7 Iterative methods for matrix equations

7.1 The need for iterative methods

We have seen that Gaussian elimination provides a method for finding the exact solution (if rounding errors can be avoided) of a system of equations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. However Gaussian elimination requires approximately $n^3/3$ operations (where n is the size of the system), which may become prohibitively time-consuming if n is very large. Another weakness is that Gaussian elimination requires us to store all the components of the matrix $\boldsymbol{A}$. In many real applications (especially the numerical solution of differential equations), the matrix $\boldsymbol{A}$ is *sparse*, meaning that most of its elements are zero, in which case keeping track of the whole matrix is wasteful.

In situations like these it may be preferable to adopt a method which produces an approximate rather than exact solution. We will describe three *iterative methods*, which start from an initial guess $\boldsymbol{x}_0$ and produce successively better approximations $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$. The iteration can be halted as soon as an adequate degree of accuracy is obtained, and the hope is that this takes a significantly shorter time than the exact method of Gaussian elimination would require.

7.2 Splitting the matrix

All the methods we will consider involve splitting the matrix $\boldsymbol{A}$ into the difference between two new matrices $\boldsymbol{S}$ and $\boldsymbol{T}$:

$$\boldsymbol{A} = \boldsymbol{S} - \boldsymbol{T}.$$

Thus the equation $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ gives

$$\boldsymbol{S}\boldsymbol{x} = \boldsymbol{T}\boldsymbol{x} + \boldsymbol{b},$$

based on which we can try the iteration

$$\boldsymbol{S}\boldsymbol{x}_{k+1} = \boldsymbol{T}\boldsymbol{x}_k + \boldsymbol{b}. \tag{7.1}$$

Now *if* this procedure converges, say $\boldsymbol{x}_k \to \boldsymbol{x}$ as $k \to \infty$, then clearly $\boldsymbol{x}$ solves the original problem $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, but it is not at all clear from the outset whether a scheme like (7.1) converges or not.

Evidently there are many possible ways to split the matrix $\boldsymbol{A}$. The tests of a good choice are:

- the new vector $\boldsymbol{x}_{k+1}$ should be *easy to compute*, that is $\boldsymbol{S}$ should be easily invertible (for example $\boldsymbol{S}$ might be diagonal or triangular);
- the scheme should *converge as rapidly as possible* towards the true solution.

These two requirements are conflicting: a choice of splitting which is particularly easy to invert (see *e.g.* Jacobi's method below) may not converge especially rapidly (or at all). At the other extreme we can converge *exactly*, in just one step, by using $\boldsymbol{S} = \boldsymbol{A}$, $\boldsymbol{T} = \boldsymbol{0}$; but $\boldsymbol{S} = \boldsymbol{A}$ is usually difficult to invert: that's the whole point of splitting!

It is convenient to introduce the notation

$$\boldsymbol{A} = \boldsymbol{L} + \boldsymbol{D} + \boldsymbol{U} \quad (= \boldsymbol{S} - \boldsymbol{T}),$$

where $\boldsymbol{L}$ is strictly lower triangular,
$\boldsymbol{D}$ is diagonal,
$\boldsymbol{U}$ is strictly upper triangular.

For example, if

$$\boldsymbol{A} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix},$$

then

$$\boldsymbol{L} = \begin{pmatrix} 0 & 0 & 0 \\ 4 & 0 & 0 \\ 7 & 8 & 0 \end{pmatrix}, \quad \boldsymbol{D} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 9 \end{pmatrix}, \quad \boldsymbol{U} = \begin{pmatrix} 0 & 2 & 3 \\ 0 & 0 & 6 \\ 0 & 0 & 0 \end{pmatrix}.$$

We will consider three possible splittings.

1. **Jacobi's method** $\boldsymbol{S} = \boldsymbol{D}$, $\boldsymbol{T} = -(\boldsymbol{L} + \boldsymbol{U})$;
2. **The Gauss-Seidel method** $\boldsymbol{S} = \boldsymbol{L} + \boldsymbol{D}$, $\boldsymbol{T} = -\boldsymbol{U}$;
3. **Successive over-relaxation (SOR)** a combination of 1 and 2.

