Chapter 7
Iterative Methods

Abstract This chapter looks at iterative methods to solve linear systems and at some alternative methods to solve eigenvalue problems. That is, we now look at *iteration instead of using a finite number of noniterative steps*. Iterative methods for solving eigenvalue problems are, of course, completely natural. We looked at power iteration and at the QR iteration in Chap. 5; here we look at some methods that *take advantage of sparsity or structure*. We also use *one pass of iterative refinement* to *improve structured backward error*. ◁

7.1 Iterative Refinement and Structured Backward Error

Let us begin with the simplest possible iterative method for solving a linear system. We first consider a 3×3 example that hardly needs iteration, but we will shortly extend to larger matrix sizes. So suppose we wish to solve

$$\begin{bmatrix} 4 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}.$$

The exact solution, which is easy to find by any method, is $\mathbf{x} = [5, -6, 5]/14$. Let us imagine that we don't know that, but that due to a prior computation, we do know that the matrix

$$\mathbf{B} = \begin{bmatrix} 2+\sqrt{3} & 1 & & \\ 1 & & 4 & 1 \\ & & 1 & 4 \end{bmatrix}$$

R.M. Corless and N. Fillion, *A Graduate Introduction to Numerical Methods: From the Viewpoint of Backward Error Analysis*, DOI 10.1007/978-1-4614-8453-0_7,

has the Cholesky factoring $\mathbf{LDL}^T$ with

$$\mathbf{L} = \begin{bmatrix} 1 & & \\ \alpha & 1 & \\ & \alpha & 1 \end{bmatrix},$$

$\alpha = 1/(2+\sqrt{3})$, and $\mathbf{D} = \mathrm{diag}(2+\sqrt{3}, 2+\sqrt{3}, 2+\sqrt{3})$. As a result, $\mathbf{B}^{-1} = \mathbf{L}^{-T}\mathbf{D}^{-1}\mathbf{L}^{-1}$ is easy to compute, or, more properly,

$$\mathbf{Bx} = \mathbf{b} \quad \Leftrightarrow \quad \mathbf{LDL}^T\mathbf{x} = \mathbf{b}$$

is easy to solve. Here, if we let $\mathbf{P} = \mathbf{B}^{-1}$ (at least in thinking about it, not in actually doing it), we have

$$\mathbf{PAx} = \mathbf{Pb} = \begin{bmatrix} 0.3847 \\ -0.4359 \\ 0.35898 \end{bmatrix}.$$

Notice that $\mathbf{PA}$ is nearly the identity, that is, $\mathbf{PA} = \mathbf{I} - \mathbf{S}$, where $\mathbf{S}$ is a matrix with small entries:

$$\mathbf{S} = \begin{bmatrix} -0.073216421430700 & 0 & 0 \\ 0.0206191045714862 & 0 & 0 \\ -0.00515477614287156 & 0 & 0 \end{bmatrix}.$$

Our equation has thus become

$$(\mathbf{I} - \mathbf{S})\mathbf{x} = \mathbf{Pb} = \begin{bmatrix} 0.3847 \\ -0.4359 \\ 0.35898 \end{bmatrix},$$

and we are left with the problem, seemingly as difficult, of solving a linear system with matrix $\mathbf{I} - \mathbf{S}$. However, we have made some progress, since we can use the smallness of $\mathbf{S}$ to solve the system by means of an iterative scheme. First, observe that $(\mathbf{I} - \mathbf{S})\mathbf{x} = \mathbf{Pb} = \mathbf{x}_0$ implies $\mathbf{x} = \mathbf{x}_0 + \mathbf{Sx}$. Hence, we can then write the following natural iteration:

$$\mathbf{x}_{k+1} = \mathbf{x}_0 + \mathbf{Sx}_k\,.$$

This is the *Richardson iteration*, which is about as simple an iterative method as it gets. Then we obtain

$$\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{Sx}_0\,.$$

Similarly, we find that

$$\begin{aligned} \mathbf{x}_2 &= \mathbf{x}_0 + \mathbf{Sx}_0 + \mathbf{S}^2\mathbf{x}_0 \\ \mathbf{x}_3 &= \mathbf{x}_0 + \mathbf{Sx}_0 + \mathbf{S}^2\mathbf{x}_0 + \mathbf{S}^3\mathbf{x}_0. \end{aligned}$$

In general, the kth iteration results in

$$\mathbf{x}_k = \sum_{j=0}^{k} \mathbf{S}^j \mathbf{x}_0 \,.$$

This series converges if $\|\mathbf{S}^k\|$ goes to zero, which it does, exactly as for the geometric series, if there is a $\rho < 1$ for which $\|\mathbf{S}^k\| \le \rho^k$. In this case, $\|\mathbf{S}\| \le 0.01$. An obvious induction gives $\|\mathbf{S}^k\| \le \|\mathbf{S}\|^k \le (0.01)^k$ and so this iteration converges; indeed, already $\mathbf{x}_4$ is correct to four digits. Note that $\max|\lambda| \le \|\mathbf{S}\|$ in general, and it is very possible that $\max|\lambda| < 1$. In this case the powers eventually decay even though $\|\mathbf{S}\| > 1$. We will see examples shortly.

