CHAPTER 5 Boundary-value problems

5.1 Introduction

Boundary-value problems 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. They can be solved by a wide variety of methods, which are discussed in standard texts on numerical analysis.

The solution of linear boundary-value problems is conceptually simple, but may nevertheless present challenges in practice. *The main issue in the numerical solution of boundary-value problems is how to minimize the amount of computational work that must be done to obtain the solution*, while at the same time minimizing amount of storage required. For the problems that arise in atmospheric science, and considering the characteristics of modern computers, maximizing computational speed is usually more of a concern than minimizing storage.

Two-dimensional linear boundary-value problems occur quite often in atmospheric science. A particularly ubiquitous example is the following. Consider a two-dimensional flow. Let ζ and δ be the vorticity and divergence, respectively. We can define a stream function, ψ, and a velocity potential, χ, by

$$\mathbf{V}_r = \mathbf{k} \times \nabla\psi \,, \tag{5.1}$$

and

$$\mathbf{V}_d = \nabla\chi \,, \tag{5.2}$$

respectively. Here $\mathbf{k}$ is the unit vector perpendicular to the plane of the motion, and $\mathbf{V}_r$ and $\mathbf{V}_d$ are the rotational and divergent parts of the wind vector, respectively, so that

$$\mathbf{V} = \mathbf{V}_r + \mathbf{V}_d \,. \tag{5.3}$$

The vorticity and divergence then satisfy

$$\zeta = \nabla^2\psi \tag{5.4}$$

and

$$\delta = \nabla^2 \chi , \tag{5.5}$$

respectively.

Suppose that we are given the distributions of ζ and δ, and we need to determine the wind vector. This can be done by first solving the two boundary-value problems represented by (5.4)-(5.5), with suitable boundary conditions, then using (5.1)-(5.2) to obtain $\mathbf{V}_r$ and $\mathbf{V}_d$, and finally using (5.3) to obtain the total horizontal wind vector.

A second example is the solution of the anelastic pressure equation.

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

5.2 Solution of one-dimensional boundary-value problems

As a simple one-dimensional example, consider

$$\frac{d^2}{dx^2} q(x) = f(x) , \tag{5.6}$$

on a periodic domain, where $f(x)$ is a given periodic function of x. Solution of (5.6) 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 (5.6) can be obtained by expanding $q(x)$ and $f(x)$ in infinite Fourier series. The individual Fourier modes will satisfy

$$-k^2 \hat{q}_k = \hat{f}_k , \tag{5.7}$$

which can readily be solved for the q_k, provided that k is not zero. The value of q_0 must be obtained directly from the second boundary condition mentioned above. The full solution for $q(x)$ can be obtained by Fourier summing the q_k.

This method to find the exact solution of (5.6) 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, it has both strengths and weaknesses. It will be discussed in a later chapter.

Suppose, however, that the problem posed by (5.6) 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 (5.6) be approximated by a finite-difference method, such as

$$\frac{q_{i+1} - 2q_i + q_{i-1}}{d^2} = f_i \,. \tag{5.8}$$

Here d is the grid spacing in the x-direction. We have used centered second-order spatial differences in (5.8). Assuming a periodic, wave-like solution for q_i, and correspondingly expanding f_i, we obtain, in the usual way,

$$-k^2 \hat{q}_k \left(\frac{\sin\frac{kd}{2}}{\frac{kd}{2}} \right)^2 = \hat{f}_k \,. \tag{5.9}$$

Note the similarity between (5.9) and (5.7). Clearly (5.9) can be solved to obtain each of the q_k, except q_0, and *the result will be consistent with the finite-difference approximation (5.8)*. This example illustrates that Fourier solution methods can be used even in combination with finite-difference approximations. The factor of $-k^2 \left(\frac{\sin\frac{kd}{2}}{\frac{kd}{2}} \right)^2$ in (5.9) need only be evaluated once and then stored, for each k, even if (5.9) must be solved on each of many time steps.

The 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, provided only that the geometry of the problem is compatible with Fourier expansion.