7.3 Jacobi's method

In Jacobi's method, $\boldsymbol{S}$ is simply the diagonal part of $\boldsymbol{A}$. We illustrate it with a simple two-dimensional example.

Example 7.1 *Consider the system*

$$\begin{aligned} 2x - y &= 3, \\ -x + 2y &= 0. \end{aligned}$$

We solve the first equation for x and the second for y:

$$\begin{aligned} x &= y/2 + 3/2, \\ y &= x/2, \end{aligned}$$

which suggests the iterative scheme

$$\begin{aligned} x_{k+1} &= y_k/2 + 3/2 \\ y_{k+1} &= x_k/2. \end{aligned}$$

Note that this corresponds to $\boldsymbol{S}\boldsymbol{x}_{k+1} = \boldsymbol{T}\boldsymbol{x}_k + \boldsymbol{b}$ *where the splitting*

$$\boldsymbol{S} = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}, \quad \boldsymbol{T} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

has been employed.

Suppose we start with the initial guess

$$\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Then we generate successively

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} 2 \\ 1/2 \end{pmatrix}, \quad \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} 7/4 \\ 1 \end{pmatrix}, \quad \begin{pmatrix} x_3 \\ y_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 7/8 \end{pmatrix},$$

and so forth, which appears to be converging nicely to the solution $x = 2$, $y = 1$.

7.4 The Gauss-Seidel method

A drawback with Jacobi's method is that it requires us to store all the components of $\boldsymbol{x}_k$ until we have finished computing the next iteration $\boldsymbol{x}_{k+1}$. For very large systems, the memory required may become a problem. In any case, it would appear to make more sense to use the "new" values of $\boldsymbol{x}$ (which presumably are more accurate than the old values) as soon as we've calculated them. The scheme which results from doing so is called the Gauss-Seidel method. We illustrate it with the same two-dimensional system as in example 7.1.

Example 7.2 *Consider the same system*

$$\begin{aligned} 2x - y &= 3 \\ -x + 2y &= 0, \end{aligned}$$

as in example 7.1. *As in Jacobi's method we use the first equation to find* x_{k+1} *in terms of* y_k*:*

$$x_{k+1} = y_k/2 + 3/2.$$

But now that we've found x_{k+1} *we* use it *when working out* y_{k+1}*:*

$$y_{k+1} = x_{k+1}/2$$

(Jacobi has $x_k/2$ *on the right-hand side). In terms of splitting this corresponds to the choice*

$$\boldsymbol{S} = \begin{pmatrix} 2 & 0 \\ -1 & 2 \end{pmatrix}, \quad \boldsymbol{T} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix},$$

that is Gauss-Seidel corresponds to making $\boldsymbol{S}$ lower triangular and $\boldsymbol{T}$ strictly upper triangular.

Again we start with the initial guess

$$\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix},$$

and find that

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix},$$

so for this example, Gauss-Seidel converges to the exact solution after just one iteration.

It is a fluke that the scheme in example 7.2 converges in one step, but it is generally the case that Gauss-Seidel converges better than Jacobi, and on the whole it's a better method.

7.5 Implementation of Gauss-Seidel

Now consider the general $n \times n$ system

$$\begin{array}{rcl} a_{11}x_1 + a_{12}x_2 + \quad \ldots + a_{1n}x_n & = & b_1, \\ a_{21}x_1 + a_{22}x_2 + \quad \ldots + a_{2n}x_n & = & b_2, \\ \vdots \qquad\qquad\qquad\qquad\quad \vdots & & \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \quad \ldots + a_{nn}x_n & = & b_n. \end{array}$$

The Gauss-Seidel scheme proceeds exactly as in example 7.2. We begin by using the first equation to update x_1:

$$(x_1)_{\text{new}} = \frac{1}{a_{11}} \left[b_1 - \sum_{j=2}^{n} a_{1j}x_j \right].$$

Next we use the second equation to update x_2:

$$(x_2)_{\text{new}} = \frac{1}{a_{22}} \left[b_2 - \sum_{j=3}^{n} a_{2j}x_j \right] - \frac{a_{21}}{a_{22}}x_1.$$

Note that the value of x_1 used in this second step is the "new" updated value: we don't bother to store the "old" value of x_1.