Before we look at larger matrices, let's look at this iteration in a different way. Using a matrix $\mathbf{P}$, which is close to the inverse of $\mathbf{A}$, we make the initial guess $\mathbf{x}_0 = \mathbf{Pb}$ (since $\mathbf{Ax} = \mathbf{b}$ then implies $\mathbf{x} \approx \mathbf{Pb}$). The residual resulting from this choice is

$$\mathbf{r}_0 = \mathbf{b} - \mathbf{Ax}_0 = \mathbf{b} - \mathbf{APb} \,.$$

Since $\mathbf{0} = \mathbf{b} - \mathbf{Ax}$, we find that

$$\mathbf{r}_0 = \mathbf{b} - \mathbf{Ax}_0 - (\mathbf{b} - \mathbf{Ax}) = \mathbf{Ax} - \mathbf{Ax}_0 = \mathbf{A}(\mathbf{x} - \mathbf{x}_0) = \mathbf{A}\Delta\mathbf{x} \,.$$

Thus, we see that $\Delta\mathbf{x} = \mathbf{x} - \mathbf{x}_0$ solves

$$\mathbf{A}\Delta\mathbf{x} = \mathbf{r}_0 \,.$$

Now, with this equation, we can use $\mathbf{P}$ as above and let $\mathbf{x}_1 - \mathbf{x}_0 = \mathbf{Pr}_0$. Then

$$\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{Pr}_0 \,.$$

The process can clearly be repeated:

$$\begin{aligned} \mathbf{x}_2 &= \mathbf{x}_1 + \mathbf{Pr}_1 \\ \mathbf{x}_3 &= \mathbf{x}_2 + \mathbf{Pr}_2 \,, \end{aligned}$$

where $\mathbf{r}_2 = \mathbf{b} - \mathbf{Ax}_2$ and $\mathbf{r}_1 = \mathbf{b} - \mathbf{Ax}_1$ are the corresponding residuals. This process is called *iterative refinement*. Note that

$$\begin{aligned} \mathbf{x}_1 &= \mathbf{x}_0 + \mathbf{P}(\mathbf{b} - \mathbf{Ax}_0) = \mathbf{x}_0 + \mathbf{Pb} - \mathbf{PAx}_0 = \mathbf{x}_0 + \mathbf{x}_0 - \mathbf{PAx}_0 \\ &= \mathbf{x}_0 + (\mathbf{I} - \mathbf{PA})\mathbf{x}_0 = \mathbf{x}_0 + \mathbf{Sx}_0 \,, \end{aligned}$$

since $\mathbf{PA} = \mathbf{I} - \mathbf{S}$ in our earlier notation. Similarly, one obtains

$$\begin{aligned} \mathbf{x}_2 &= \mathbf{x}_1 + \mathbf{P}(\mathbf{b} - \mathbf{Ax}_1) = \mathbf{x}_0 + \mathbf{Sx}_0 + \mathbf{Pb} - \mathbf{PA}(\mathbf{x}_0 + \mathbf{Sx}_0) \\ &= \mathbf{x}_0 + \mathbf{Sx}_0 + \mathbf{x}_0 - (\mathbf{I} - \mathbf{S})(\mathbf{x}_0 + \mathbf{Sx}_0) \\ &= \mathbf{x}_0 + \mathbf{Sx}_0 + \mathbf{S}^2\mathbf{x}_0 \,, \end{aligned}$$

which is mathematically equivalent to what we had before and converges under the same conditions.

The matrix $\mathbf{P}$, our approximate inverse, is called a *preconditioner* (and its inverse is usually denoted $\mathbf{M}$). Probably the most important part of any iterative method is choosing the right preconditioner. For solving $\mathbf{Ax} = \mathbf{b}$, we need for $\mathbf{P}$ to allow fast evaluation of products $\mathbf{Pv}$ and simultaneously be close to $\mathbf{A}^{-1}$. Unfortunately, these goals are often in opposition. It is useful in practice to use even quite crude approximations to $\mathbf{A}^{-1}$ as preconditioners, though.