There are other ways to solve (5.8). It can be regarded as a system of linear equations, in which the unknowns are the q_i. The matrix of coefficients is then “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:

$$\begin{bmatrix} d_1 & a_2 & 0 & 0 & 0 & b_6 \\ b_1 & d_2 & a_3 & 0 & 0 & 0 \\ 0 & b_2 & d_3 & a_4 & 0 & 0 \\ 0 & 0 & b_3 & d_4 & a_5 & 0 \\ 0 & 0 & 0 & b_4 & d_5 & a_6 \\ a_1 & 0 & 0 & 0 & b_5 & d_6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \end{bmatrix} . \tag{5.10}$$

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 (5.10) can be solved for x_1 as a function of x_2, and x_6, provided that $d_1 \neq 0$. This solution can be used to eliminate x_1 in the five remaining equations. The (modified version of the) second equation can then be solved for x_2 as a function of x_3 and x_6, and this solution can be used to eliminate x_2 from the remaining four equations. Continuing in this way, we can ultimately obtain a single equation for the single unknown x_6. Once the value of x_6 has been determined, we can obtain the other unknowns by back-substitution. It should be clear that the amount of arithmetic needed to implement this algorithm is simply proportional to the number of unknowns. This is good.

In case $d_1 = 0$ (assumed *not* to be true in the preceding discussion) we can immediately solve the first equation for x_2 in terms of x_6, provided that a_2 is not also equal to zero.

Highly optimized versions of this simple tri-diagonal solver can be found in standard software libraries. Because tri-diagonal systems are easy to deal with, we are always happy when we can express a problem that we are working on as 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 methods work but they are very inefficient compared to the Fourier and tri-diagonal methods discussed above. The amount of arithmetic involved 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 (5.8) 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 (5.8) is obtained. Several relaxation methods are discussed later in this chapter.

5.3 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

$$\begin{aligned} -\nabla^2 u &= f \text{ in } \Omega, \\ u &= g \text{ on } \partial\Omega. \end{aligned} \tag{5.11}$$

Here Ω is a two-dimensional domain, and $\partial\Omega$ is the boundary of Ω. We consider f and g to be known.

We approximate (5.11) on a grid with uniform spacing $h = 1/N$ in both the x and y directions, with $N+1$ grid points in each direction. Note that, with this definition, h is non-dimensional; in terms of dimensional quantities, h is the distance between grid points divided by the total width of the domain. Normally $h \ll 1$. Using second-order centered differences (for example), we write:

$$\begin{gathered} h^{-2}(4u_{j,k} - u_{j-1,k} - u_{j+1,k} - u_{j,k-1} - u_{j,k+1}) = f_{j,k},\ 0 < (j,k) < N; \\ u_{j,k} = g_{j,k},\ j = 0, j = N, k = 0, \text{ or } k = N. \end{gathered} \tag{5.12}$$

We now explore relaxation methods for the solution of (5.12). 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 to distinguish approximate solutions from exact solutions. Here by "exact" solution we mean an exact solution to the finite-difference problem posed in (5.12). We use a "hat" to denote the approximate solution, i.e. we let $\hat{u}_{j,k}$ denote an approximation to $u_{j,k}$.

The simplest relaxation method is called Jacobi relaxation or simultaneous relaxation. The Jacobi method defines the new value $\hat{u}_{j,k}^{new}$ by applying (5.12) with the new value at the point (j,k) and the "old" values at the neighboring points, i.e.

$$h^{-2}(4\hat{u}_{j,k}^{new} - \hat{u}_{j-1,k} - \hat{u}_{j+1,k} - \hat{u}_{j,k-1} - \hat{u}_{j,k+1}) = f_{j,k}\ , \tag{5.13}$$

or

$$\hat{u}_{j,k}^{new} = \frac{1}{4}(h^2 f_{j,k} + \hat{u}_{j-1,k} + \hat{u}_{j+1,k} + \hat{u}_{j,k-1} + \hat{u}_{j,k+1})\ . \tag{5.14}$$

With this approach, we compute $\hat{u}_{j,k}^{new}$ at all interior points using (5.13), and then bodily replace the "old" approximate solution by the new one. This procedure is repeated until convergence is deemed adequate. Conditions for convergence are not discussed here.