The right-hand side can be rearranged to

$$(x_2)_{\text{new}} = \frac{1}{a_{22}} \left[b_2 - \sum_{j=1}^{n} a_{2j}x_j + a_{22}x_2 \right],$$

or

$$(x_2)_{\text{new}} = x_2 + \frac{1}{a_{22}} \left[b_2 - \sum_{j=1}^{n} a_{2j}x_j \right].$$

This form persists for the subsequent steps, so the general formula for updating x_i is

$$(x_i)_{\text{new}} = x_i + \frac{1}{a_{ii}} \left[b_i - \sum_{j=1}^{n} a_{ij} x_j \right]. \tag{7.2}$$

On the right-hand side of equation (7.2), the first term is the "old" value of x_i, and the second term is the *correction*, which is added to x_i to obtain the improved value $(x_i)_{\text{new}}$. Note also from the form of (7.2) that for the scheme to work *we must set up the system in such a way that its diagonal entries are all nonzero.*

We can estimate the number of operation required for the Gauss-Seidel method as follows. Each step of the form (7.2) requires n multiplications and 1 division, making a total of $n+1$ operations. These are required for each of the n components of $\boldsymbol{x}$, so that $n(n+1) \approx n^2$ operations are needed for one complete iteration. Thus the total number of operations required is approximately n^2 times the number of iterations required for satisfactory convergence. This should be compared with the $n^3/3$ operations needed for Gaussian elimination. We can observe that Gauss-Seidel (or something similar) may indeed be preferable to Gaussian elimination if n is very large and *the number of iterations required can be kept reasonably low.* We should also observe that in the operations count carried out above, the number of multiplications required may be greatly reduced if the matrix A is sparse.

The question arises: *how can we tell if the scheme has converged adequately?* That is, how do we decide when enough iterations have been performed? One possibility is to look at the difference between successive values of $\boldsymbol{x}_k$, since these become closer as $\boldsymbol{x}_k$ approaches the solution $\boldsymbol{x}$. Thus we might, for example, choose to continue the iterations until $|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k|$ is smaller than some prescribed small amount. A slightly better approach is to use the *relative* difference (*i.e.* compared to the actual size of the solution) as a measure of how well the scheme has converged. Thus the iteration is halted when

$$\frac{|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k|}{|\boldsymbol{x}_{k+1}|} < \delta,$$

where δ is some prescribed small number. In the Gauss-Seidel step this is very easy to calculate, since $(x_i)_{k+1} - (x_i)_k$ is just the second term on the right-hand side of (7.2).

We implement these ideas in the following fragment of code in FORTRAN90 which performs a single iteration of the Gauss-Seidel scheme.

```
!
!     This code performs a single Gauss-Seidel iteration
!
      error = 0.0
      do i = 1, n
         temp = b(i)
         do j = 1, n
            temp = temp - a(i,j) * x(j)
         end do
```

```
         temp = temp / a(i,i)
!
!        Now temp contains the correction to be added to x(i),
!        that is (x(i))_(k+1) - (x(i))_k
!
         error = error + temp**2
         x(i) = x(i) + temp
      end do
      error = sqrt(error)
!
!     Now error contains |x_(k+1) - x_k|
!
```

All that remains is to perform this loop until the relative error is sufficiently small. We should also allow for the fact that the scheme may not converge at all. The following code demonstrates a straightforward implementation, where the "iteration step" is the code fragment given above.

```
... preamble ...
   do k = 1, 30
... iteration step ...
         if (error / sqrt(DOT_PRODUCT(x,x)) < 0.001) exit
   end do
   if (k > 30) then                ! Scheme hasn't converged
      write(*,*) 'Method failed to converge after 30 iterations'
   else                            ! Scheme has converged
      write(*,*) 'Method converged after', k, 'iterations'
      write(*,*) 'Solution is'
      do i = 1, n
         write(*,*) x(i)
      end do
   end if
... etc. ...
```

Note that here we use the function `DOT_PRODUCT` to calculate $|\boldsymbol{x}_k|$.

