}} + \hat{q}_{j,k-1}^{\text{new}} + \hat{q}_{j,k+1}^{\text{new}} \right), \]  

(30)

where this time we choose \( \omega > 1 \). It can be shown that the convergence of (30) is optimized (i.e., made as rapid as possible) if we choose...
\[ \omega = \frac{2}{1 + \sin \left( \frac{\pi d}{L} \right)}, \]  
\text{where } L \text{ is the total width of the domain. The algorithm represented by (30) and (31) is called “successive over-relaxation,” or SOR. Choosing } \omega \text{ too large will cause the iteration to diverge. In practice, some experimentation may be needed to find the best value of } \omega. \]

### 6.5 The alternating-direction implicit method

Yet another relaxation scheme is the “alternating-direction implicit” method, often called “ADI” for short. With ADI, the spatial coordinates are treated separately and successively within each iteration sweep. We rewrite (12) as

\[ (-q_{j-1,k} + 2q_{j,k} - q_{j+1,k}) + (-q_{j,k-1} + 2q_{j,k} - q_{j,k+1}) = d^2 f_{j,k}. \]

The first quantity in parentheses on the left-hand side of (32) involves variations in the x-direction only, and the second involves variations in the y-direction only. We proceed in two steps on each sweep. The first step treats the x-dependence to produce an intermediate approximation by solving

\[ \left[ -\hat{q}_{j-1,k}^{\text{int}} + (2 + r)\hat{q}_{j,k}^{\text{int}} - \hat{q}_{j+1,k}^{\text{int}} \right] + \left[ -\hat{q}_{j,k-1}^{\text{int}} + (2 - r)\hat{q}_{j,k}^{\text{int}} - \hat{q}_{j,k+1}^{\text{int}} \right] = d^2 f_{j,k} \]

for the values with superscript “int.” Here \( r \) is a parameter used to control convergence, as discussed below. Eq. (33) is a tri-diagonal system, which can easily be solved. The sweep is completed by solving

\[ \left[ -\hat{q}_{j-1,k}^{\text{new}} + (2 - r)\hat{q}_{j,k}^{\text{new}} - \hat{q}_{j+1,k}^{\text{new}} \right] + \left[ -\hat{q}_{j,k-1}^{\text{int}} + (2 + r)\hat{q}_{j,k}^{\text{int}} - \hat{q}_{j,k+1}^{\text{int}} \right] = d^2 f_{j,k} \]

as a second tridiagonal system. It can be shown that the ADI method converges if \( r \) is positive and constant for all sweeps. The optimal value of \( r \) is

\[ r = 2 \sin \left( \frac{\pi d}{L} \right). \]

### 6.6 Multigrid methods

Fulton et al. (1986) summarize the multi-grid approach to solving boundary-value problems, which was developed by Achi Brandt (1973, 1977; see additional references in...
Fulton’s paper). The basic idea is very simple and elegant, although implementation can be complicated.

As we have already discussed, with Gauss-Seidel relaxation the small-scale errors are eliminated quickly, while the large-scale errors disappear more slowly. As the iteration proceeds, the error becomes smoother and undergoes an overall decrease in amplitude. A key observation is that, essentially by definition of “large-scale,” the large-scale part of the error can be represented on a relatively coarse grid. On such a coarse grid, the large-scale errors are represented using fewer grid points, and so can be removed quickly.

Putting these ideas together, we arrive at a strategy whereby we use a coarse grid to relax away the large-scale errors, and a fine grid to relax away the small-scale errors. In practice, we introduce as many “nested” grids as possible, each coarse grid is composed of a subset of the points used in the finer grids. The “multi” in the multi-grid method is quite important. We move back and forth between the grids, from coarse to fine by interpolation, and from fine to coarse by “injection” (copying) of the fine grid values onto the corresponding points of the coarse grid. A relaxation (e.g., Gauss-Seidel) is done on each grid in turn. The sweeps on the coarser grids remove the large-scale part of the error, while the sweeps on the finer grids remove the small-scale part of the error.

Although the transfers between grids involve some computational work, the net effect is to speed up the solution (for a given degree of error) considerably beyond what can be achieved through relaxation on a single grid.