Let the error of a given approximation be denoted by

$$v_{j,k} \equiv \hat{u}_{j,k} - u_{j,k} . \tag{5.15}$$

Here again $u_{j,k}$ is the exact solution of the finite-difference system. Consider one sweep of Jacobi relaxation. Substituting (5.15) into (5.14), we find that

$$\begin{aligned} v_{j,k}^{new} + u_{j,k} = \frac{1}{4}\Big[h^2 f_{j,k} &+ (\nu_{j-1,k} + \nu_{j+1,k} + \nu_{j,k-1} + \nu_{j,k+1}) \\ &+ (u_{j-1,k} + u_{j+1,k} + u_{j,k-1} + u_{j,k+1})\Big] \end{aligned} \tag{5.16}$$

Using (5.12), this can be simplified to

$$v_{j,k}^{new} = \frac{1}{4}(v_{j-1,k} + v_{j+1,k} + v_{j,k-1} + v_{j,k+1}) . \tag{5.17}$$

This 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., $v_{j,k} = 1$. For our assumed checkerboard error pattern, it follows that the errors at the neighboring points referenced on the right-hand side of (5.17) are all equal to -1. We conclude that $v_{j,k}^{new} = -1$. Then, on the next iteration, we will again obtain $v_{j,k} = 1$. You should be able to see that the checkerboard error pattern "flips sign" from one iteration to the next. The checkerboard error is never reduced to zero by Jacobi iteration.

A strategy to overcome this problem is to "under-relax." To understand this approach, we first re-write (5.14) as

$$\hat{u}_{j,k}^{new} = \hat{u}_{j,k} + \left[\frac{1}{4}(h^2 f_{j,k} + \hat{u}_{j-1,k} + \hat{u}_{j+1,k} + \hat{u}_{j,k-1} + \hat{u}_{j,k+1}) - \hat{u}_{j,k}\right] . \tag{5.18}$$

This simply says that $\hat{u}_{j,k}^{new}$ is equal to $u_{j,k}$ plus an "increment." For the checkerboard error the increment given by Jacobi relaxation is too large; this is why the sign flips from one iteration to the next. We can reduce the increment by multiplying it by a factor less than one, called ω, i.e., we replace (5.18) by

$$\hat{u}_{j,k}^{new} = \hat{u}_{j,k} + \omega\left[\frac{1}{4}(h^2 f_{j,k} + \hat{u}_{j-1,k} + \hat{u}_{j+1,k} + \hat{u}_{j,k-1} + \hat{u}_{j,k+1}) - \hat{u}_{j,k}\right] . \tag{5.19}$$

where $0 < \omega < 1$. We can now rewrite (5.18) as

$$\hat{u}_{j,k}^{new} = \hat{u}_{j,k}(1-\omega) + \frac{\omega}{4}(h^2 f_{j,k} + \hat{u}_{j-1,k} + \hat{u}_{j+1,k} + \hat{u}_{j,k-1} + \hat{u}_{j,k+1}) . \tag{5.20}$$

Substitution of (5.15) into (5.20), with the use of (5.12), gives

$$v_{j,k}^{new} = v_{j,k}(1-\omega) + \frac{\omega}{4}[(\nu_{j-1,k} + \nu_{j+1,k} + \nu_{j,k-1} + \nu_{j,k+1})] \tag{5.21}$$

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.

Suppose that on a particular sweep the error is spatially uniform over the grid. Then, according to (5.17), the error will never change under Jacobi relaxation, and this is true even with under-relaxation, as can be seen from (5.21). 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.