7.6 Convergence of Jacobi and Gauss-Seidel

In general our iterative process takes the form (7.1). Of course, for the method to be at all sensible $\boldsymbol{S}$ must be invertible, so we can write

$$\boldsymbol{x}_{k+1} = \boldsymbol{S}^{-1}\boldsymbol{T}\boldsymbol{x}_k + \boldsymbol{S}^{-1}\boldsymbol{b}.$$

The combination $\left(\boldsymbol{S}^{-1}\boldsymbol{T}\right)$ is known as the *iteration matrix*.

Now we set

$$\boldsymbol{x}_k = \boldsymbol{x} + \boldsymbol{\epsilon}_k,$$

where $\boldsymbol{x}$ is the exact solution (*i.e.* it satisfies $\boldsymbol{x} = \boldsymbol{S}^{-1}\boldsymbol{T}\boldsymbol{x} + \boldsymbol{S}^{-1}\boldsymbol{b}$) and $\boldsymbol{\epsilon}_k$ is the error after k iterations. Then $\boldsymbol{\epsilon}_k$ satisfies

$$\boldsymbol{\epsilon}_{k+1} = \left(\boldsymbol{S}^{-1}\boldsymbol{T}\right)\boldsymbol{\epsilon}_k,$$

and the question is *does* $\boldsymbol{\epsilon}_k \to \boldsymbol{0}$ *as* $k \to \infty$*?*

We will answer this question under the assumption that *the iteration matrix* $\boldsymbol{S}^{-1}\boldsymbol{T}$ *has a complete set of* n *linearly independent eigenvectors*, say $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n$, with corresponding eigenvalues $\lambda_1, \ldots, \lambda_n$. Thus we can use $\{\boldsymbol{e}_i\}$ as a basis and write

$$\boldsymbol{\epsilon}_0 = \sum_{i=1}^{n} c_i \boldsymbol{e}_i,$$

for some constants c_i, where $\boldsymbol{\epsilon}_0$ is the error in the initial guess $\boldsymbol{x}_0$. Then

$$\boldsymbol{\epsilon}_1 = \boldsymbol{S}^{-1}\boldsymbol{T}\boldsymbol{\epsilon}_0 = \sum_{i=1}^{n} c_i \lambda_i \boldsymbol{e}_i,$$

and in general

$$\boldsymbol{\epsilon}_k = \sum_{i=1}^{n} c_i \lambda_i^k \boldsymbol{e}_i. \tag{7.3}$$

So $\boldsymbol{\epsilon}_k$ tends to zero as $k \to \infty$ if and only if $|\lambda_i| < 1$ for all i. That is, the iteration (7.1) converges if and only if *the eigenvalues of the iteration matrix* $\boldsymbol{S}^{-1}\boldsymbol{T}$ *all have absolute value less than one.*

Another way of stating this result is as follows. Define the *spectral radius* $\rho(M)$ of a matrix $\boldsymbol{M}$ to be the largest value of $|\lambda_i|$ over the eigenvalues λ_i of $\boldsymbol{M}$:

$$\rho(\boldsymbol{M}) = \max_i \left|\lambda_i\right|.$$

Then the iterative scheme (7.1) converges if and only if $\rho(\boldsymbol{S}^{-1}\boldsymbol{T}) < 1$.

Another important aspect of the spectral radius is its implications for the rate at which the scheme converges. If (without loss of generality) λ_1 is the largest eigenvalue of $\boldsymbol{S}^{-1}\boldsymbol{T}$ in absolute value (with $|\lambda_1| = \rho$) then clearly (7.3) implies that the behaviour of the error $\boldsymbol{\epsilon}_k$ is dominated by λ_1^k as $k \to \infty$:

$$\begin{aligned} \boldsymbol{\epsilon}_k &\sim c_1 \lambda_1^k \boldsymbol{e}_1 \text{ as } k \to \infty \\ \Rightarrow |\boldsymbol{\epsilon}_k| &\sim \text{const. } \rho^k \text{ as } k \to \infty. \end{aligned}$$

Thus *the error is reduced by roughly a factor of* ρ *at each iteration* and so *the smaller* ρ *is, the faster the scheme converges.*

With this in mind, the *rate of convergence* is often defined as $-\log \rho$ when $\rho < 1$.