Here is a brief summary of how a multigrid method can be implemented in practice. Suppose that our unknown, \( q \), satisfies

\[
-L^M q^M = f^M ,
\]

(36)

where \( L \) is a linear operator (which could be the Laplacian). Eq. (36) is a generalization of (11). The superscripts \( M \) in (36) denote the fine grid on which we want to obtain the solution of (36). We need this grid-naming notation because the multigrid method involves additional grids. Prior to convergence, our approximate solution on grid \( M \) is given by \( \hat{q}^M \), and the error is denoted by \( \epsilon^M \). Substituting (37) into (36), we find that

\[
-L^M (\hat{q}^M - \epsilon^M) = f^M .
\]

(38)

Since \( L \) is linear (by assumption), we know that
With the use of (39), we can rewrite (38) as

$$L^M (\hat{q}^M - \epsilon^M) = L^M \hat{q}^M - L^M \epsilon^M. \quad (39)$$

$$L^M \epsilon^M = r^M, \quad (40)$$

where

$$r^M \equiv f^M + L^M \hat{q}^M \quad (41)$$

is called the “residual,” and Eq. (40) is called the “residual equation.” The residual is the what comes out when the operator $L^M$ is applied to the error. Eq. (40) shows that when the error is zero everywhere, the residual is also zero.

The concept of the residual is useful because, as can be seen from (41), the quantities needed to compute $r^M$ are known. They are the forcing function, $f^M$, and the approximate (partially converged) solution, $\hat{q}^M$. In contrast, the error itself is not known. If the error was known, we could just use (37) to compute the exact solution, i.e.,

$$q^M = \hat{q}^M - \epsilon^M. \quad (42)$$

and we would be done! Since the residual is known, the unknown in (40) is the error, $\epsilon^M$. Instead of solving (36) for $q^M$, we can solve the residual equation (40) for $\epsilon^M$.

There is a good reason to solve for $\epsilon^M$ instead of $q^M$. Recall that, during the iteration, the high-wave-number part of the error is quickly eliminated, so that part-way through the solution procedure $\epsilon^M$ is smooth, even if the final solution for $q^M$ is not smooth. Because $\epsilon^M$ is smooth, we can represent it on a coarser grid, which will be denoted by superscript $l$. This is the motivation for solving for the smooth $\epsilon^M$ rather than for (the possibly noisy) $q^M$.

Usually the coarser grid has half as many points in each direction as the next finer grid, so, with a two-dimensional domain, grid $l$ would have $\frac{1}{4}$ as many points as grid $M$. With this in mind, we replace (40) by

$$L^l \epsilon^l = I^l_M r^M, \quad (43)$$

where $I^l_M$ is an operator that transfers the residual, $r^M$, from grid $M$ to grid $l$. This process is called “restriction.” The points that comprise grid $l$ may simply be a subset of the points on grid
In which case only copies are needed; no interpolation is necessary. This is called “injection.”

You can probably see where this is going. After doing a sweep on grid \( l \), the error has been further smoothed, and a new residual equation can be solved on an even coarser grid. This process can be repeated until reaching the coarsest possible grid -- say a 2 x 2 grid (Fig. 6.1). On the coarsest grids, direct solves (e.g., matrix inversion) may be preferable to iterative methods.

Having done worked our way up to the coarsest possible grid, we start back the other way, towards the finer grids. The error on the coarsest grid is interpolated to construct the error on the next finer grid. This is called “prolongation.” A sweep is performed on the finer grid, and the result is interpolated to the next finer grid, and so on, until we arrive back at the finest grid.

The sequence of sweeps on successively coarser grids, followed by interpolation and sweeps on the successively finer grids, is called a \( V \)-cycle.

For further discussion of multi-grid methods, see the paper by Fulton et al. (1986).

### 6.7 Summary

Boundary-value problems occur quite frequently in atmospheric science. The main issue is not finding the solution, but rather finding it quickly. Fast solutions to one-dimensional problems are very easy to obtain, but two- and three-dimensional problems are more challenging, particularly when complex geometry is involved. Among the most useful methods available today for multi-dimensional problems are the multi-grid methods and the conjugate-gradient methods (e.g., Shewchuk, 1994).
Table 1 summarizes the operations counts and storage requirements of some well known methods for solving boundary-value problems. The best possible scalings for the operation count and storage requirement are $O(N^2)$. Only the multi-grid method achieves this ideal.