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

5.4 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 u's that have already been updated, so that (5.14) is replaced by

$$\hat{u}_{j,k}^{new} = \frac{1}{4}(h^2 f_{j,k} + \hat{u}_{j-1,k}^{new} + \hat{u}_{j+1,k} + \hat{u}_{j,k-1}^{new} + \hat{u}_{j,k+1}) . \tag{5.22}$$

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 (5.22) 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 (5.14). It is only for the second and later rows of points that (5.22) 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 (5.12) exactly, the final result will be independent of where the sweep starts.

For Gauss-Seidel relaxation, the error-reduction formula corresponding to (5.17) is

$$v_{j,k}^{new} = \frac{1}{4}(v_{j-1,k}^{new} + v_{j+1,k} + v_{j,k-1}^{new} + v_{j,k+1}) \ . \tag{5.23}$$

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 u is spatially constant. We make the rather ill-considered first guess that the solution is a checkerboard:

$$\hat{u}_{j,k}^{0} = \begin{bmatrix} 1 & -1 & 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 & -1 & 1 \end{bmatrix} . \tag{5.24}$$

Here the superscript zero indicates 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:

$$\hat{u}_{j,k}^{1,\text{ partial}} = \begin{bmatrix} 1 & -1 & 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 & -1 & 1 \\ -0.5 & 0.25 & -0.281 & -1 & 1 & -1 \\ 1 & -0.5 & 0.625 & -0.593 & 0.602 & -0.60 \end{bmatrix} . \tag{5.25}$$

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.

5.5 Over-relaxation

The convergence of Gauss-Seidel relaxation can be accelerated by multiplying the increment by a factor *greater* than one. This is called "over-relaxation." Corresponding to (5.18), we write

$$\hat{u}_{j,k}^{new} = \hat{u}_{j,k}(1-\omega) + \frac{\omega}{4}(h^2 f_{j,k} + \hat{u}_{j-1,k}^{new} + \hat{u}_{j+1,k} + \hat{u}_{j,k-1}^{new} + \hat{u}_{j,k+1}) \ . \tag{5.26}$$

It can be shown that the convergence of (5.26) is optimized (i.e., made as rapid as possible) if we choose

$$\omega = \frac{2}{1+\sin(\pi h)} . \tag{5.27}$$

The algorithm represented by (5.26) and (5.27) is called "successive over-relaxation," or SOR. Choosing ω too large will cause the iteration to diverge. In practice, some experimentation may be needed to find the best value of ω .

5.6 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 (5.12) as

$$(-u_{j-1,k} + 2u_{j,k} - u_{j+1,k}) + (-u_{j,k-1} + 2u_{j,k} - u_{j,k+1}) = h^2 f_{j,k} . \tag{5.28}$$

The first quantity in parentheses on the left-hand side of (5.28) 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

$$[-\hat{u}^{int}_{j-1,k} + (2+r)\hat{u}^{int}_{j,k} - \hat{u}^{int}_{j+1,k}] + [-\hat{u}_{j,k-1} + (2-r)\hat{u}_{j,k} - \hat{u}_{j,k+1}] = h^2 f_{j,k} \tag{5.29}$$

for the values with superscript "*int*." Here r is a parameter used to control convergence, as discussed below. Eq. (5.29) represents a set of *tri-diagonal* systems, each of which can easily be solved. The sweep is completed by solving

$$[-\hat{u}^{new}_{j,k-1} + (2-r)\hat{u}^{new}_{j,k} - \hat{u}^{new}_{j,k+1}] + [-\hat{u}^{int}_{j-1,k} + (2+r)\hat{u}^{int}_{j,k} - \hat{u}^{int}_{j+1,k}] = h^2 f_{j,k} \tag{5.30}$$

as a second set of N tridiagonal systems. It can be shown that the ADI method converges if r is positive and constant for all sweeps. The optimal value of r is

$$r = 2\sin(\pi h) . \tag{5.31}$$

5.7 Multigrid methods

Fulton et al. (1986) summarize the multi-grid approach to solving boundary-value problems. The basic idea is very simple, and comes from the fact that the largest-scale features in the error field (the difference between the approximate solution and the true solution) take the largest number of sweeps to eliminate.