Finally we reiterate that we have assumed throughout this section that the iteration matrix has a complete set of eigenvectors. This may seem at first glance to be a dangerous assumption, but the following points should be borne in mind.

- A matrix is guaranteed to have a complete set of eigenvectors so long as all its eigenvalues are distinct.
- Thus the case in which two or eigenvalues are repeated, in which there is the possibility that the set of eigenvectors might be incomplete, is not generic.
- In any case, the results of this section *can*, with a little more work, be shown to hold even when $\boldsymbol{S}^{-1}\boldsymbol{T}$ does not have a complete set of eigenvectors.

Now we illustrate the calculation of the spectral radius for the Jacobi and Gauss-Seidel schemes considered in examples 7.1 and 7.2.

Example 7.3 *The Jacobi iteration from* example 7.1*,*

$$\begin{aligned} x_{k+1} &= y_k/2 + 3/2, \\ y_{k+1} &= x_k/2, \end{aligned}$$

can be written as $\boldsymbol{x}_{k+1} = \boldsymbol{S}^{-1}\boldsymbol{T}\boldsymbol{x}_k + \boldsymbol{S}^{-1}\boldsymbol{b}$*, where*

$$\boldsymbol{S}^{-1}\boldsymbol{T} = \begin{pmatrix} 0 & 1/2 \\ 1/2 & 0 \end{pmatrix}, \quad \boldsymbol{S}^{-1}\boldsymbol{b} = \begin{pmatrix} 3/2 \\ 0 \end{pmatrix}.$$

The eigenvalues λ *of* $\boldsymbol{S}^{-1}\boldsymbol{T}$ *are given by*

$$\begin{vmatrix} -\lambda & 1/2 \\ 1/2 & -\lambda \end{vmatrix} = 0 \quad \Rightarrow \quad \lambda^2 = 1/4 \quad \Rightarrow \quad \lambda = \pm 1/2.$$

Thus the spectral radus $\rho_{\text{Jacobi}} = 1/2 < 1$ *and the scheme therefore converges.*

Example 7.4 *For the Gauss-Seidel iteration from* example 7.2*, we have*

$$\boldsymbol{S} = \begin{pmatrix} 2 & 0 \\ -1 & 2 \end{pmatrix}, \quad \boldsymbol{T} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \quad \Rightarrow \quad \boldsymbol{S}^{-1}\boldsymbol{T} = \frac{1}{4}\begin{pmatrix} 2 & 0 \\ 1 & 2 \end{pmatrix}\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1/2 \\ 0 & 1/4 \end{pmatrix},$$

using the usual formula for the inverse of a 2×2 *matrix. The eigenvalues are given by*

$$\begin{vmatrix} -\lambda & 1/2 \\ 0 & 1/4 - \lambda \end{vmatrix} = 0 \quad \Rightarrow \quad \lambda(\lambda - 1/4) = 0 \quad \Rightarrow \quad \lambda = 0 \text{ or } 1/4,$$

and the spectral radius $\rho_{\text{G-S}} = 1/4$*.*

Thus for the system considered in these examples the Gauss-Seidel scheme converges *twice as fast* as the Jacobi scheme. Recall that the error is roughly speaking reduced by a factor of ρ at each iteration. Thus it takes two Jacobi steps to reduce the error by 25% while Gauss-Seidel does the same job in one step.

This rule of thumb — "each Gauss-Seidel step is worth two Jacobi steps" — holds for many examples, in particular all convergent two-dimensional systems. However it is possible to construct larger systems for which Jacobi converges and Gauss-Seidel does not (or vice versa).