Let us illustrate the usefulness of this method. Suppose we want to solve $\mathbf{Ax} = \mathbf{b}$ and, moreover, suppose $\mathbf{A} = \mathbf{F}_n(\mathbf{I} + \mathbf{S})$, where

$$\mathbf{F}_n = \begin{bmatrix} 2+\sqrt{3} & 1 & & & \\ 1 & 4 & 1 & & \\ & 1 & 4 & 1 & \\ & & \ddots & \ddots & \ddots \end{bmatrix}$$

is $n \times n$ and $\mathbf{S}$, off its diagonal, is small [we will allow $s_{11} = (4-(2+\sqrt{3}))/(2+\sqrt{3})$ to be sort of big]. Then, let $\mathbf{P} = \mathbf{F}_n^{-1}$, although because $\mathbf{F}_n^{-1}$ is full, we never compute it. Instead, we note that by symmetric factoring, we have $\mathbf{F}_n = \mathbf{L}_n \mathbf{D} \mathbf{L}_n^T$, where

$$\mathbf{L}_n = \begin{bmatrix} 1 & & & \\ \alpha & 1 & & \\ & \alpha & 1 & \\ & & \ddots & \ddots \end{bmatrix}$$

and $\mathbf{D} = \text{diag}(2+\sqrt{3}, 2+\sqrt{3}, \ldots, 2+\sqrt{3})$. Note that we won't compute $\mathbf{S}$, either. Instead, we solve the sequence of equations

$$\begin{aligned} \mathbf{L}_n \mathbf{z}_0 &= \mathbf{b} \\ \mathbf{D}_n \mathbf{y}_0 &= \mathbf{z}_0 \\ \mathbf{L}_n^T \mathbf{x}_0 &= \mathbf{y}_0 \end{aligned}$$

in $O(n)$ flops to get $\mathbf{x}_0$, by means of which we will use iterative refinement to get an accurate value of $\mathbf{x}$ as shown below:

for $k = 1, 2, \ldots$ **do**
  Compute $\mathbf{r}_{k-1} = \mathbf{b} - \mathbf{A}\mathbf{x}_{k-1}$
  % Now, we compute $\mathbf{x}_k - \mathbf{x}_{k-1} = \mathbf{P}\mathbf{r}_{k-1}$
  Solve $\mathbf{L}\mathbf{z}_k = \mathbf{r}_{k-1}$
  Solve $\mathbf{D}\mathbf{y}_k = \mathbf{z}_k$
  Solve $\mathbf{L}^T \Delta \mathbf{x}_k = \mathbf{y}_k$
  Let $\mathbf{x}_k = \mathbf{x}_{k-1} + \Delta \mathbf{x}_k$
end for

This is an iterative refinement formulation of the iteration. Because $\|\mathbf{S}\| \doteq 0.01$, 10 or so iterations of this process gets $\mathbf{x}$ accurate to most significant digits; and each

iteration costs $O(n)$ flops. Thus, in $O(n)$ flops, we have solved our system. This is significantly better than the $O(n^3)$ cost for full matrices!

Note that $\mathbf{A}$ need not really be tridiagonal: It can have a few more entries here and there off the main diagonals, contributing to $\mathbf{S}$, if they're not too large. Even if there are lots of them, the cost of computing the residual is at most $O(n^2)$ per iteration, and if $\mathbf{S}$ is small, we will need only $O(1)$ iterations.

It's hard to overemphasize the importance of this seemingly trivial change from direct, algorithmic finite-number-of-steps solution to a convergent iteration, but most large systems are, in practice, solved with such methods. As Greenbaum notes,

> With a sufficiently good preconditioner, each of these iterative methods can be expected to find a good approximate solution quickly. In fact, with a sufficiently good preconditioner $\mathbf{M}$, an even simpler iteration method such as $\mathbf{x}_k = \mathbf{x}_{k-1} + \mathbf{M}^{-1}(\mathbf{b} - \mathbf{A}\mathbf{x}_{k-1})$ may converge in just a few iterations, and this avoids the cost of inner products and other things in the more sophisticated Krylov space methods (in Hogben 2006, p. 41–10)

(which highlights the importance of choosing $\mathbf{P}$ well). The iterative methods included in MATLAB are (for $\mathbf{A}\mathbf{x} = \mathbf{b}$)

- `bicg`—biconjugate gradient
- `bicgstab`—biconjugate gradient stabilized
- `cgs`—conjugate gradient squared
- `gmres`—generalized minimum residual
- `lsqr`—least squares
- `minres`—minimum residual
- `pcg`—preconditioned conjugate gradient
- `qmr`—quasiminimal residual
- `symmlq`—symmetric LQ

but there is no explicit program for iterative refinement, because it is so simple. See, for example, Olshevsky (2003b) for pointers to the literature, or perhaps Hogben (2006).