As we have already discussed, with Gauss-Seidel relaxation the *small-scale errors are eliminated quickly, while the large-scale errors are removed more slowly*. As the iteration proceeds, the error field becomes smoother at the same time that it undergoes an overall decrease in amplitude.

Essentially by definition of "large-scale," the large-scale errors can be represented on a relatively coarse grid. On such a coarse grid, the large-scale errors actually appear to be of "smaller" scale, in the sense that they are represented by fewer grid points. The large-scale error can thus be removed quickly by relaxing on a coarse grid.

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 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 is done on each grid in turn. The relaxations on the coarser grids remove the large-scale part of the error, while the relaxations 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.

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

5.8 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 the geometry of the problem is complex. 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 5.1 summarizes the operations counts and storage requirements of some well

Table 5.1: Well known methods for solving boundary value problems, and the operation count and storage requirements for each, measured in terms of N^2, the number of equations to be solved.

Method	Operation Count	Storage Requirement
Gaussian Elimination	N^4	N^3
Jacobi	N^4	N^2
Gauss-Seidel	N^4	$< N^2$

Table 5.1: Well known methods for solving boundary value problems, and the operation count and storage requirements for each, measured in terms of N^2, the number of equations to be solved.

Method	Operation Count	Storage Requirement
Successive Over-Relaxation	N^3	N^2
Alternating Direction Implicit	$N^3 \ln N$	N^2
Multigrid	N^2	N^2

known methods for solving boundary-value problems.

Problems

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

$$\nabla^2 u = \left(\sin\frac{4\pi x}{L}\right)\left(\cos\frac{4\pi y}{L}\right) \tag{5.32}$$

for the unknown function $u(x, y)$, where

$$\begin{aligned} 0 \le x \le L, \\ 0 \le y \le L. \end{aligned} \tag{5.33}$$

Assume that the domain-average value of u is zero. For simplicity, use $L = 4\pi\sqrt{2}$. Use centered second-order differences to approximate $\nabla^2 u$. 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. For each of the relaxation methods, try the following two initial guesses:

$$\begin{aligned} &1) \quad u_{j,k} = (-1)^{j+k}, \\ &2) \quad u_{j,k} = 0 \text{ everywhere.} \end{aligned} \tag{5.34}$$

Jacobi relaxation;

Jacobi under-relaxation;

Gauss-Seidel relaxation

Gauss-Seidel over-relaxation.

c) For Jacobi, Gauss-Seidel, and SOR, define the RMS error by

$$R^{\upsilon} \equiv \sqrt{\frac{1}{N^2}\sum_{j=1}^{N}\sum_{k=1}^{N}\left(\hat{u}^{\nu}_{j,k} - u_{j,k}\right)^2} \ . \tag{5.35}$$

Here $\hat{u}^{\nu}_{j,k}$ is the approximate solution, and $u_{j,k}$ is the “exact” solution of the finite-difference problem. The parameter ν is not an exponent; it is an “iteration counter,” i.e. $\nu = 0$ for the initial guess, $\nu = 1$ after one sweep, etc. Define the maximum absolute error by

$$M^{\nu} \equiv Max\forall(j,k)\{|\hat{u}_{j,k} - u_{j,k}|\} \ . \tag{5.36}$$

Let the convergence criterion be

$$M^{\nu} < 10^{-2} Max\forall(j,k)\{|u_{j,k}|\} \ . \tag{5.37}$$

How many iterations are needed to obtain convergence with Jacobi, Gauss-Seidel, and SOR? (Note that this convergence criterion refers to $u_{j,k}$, the exact solution of the finite-difference problem, and so could not be used in practice.)

d) Plot the error $u^{\nu}_{j,k} - u_{j,k}$, for all three methods, for $\nu = 0$ (all the same), $\nu = 10$, and $\nu = 20$.

e) Plot R^{ν} as a function of ν (or, if you prefer, as a function of $\ln\nu$) for all three methods.