One final point worth emphasising is that it may not be immediately obvious how to order the equations. Apart from ensuring that the diagonal entries are all nonzero, there appears to be quite a bit of freedom (there are $n!$ ways of arranging n equations).

Example 7.5 *The system from* example 7.1 *can be rearranged to*

$$\begin{aligned} -x + 2y &= 0, \\ 2x - y &= 3, \end{aligned}$$

which suggests the Jacobi scheme

$$\begin{aligned} x_{k+1} &= 2y_k, \\ y_{k+1} &= 2x_k - 3. \end{aligned}$$

The iteration matrix for this scheme is

$$\boldsymbol{S}^{-1}\boldsymbol{T} = \begin{pmatrix} 0 & 2 \\ 2 & 0 \end{pmatrix},$$

whose eigenvalues are $\lambda = \pm 2$. *Thus* $\rho = 2 > 1$ *and, with the equations ordered this way, Jacobi's method* diverges. *It is straightforward to show that Gauss-Seidel does too.*

The situation demonstrated by example 7.5 is quite general. For two-dimensional systems it is usually the case that one of the obvious Jacobi schemes converges while the other diverges, and the corresponding Gauss-Seidel schemes behave likewise. For larger systems it is not obvious in advance whether or not a particular Jacobi or Gauss-Seidel scheme will converge, or what permutation of the equations gives the best convergence. However it is clearly a good idea to try to *make the diagonal be the largest entry in each row* — this will make $\boldsymbol{S}^{-1}\boldsymbol{T}$ as small as possible.

In practice it is usually *not* feasible to compute the eigenvalues of $\boldsymbol{S}^{-1}\boldsymbol{T}$ when the system is large. Therefore some trial and error may be required to formulate a scheme like Jacobi or Gauss-Seidel in such a way that it converges. However, for many classes of matrices which crop up particularly frequently (*e.g.* in the numerical solution of differential equations) the spectral radii of the Jacobi and Gauss-Seidel iteration matrices *are* known, and can be shown to be smaller than one.

7.7 Successive over-relaxation (SOR)

The origin of the SOR method is the observation that Gauss-Seidel converges monotonically: the approximations $\boldsymbol{x}_k$ stay on the same side of the solution $\boldsymbol{x}$ as k increases. This suggests going slightly further than Gauss-Seidel tells us to at each step. We introduce an *over-relaxation* or *acceleration parameter* ω: when $\omega = 1$ SOR is identical to Gauss-Seidel and acceleration is achieved by increasing ω. The best choice of ω depends on the problem being solved, but it always lies between 1 and 2.

For SOR the matrix splitting is

$$(\boldsymbol{D} + \omega\boldsymbol{L})\,\boldsymbol{x}_{k+1} = [(1-\omega)\boldsymbol{D} - \omega\boldsymbol{U}]\,\boldsymbol{x}_k + \omega\boldsymbol{b}. \tag{7.4}$$

Notice that

1. if the method converges, *i.e.* $\boldsymbol{x}_k \to \boldsymbol{x}$ as $k \to \infty$, then $\boldsymbol{x}$ satisfies the original problem $\boldsymbol{A}\boldsymbol{x} = (\boldsymbol{L} + \boldsymbol{D} + \boldsymbol{U})\boldsymbol{x} = \boldsymbol{b}$;

2. when $\omega = 1$, (7.4) is identical to the splitting for Gauss-Seidel.

The iteration matrix is

$$\boldsymbol{M}_\omega = (\boldsymbol{D} + \omega\boldsymbol{L})^{-1}\left[(1-\omega)\boldsymbol{D} - \omega\boldsymbol{U}\right],$$

and the point of SOR is to discover the value of ω which minimises the spectral radius of $\boldsymbol{M}_\omega$.