It was Skeel who first noticed that a single pass of iterative refinement could be used to improve the structured backward error. He noticed that computing the residual in the same precision (not twice the precision, which might not be easily available) gives the exactly rounded residual for $(\mathbf{A} + \Delta\mathbf{A})\mathbf{x} = \mathbf{b} + \mathbf{r}$ for some $|\Delta\mathbf{A}| \leq O(\mu_M)|\mathbf{A}|$. That is, the computed residual is the exact residual for only $O(\mu_M)$ relative backward errors in $\mathbf{A}$, preserving structure. Notice that the computed solution $\mathbf{x}$ usually comes only with a *normwise* backward error guarantee: It is the correct solution to $(\mathbf{A} + \Delta\mathbf{A})\mathbf{x} = \mathbf{b} + \Delta\mathbf{b}$ with $\|\Delta\mathbf{A}\| = O(\mu_M\|\mathbf{A}\|)$ and $\|\Delta\mathbf{b}\| = O(\mu_M\|\mathbf{b}\|)$, which does not preserve structure. A single pass of iterative refinement can, if the condition number of $\mathbf{A}$ is not too large, improve this situation considerably. Let $\mathbf{x}_1 = \mathbf{x} + \Delta\mathbf{x}$, where

$$\mathbf{A}(\Delta\mathbf{x}) = \mathbf{r}\,.$$

Then solving this system gives us, more nearly, a solution of the same sort of problem.

The following argument, though not "tight," gives some idea of why this is so. Suppose we have approximately solved $\mathbf{A}\mathbf{x} = \mathbf{b}$ and found a computed solution, which we will call $\mathbf{x}_0$. Then, on computing the residual $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ in single precision, we know that we have found the exact solution of

$$(\mathbf{A} + \Delta\mathbf{A}_0)\,\mathbf{x}_0 = \mathbf{b} - \mathbf{r}_0\,,$$

where $|\Delta\mathbf{A}_0| \le c\mu_M|\mathbf{A}|$ and c is a small constant that depends linearly on the dimension n. Notice that the $\Delta\mathbf{A}_0$ is componentwise small. The working-precision residual $\mathbf{r}_0$ is included (it might not be very small), and what this statement says is merely that we have an accurate residual for a closely perturbed system. How small is $\mathbf{r}_0$? It is easy to see that, normwise,

$$\begin{aligned}\|\mathbf{r}_0\| &\doteq \rho\|\mathbf{A}\|\,\|\mathbf{A}^{-1}\|\|\mathbf{b}\|\mu_M \\ &\doteq \rho\,\kappa(\mathbf{A})\|\mathbf{b}\|\mu_M\,,\end{aligned} \tag{7.1}$$

at most (being sloppy with constants, though). ρ is called a growth factor. Now we suppose that in solving $\mathbf{A}\Delta\mathbf{x} = \mathbf{r}_0$ in the same approximate fashion (call the solution $\Delta\mathbf{x}_0$), we get the same approximate growth, so that the residual in *this* equation can be written

$$(\mathbf{A} + \Delta\mathbf{A}_1)\,\Delta\mathbf{x} = \mathbf{r}_0 - \mathbf{s}_0\,,$$

where again the perturbation $\Delta\mathbf{A}_1$ is small componentwise compared to $\mathbf{A}$, and $\mathbf{s}_0$ is the residual that we could compute using working precision in the update equation:

$$\mathbf{s}_0 = \mathbf{r}_0 - \mathbf{A}\Delta\mathbf{x}_0\,.$$

Our "similar growth" assumption says that $\|\mathbf{s}_0\| \doteq \rho\,\kappa(\mathbf{A})\|\mathbf{r}_0\|$. This will be, roughly speaking, $\rho^2\kappa(\mathbf{A})^2\|\mathbf{b}\|\mu_M^2$ and might, if we are lucky, be quite a bit smaller. Adding together the two equations, we find that

$$\begin{aligned}(\mathbf{A} + \Delta\mathbf{A}_0)\,(\mathbf{x}_0 + \Delta\mathbf{x}_0) &= \mathbf{b} + (\Delta\mathbf{A}_0 - \Delta\mathbf{A}_1)\,\Delta\mathbf{x}_0 - \mathbf{s}_0 \\ &= \mathbf{b} + O(\mu_M^2)\,,\end{aligned}$$

where we have suppressed the $\rho^2\kappa^2(\mathbf{A})$ and the dependence on $\kappa(\mathbf{A})$ from the other small term in the order symbol. This loose argument leads us to expect that a single pass *ought* to give us nearly the exact solution to a perturbed problem where the perturbation is componentwise small.