<table>
<thead>
<tr>
<th>Method</th>
<th>Operation Count</th>
<th>Storage Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gaussian Elimination</td>
<td>$N^4$</td>
<td>$N^3$</td>
</tr>
<tr>
<td>Jacobi</td>
<td>$N^4$</td>
<td>$N^2$</td>
</tr>
<tr>
<td>Gauss-Seidel</td>
<td>$N^4$</td>
<td>$N^2$</td>
</tr>
<tr>
<td>Successive Over-Relaxation</td>
<td>$N^3$</td>
<td>$N^2$</td>
</tr>
<tr>
<td>Alternating Direction Implicit</td>
<td>$N^3 \ln N$</td>
<td>$N^2$</td>
</tr>
<tr>
<td>Multigrid</td>
<td>$N^2$</td>
<td>$N^2$</td>
</tr>
</tbody>
</table>

Table 1. Well known methods for solving boundary value problems, and the operation count and storage. Here the total number of points is $N^2$. 
Problems

1. Consider a square domain, of width $L$, with periodic boundary conditions in both $x$ and $y$ directions. We wish to solve

$$\nabla^2 q = f(x,y) = \left( \sin \frac{4\pi x}{L} \right) \left( \cos \frac{4\pi y}{L} \right)$$

for the unknown function $q$, where

$$0 \leq x \leq L,$n

$$0 \leq y \leq L .$$

Assume that the domain-average value of $q$ is zero, and impose this condition on your numerical solution. For simplicity, use $L = 1$. Use centered second-order differences to approximate $\nabla^2 q$. Use $N = 100$ points in both directions. The periodic boundary conditions mean that $j = 1$ is the same as $j = 101$, and $k = 1$ is the same as $k = 101$.

(a) Find and plot the exact solution.

(b) Also find and plot the solution using each of the relaxation methods listed below.

- Jacobi relaxation;
- Jacobi under-relaxation, with a suitable choice of the parameter $\omega$;
- Gauss-Seidel relaxation;
- Gauss-Seidel over-relaxation, with a suitable choice of the parameter $\omega$.

For each of the relaxation methods, try the following two initial guesses:

1) $q_{j,k} = (-1)^{jk}$,

2) $q_{j,k} = 0$ everywhere.

(c) Let $n$ be an “iteration counter,” i.e., $n = 0$ for the initial guess, $n = 1$ after one sweep, etc. Define the error after $n$ sweeps by

$$\varepsilon_{j,k}^n \equiv \left( q_{j,k}^n \right) - f_{j,k} .$$

(46)
Here $\nabla^2 \left( \hat{q}_{j,k}^n \right)$ is the finite-difference Laplacian of the approximate solution, and $f_{j,k}$ is “forcing function” given in (20), as evaluated on the grid. Let the convergence criterion be

$$\max \forall (j,k) \left\{ \left| e_{j,k}^n \right| \right\} < 10^{-2} \max \forall (j,k) \left\{ \left| f_{j,k} \right| \right\}. \quad (48)$$

How many iterations are needed to obtain convergence with Jacobi, Gauss-Seidel, and SOR?

(d) Plot the RMS error $R^n \equiv \sqrt{\frac{1}{N^2} \sum_{j=1}^{N} \sum_{k=1}^{N} \left( e_{j,k}^n \right)^2}$ as a function of $n$ (or, if you prefer, as a function of $\ln n$) for all three methods.

2. Use von Neumann’s method to analyze the convergence of

$$\hat{q}_{j,k}^{\text{new}} = \hat{q}_{j,k} + \omega \left[ \frac{1}{4} \left( d^2 f_{j,k} + \hat{q}_{j-1,k} + \hat{q}_{j+1,k} + \hat{q}_{j,k,1} + \hat{q}_{j,k,-1} \right) - \hat{q}_{j,k} \right].$$

3. Construct a subroutine that solves for the stream function, given the vorticity, on the hexagonal grid, starting from the code that you created for an earlier homework problem. Use a relaxation method of your choice, and state what the method is. Test the subroutine by feeding it functions for which you can compute the solution analytically.