Example 7.6 *For the same system as in* example 7.1*,*

$$\boldsymbol{A} = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix} \quad \Rightarrow \boldsymbol{L} = \begin{pmatrix} 0 & 0 \\ -1 & 0 \end{pmatrix}, \quad \boldsymbol{D} = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}, \quad \boldsymbol{U} = \begin{pmatrix} 0 & -1 \\ 0 & 0 \end{pmatrix}.$$

Thus the SOR iteration matrix is

$$\begin{aligned} \boldsymbol{M}_\omega &= \begin{pmatrix} 2 & 0 \\ -\omega & 2 \end{pmatrix}^{-1} \begin{pmatrix} 2(1-\omega) & \omega \\ 0 & 2(1-\omega) \end{pmatrix} = \frac{1}{4}\begin{pmatrix} 2 & 0 \\ \omega & 2 \end{pmatrix}\begin{pmatrix} 2(1-\omega) & \omega \\ 0 & 2(1-\omega) \end{pmatrix} \\ &= \begin{pmatrix} 1-\omega & \omega/2 \\ \omega(1-\omega)/2 & 1-\omega+\omega^2/4 \end{pmatrix}. \end{aligned}$$

The eigenvalues of $\boldsymbol{M}_\omega$ *are given by*

$$\begin{vmatrix} 1-\omega-\lambda & \omega/2 \\ \omega(1-\omega)/2 & 1-\omega+\omega^2/4-\lambda \end{vmatrix} = 0$$

and thus

$$\lambda^2 - (2 - 2\omega + \omega^2/4)\lambda + (\omega-1)^2 = 0. \tag{7.5}$$

When $\omega = 1$ *this gives* $\lambda = 0$ *and* $1/4$ *as in* example 7.4*. As* ω *is increased, the smaller eigenvalue increases while the larger one decreases (their product is* $(\omega-1)^2$*) and it can be shown that* the optimal value of ω occurs when the two eigenvalues are equal.

Therefore set $\lambda_1 = \lambda_2 = \lambda$*. From the quadratic equation (7.5) we can read off the sum and product of the eigenvalues:*

$$\lambda_1\lambda_2 = \lambda^2 = (\omega-1)^2 \quad \Rightarrow \quad \lambda = \omega - 1,$$

and

$$\begin{aligned} \lambda_1 + \lambda_2 = 2\lambda = 2 - 2\omega + \omega^2/4 \quad &\Rightarrow \quad 2(\omega-1) = 2 - 2\omega + \omega^2/4 \\ &\Rightarrow \quad \omega^2 - 16\omega + 16 = 0. \end{aligned}$$

The roots of this quadratic are $\omega = 4(2 \pm \sqrt{3})$*, of which we take the one between* 1 *and* 2*:*

$$\omega = 4(2 - \sqrt{3}) \approx 1.072$$

is the optimal value of the acceleration parameter for this system.

At this value of ω *the two equal eigenvalues of* $\boldsymbol{M}_\omega$ *are*

$$\lambda = \omega - 1 \approx 0.072,$$

and thus the spectral radius

$$\rho \approx 0.072.$$

Notice the improvement on Gauss-Seidel, which had $\rho = 1/4$*. For this example SOR is nearly twice as good again as Gauss-Seidel (since* $(1/4)^2 = 0.0625$ *is only slightly less than* 0.072*).*

The observation that the optimal value of ω occurs when two eigenvalues coincide holds for other systems also. However for large systems it is not practicable to find the eigenvalues explicitly and thus calculate the optimum value of ω as shown in example 7.6. Instead the best choice of ω must often be determined by trial and error, but it's certainly worth the effort. For large systems the improvement in convergence compared with Gauss-Seidel is typically much more dramatic than that seen in the two-dimensional example 7.6: each SOR step could be worth thirty or more Gauss-Seidel steps!