Of course, it takes more effort to establish in detail that it actually does so under many circumstances, and to describe exactly what those circumstances are. We can easily see in the above argument though that if the condition number of $\mathbf{A}$ or the growth factor ρ or both are "too large," there will be trouble. Full details of a much tighter argument are in Skeel (1980).

Example 7.1. This idea helps in coping with examples where the residual is unacceptably large. This can happen even with well-scaled matrices (in theory, though

as we have discussed it is almost unheard of in practice). Consider the family of matrices shaped like the following (we show the $n = 6$ case):

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 & 0 & 0 \\ -1 & -1 & 1 & 0 & 0 & 0 \\ -1 & -1 & -1 & 1 & 0 & 0 \\ -1 & -1 & -1 & -1 & 1 & 0 \\ -1 & -1 & -1 & -1 & -1 & 1 \end{bmatrix} . \tag{7.2}$$

This well-known example has a growth factor for Gaussian elimination with partial pivoting (although pivoting doesn't actually happen because it is arranged that the pivots are already in the right place) that is as bad as possible: The largest element in $\mathbf{U}$ where $\mathbf{A} = \mathbf{LU}$ is $2^n + 1$. The condition number of the matrix is quite reasonable, however; it is only 33 or so when $n = 32$. But the solution with GEPP is not acceptable, without iterative refinement, as we will see. As proved in Skeel (1980), a single pass of iterative refinement is enough to stabilize the algorithm in the strong sense discussed above.

Suppose we take $\mathbf{b}$ to be the vector $\mathbf{v}_n$ corresponding to the smallest singular value of $\mathbf{A}$. The choice of $\mathbf{b}$ doesn't really matter very much, though this choice is especially cruel. When we compute (for $n = 32$) the solution of $\mathbf{Ax} = \mathbf{b}$, we should get $\mathbf{u}_n$, the final vector of the $\mathbf{U}$ matrix from the SVD. Call our computed solution $\mathbf{x}_0$. We compute the residual $\mathbf{r}_0 = \mathbf{b} - \mathbf{Ax}_0$, using the same 15-digit precision used to compute $\mathbf{x}_0$. The norm of $\mathbf{r}_0$ is about 10^{-9}, and thus the nearest matrix $\mathbf{A} + \Delta\mathbf{A}$ for which $\mathbf{x}_0$ really solves the problem is about the same distance away, componentwise. If we now solve $\mathbf{A}\Delta\mathbf{x} = \mathbf{r}$ and put $\mathbf{x}_1 = \mathbf{x}_0 + \Delta\mathbf{x}$, then when we compute the residual again, we find that $\|\mathbf{r}_1\|_\infty$ is about 10^{-17}. This produces an entirely satisfactory backward error.

For $n = 64$, the situation is much worse, at the beginning. The zeroth solution has a residual with infinity norm nearly 1; that is, almost no figures in the solution are correct. A single pass of iterative refinement gives $\mathbf{x}_1$ with $\|\mathbf{r}_1\|_\infty \doteq 1.22 \cdot 10^{-13}$, 13 orders of magnitude better. The 2-norm condition number of the matrix is only about 56.8, mind, and the ∞-norm condition number is 128. The Skeel condition number (see Eq. (6.9)) $\mathrm{cond}(\mathbf{A}) = \||\mathbf{A}^{-1}||\mathbf{A}|\|_\infty$ is not very different, being very close to 66. However, the structured condition number *for this* $\mathbf{x}$ is quite a bit smaller:

$$\mathrm{cond}(\mathbf{A}, \mathbf{x}) = \frac{\||\mathbf{A}^{-1}|\,|\mathbf{A}|\,|\mathbf{x}|\,\|_\infty}{\|\mathbf{x}\|_\infty} \doteq 5.548 .$$

Thus, for $n = 64$, we can expect nearly 13 figures of accuracy in $\mathbf{x}_1$, because the residual is so small. ◁

Remark 7.1. We should point out that $|\mathbf{A}|$ does not commute with $|\mathbf{A}^{-1}|$ in general, and in particular does not commute for this example. The Skeel condition number uses the inverse first. ◁

7.2 What Could Go Wrong with an Iterative Method?

Let us now return to the iterative idea itself, and no longer think about the effects of just one pass, but rather now think about what happens if many iterations are needed. Indeed, thousands of iterations are common in some applications. The basic theoretical question is now: when does $\mathbf{S}^k \to 0$, and how fast does it do so? A theorem of eigenvalues, $\mathbf{S}^k \to 0$ if all eigenvalues have $|\lambda| \leq \rho < 1$, seems to characterize things completely. However, as we saw in Sect. 5.5.2, pseudospectra turn out to play a role for nonnormal $\mathbf{S}$. There are other methods to look at this problem, and there is an extensive discussion in Higham (2002, chapter 18). We content ourselves here with an example.

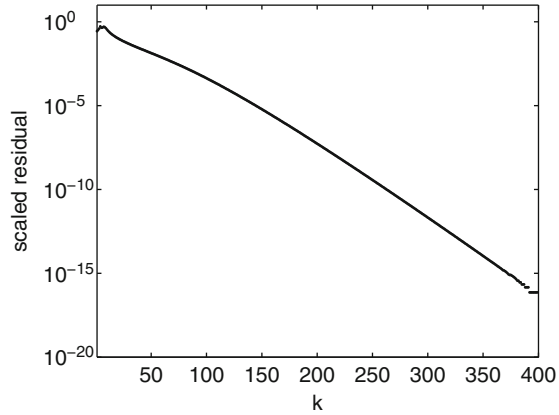

Fig. 7.1 Scaled residuals for the Richardson iteration solution of a nonnormal matrix with $n = 5$. We see fairly monotonic convergence

Example 7.2. Suppose that $\mathbf{A} = \mathbf{I} - \mathbf{S}$, where $\mathbf{S}$ is bidiagonal, with all diagonal entries equal to $8/9$ and all entries of the first superdiagonal equal to -1. This is similar to the example matrix that was used in Sect. 5.5.2. Now, we wish to solve $\mathbf{Ax} = \mathbf{b}$, where, say, $\mathbf{b}$ has all entries equal to 1. Because all eigenvalues of $\mathbf{S}$ are less than 1 in magnitude, we know that the series $\mathbf{I} + \mathbf{S} + \mathbf{S}^2 + \cdots$ converges. Moreover, we know that ultimately the error goes to zero like "some constant" times $(8/9)^k$, and that $k = 400$ gives $(8/9)^{400} \doteq 1 \times 10^{-21}$. Therefore, the Richardson iteration

$$\mathbf{x}_{k+1} = \mathbf{b} + \mathbf{S}\mathbf{x}_k$$

should converge to the reference solution. Incidentally, the reference solution has $x_n = 9$, $x_j = O((9/8)^{n-j})$ for $j = n-1, \ldots, 1$ by back substitution. This exponential growth in the solution suggests that we should evaluate the quality of our solution by examining the *scaled* residual,

$$\delta = \frac{\|\mathbf{b} - \mathbf{Ax}\|}{\|\mathbf{A}\|\|\mathbf{x}\|} .$$

We will use the kth iterate to scale the residual of the kth solution in the figures below.

Because the pseudospectrum of this matrix (when the dimension is large) pokes out into the region $|\lambda| > 1$—that is, the pseudospectral radius ρ_ε of Eq. (5.13) is larger than 1—we expect that this iteration will encounter trouble for large dimensions. In other words, the "constant" that we hid under the blanket called "some constant" in the previous discussion actually grows exponentially with the dimension n. While it is constant for any given iteration, the size of the constant gets ridiculously large. In Problem 7.5, you are asked to give an explicit lower bound, confirming this. Thus, as might be expected, the iteration works quite well for a 5×5 matrix, as shown in Fig. 7.1. Also, as predicted, our expectation of trouble is confirmed by an 89×89 matrix, as shown in Fig. 7.2. ◁