7.8 Implementation of SOR

In general, if $\boldsymbol{A}$ has elements a_{ij} then the SOR method is

$$\begin{pmatrix} a_{11} & 0 & 0 & \dots \\ \omega a_{21} & a_{22} & 0 & \dots \\ \omega a_{31} & \omega a_{32} & a_{33} & \dots \\ \vdots & \vdots & \vdots & \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \end{pmatrix}_{k+1} = \begin{pmatrix} \omega b_1 \\ \omega b_2 \\ \omega b_3 \\ \vdots \end{pmatrix} +$$

$$+ \begin{pmatrix} (1-\omega)a_{11} & -\omega a_{12} & -\omega a_{13} & \dots \\ 0 & (1-\omega)a_{22} & -\omega a_{23} & \dots \\ 0 & 0 & (1-\omega)a_{33} & \dots \\ \vdots & \vdots & \vdots & \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \end{pmatrix}_{k} .$$

By rearranging this as we did for Gauss-Seidel, we see that each step of SOR is of the form

$$(x_i)_{\text{new}} = x_i + \frac{\omega}{a_{ii}} \left[b_i - \sum_{j=1}^{n} a_{ij} x_j \right] . \tag{7.6}$$

Now the connection with Gauss-Seidel is much clearer: at each step we simply multiply the Gauss-Seidel correction by the acceleration parameter ω. Thus it is a simple matter to modify the code given in section 7.5 to produce an SOR code.

Exercises

7.1. Consider the linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{A}$ is the matrix

$$\boldsymbol{A} = \begin{pmatrix} 3 & 2 & 1 \\ 2 & 3 & 2 \\ 1 & 2 & 3 \end{pmatrix} .$$

(a) Derive the iteration matrices for the Jacobi, Gauss-Seidel and SOR methods.

(b) Hence show that the Jacobi method diverges, but the Gauss-Seidel method converges.

7.2. (a) Write a subroutine which generates the $n \times n$ *finite difference matrix* $\boldsymbol{F}_n$ whose elements f_{ij} are defined by

$$f_{ij} = \begin{cases} 2 & i = j, \\ -1 & i = j \pm 1, \\ 0 & \text{otherwise}, \end{cases}$$

for any value of n. For example, when $n = 4$,

$$F_4 = \begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix}.$$

Write a program which prints out the matrix $\boldsymbol{F}_{10}$ to check that your subroutine works.

(b) Write a program (using the subroutine from 7.2a) which solves the system of equations $\boldsymbol{F}_n\boldsymbol{x} = \boldsymbol{b}_n$ for $\boldsymbol{x}$ using the SOR method, where $\boldsymbol{F}_n$ is the finite difference matrix defined above and $\boldsymbol{b}_n$ is the n-dimensional vector $\boldsymbol{b}_n = (1, 1, \ldots, 1)^T$. The dimension n, acceleration parameter ω and desired accuracy of the solution should be set by the user at run time, and the program should output the solution $\boldsymbol{x}$ and the number of iterations required for convergence. Test your program with the values $n = 8$, $\omega = 1.3$, desired accuracy $= 0.0001$.

(c) Modify the program written in 7.2b to tabulate the number of iterations needed to achieve an accuracy of 0.0001 versus ω, where $\omega = 1, 1.05, \ldots, 1.95, 2$. Print out the table of values generated when $n = 10$, and thus estimate the optimal value of ω when $n = 10$. How much better than Gauss-Seidel is SOR in this case?

7.3. [1998/99 Question 1]

(a) Define the spectral radius $\rho(\boldsymbol{M})$ of a square matrix $\boldsymbol{M}$.

An iterative scheme is given by $\boldsymbol{x}_{n+1} = \boldsymbol{b} + \boldsymbol{M}\boldsymbol{x}_n$, where $\boldsymbol{b}$ and $\boldsymbol{x}_n$ are vectors. Give the condition under which the iterative scheme converges, in terms of $\rho(\boldsymbol{M})$.

(b) Consider the system of equations

$$\begin{aligned} 2x &+ 3y &&= 7, \\ x &- y &&= 1. \end{aligned} \qquad (*)$$

(i). Show that a Jacobi scheme based on writing $x = (7 - 3y)/2$, $y = x - 1$ fails to converge.

(ii). Reformulate the problem in such a way that a Jacobi scheme does converge.

(iii). Formulate a convergent Gauss-Seidel method for the system (*), showing that the method is convergent. Compute two iterations of the method, starting from $x = 0$, $y = 0$.

(iv). Use the method obtained in (iii) to write down an S.O.R. scheme for (*) and determine the value of the acceleration parameter that gives the fastest convergence.