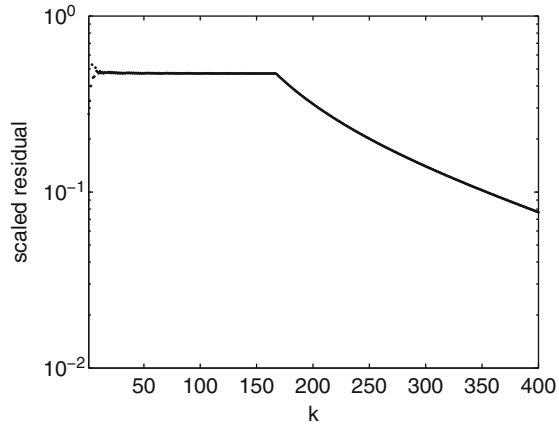

Fig. 7.2 Scaled residuals for the Richardson iteration solution of a nonnormal matrix of dimension 89×89. Convergence is very slow, which would be unexpected if we were not aware of the pseudospectra of the matrix $\mathbf{S}$

7.3 Some Classical Variations

In this section, we look at a few variations of the iterative method we have discussed thus far, namely, Jacobi iteration, Gauss–Seidel iteration, and successive overrelaxation (SOR).

Let us begin with *Jacobi iteration*. Take $\mathbf{P} = \mathbf{D}^{-1}$, the inverse of the diagonal part of the matrix (so, write the matrix as $\mathbf{D} + \mathbf{E}$). Then, mathematically, $\mathbf{PA} = \mathbf{D}^{-1}\mathbf{A}$ and $\mathbf{S} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{A}$ is pretty simple, but unless the off-diagonal elements of $\mathbf{A}$ are small compared to $\mathbf{D}$, this won't converge: $\mathbf{I} - \mathbf{D}^{-1}\mathbf{A}$ has *only* off-diagonal elements, $-a_{ij}/a_{ii}$, and we want (ideally) $\|\mathbf{S}\| < 1$. As an iteration to solve $\mathbf{Ax} = \mathbf{b}$, we proceed as follows. $\mathbf{Ax} = \mathbf{b}$ is equivalent to $(\mathbf{D} + \mathbf{E})\mathbf{x} = \mathbf{b}$. Therefore,

$$\begin{aligned}\mathbf{D}\mathbf{x} &= \mathbf{b} - \mathbf{E}\mathbf{x}\\ \mathbf{x}_{n+1} &= \mathbf{D}^{-1}(\mathbf{b} - \mathbf{E}\mathbf{x}_n)\\ &= \mathbf{x}_n - \mathbf{x}_n + \mathbf{D}^{-1}(\mathbf{b} - \mathbf{E}\mathbf{x}_n)\\ &= \mathbf{x}_n + \mathbf{D}^{-1}(\mathbf{b} - \mathbf{D}\mathbf{x}_n - \mathbf{E}\mathbf{x}_n)\\ &= \mathbf{x}_n + \mathbf{D}^{-1}(\mathbf{b} - \mathbf{A}\mathbf{x}_n)\,,\end{aligned}$$

which is the Jabobi iteration.

The Gauss–Seidel method is also worth considering. As Strang (1986, p. 406) said, "[T]his is called the Gauss–Seidel method, even though Gauss didn't know about it and Seidel didn't recommend it. Nevertheless it is a good method." Take $\mathbf{P} = \mathbf{L}^{-1}$, where $\mathbf{L}$ is the lower-triangular part of $\mathbf{A}$, including the diagonal:

$$\mathbf{L} = \begin{bmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ \vdots & & \ddots & \\ a_{n1} & a_{n2} & \cdots & a_{nn}. \end{bmatrix}$$

The iteration demands, for $\mathbf{A} = \mathbf{L} + \mathbf{U}$, that we solve

$$\mathbf{L}\mathbf{x}_{k+1} = \mathbf{b} - \mathbf{U}\mathbf{x}_k$$

for $\mathbf{x}_{k+1}$ or, alternatively, that use the map

$$\mathbf{x}_{k+1} = \mathbf{L}^{-1}\mathbf{b} - \mathbf{L}^{-1}\mathbf{U}\mathbf{x}_k$$

(at least in theory—in practice, we can write this as a simple iteration, reusing the same vector $\mathbf{x}$ as we go so; it uses less storage than Jacobi iteration). Because $\mathbf{L}$ is a better approximation to $\mathbf{A}$, this often converges twice as fast as Jacobi. This is usually win–win, although Jacobi iteration can in some cases win by use of parallelism.

But there is a dramatically better method using only trivially more effort, *successive overrelaxation* (SOR). Split $\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U}$, with $\mathbf{L}$ now being strictly lower-triangular. We get, with an "overrelaxation parameter" $\omega \in (0,2)$,

$$(\mathbf{D} + \omega\mathbf{L})\mathbf{x} = \omega\mathbf{b} - (\omega\mathbf{U} - (\omega - 1)\mathbf{D})\mathbf{x}$$

from the following:

$$\begin{aligned}\mathbf{A}\mathbf{x} &= \mathbf{b}\\ \omega\mathbf{A}\mathbf{x} &= \omega\mathbf{b}\\ \mathbf{D}\mathbf{x} + \omega\mathbf{A}\mathbf{x} &= \omega\mathbf{b} + \mathbf{D}\mathbf{x}\\ \mathbf{D}\mathbf{x} + \omega(\mathbf{L} + \mathbf{D} + \mathbf{U})\mathbf{x} &= \omega\mathbf{b} + \mathbf{D}\mathbf{x}\\ (\mathbf{D} + \omega\mathbf{L})\mathbf{x} &= \omega\mathbf{b} + \mathbf{D}\mathbf{x} - \omega\mathbf{D}\mathbf{x} - \omega\mathbf{U}\mathbf{x}\\ &= \omega\mathbf{b} - (\omega\mathbf{U} - (\omega - 1)\mathbf{D})\mathbf{x}.\end{aligned}$$

Here, $\mathbf{P} = \omega(\mathbf{L} + \omega\mathbf{D})^{-1}$ and we have a free parameter ω, the relaxation parameter, to choose. We may choose it differently for every iteration, to try to minimize the maximum eigenvalue of what we have been calling $\mathbf{S}$. As information is extracted from the solution estimating the largest Jacobi iteration matrix eigenvalue, we may improve our choice. Here $\mathbf{S} = (\mathbf{L} + \omega\mathbf{D})^{-1}((\omega - 1)\mathbf{D} - \omega\mathbf{U})$, and for some finite-difference applications the optimal ω is known. For the right choice of ω, this can seriously outperform Gauss–Seidel.

Example 7.3. We use `A = delsq( numgrid( 'B', n ) )` as an example for SOR, even though direct methods are actually better for this nearly banded matrix. We look first at small-dimension matrices, specifically for $n = 5$, 8, 13, 21, and 34. The dimension of $\mathbf{A}$ is $O(n^2) \times O(n^2)$. By fitting the data from these smaller matrices, the largest eigenvalue of the Jacobi iteration matrix $\mathbf{D}^{-1}(\mathbf{A} - \mathbf{D})$ seems to be $\mu = 1 - {}^{16.65}/_{n^2}$, which means that the optimal $\omega = 2/(1 + \sqrt{1 - \mu^2})$ is about $2/(1 + {}^{16.65}/_{n})$, and the eigenvalues of the SOR error matrix are then less than $(1 - {}^{16.65}/_{n})/(1 + {}^{16.65}/_{n})$, approximately.

When we use 150 iterations of SOR to solve the system for $n = 80$ (so the matrix is 4808×4808), we find that the residual behaves on the kth iteration as approximately $10^3 \times (\omega - 1)^k$, and after 150 iterations, the residual is 4.5×10^{-7}. In contrast, the same number of Jacobi iterations cannot be expected even to give one figure of accuracy, and Gauss–Seidel is not much better. The difference between $(1 - O({}^1/_n))^k$ and $(1 - O({}^1/_{n^2}))^k$ is huge. The constant 10^3 above changes, of course, with the dimension n. It seems experimentally to vary as $(n^2)^2$ or the square of the dimension of $\mathbf{A}$, which, though growing with n, is at least not growing *exponentially* with n. ◁

Remark 7.2. These classical methods are still useful in some circumstances, but there have been serious advances in iterative methods since these were invented. Multigrid methods and conjugate gradient methods seem to be the methods of choice. See Hogben (2006, chapter 41), by Anne Greenbaum, for an entry point to the literature. ◁

7.4 Large Eigenvalue Problems

All methods for finding eigenvalues are iterative[1]; so, unlike the case where we were solving $\mathbf{Ax} = \mathbf{b}$, where there was a distinction between finite, terminating "direct" methods (such as QR factoring or LU factoring) and nonterminating "iterative" methods such as SOR, when we tackle $\mathbf{Ax} = \lambda\mathbf{x}$, the distinction in algorithm classes is a bit fuzzy and depends chiefly on how large a "large matrix" is today. On a tablet PC in 2010, not a high-end machine by any means, it took MATLAB five seconds to compute all $1,000$ eigenvalues and eigenvectors of a random $1,000 \times 1,000$ matrix, as follows:

[1] Yes, even for $n = 2$, because while square roots are "legal," they are not finite—extracting them is iterative, too.

```
%% Eigenvalues of a 1000 by 1000 Random Matrix
A = rand( 1000 );
e = eig( A );
plot( real(e), imag(e), 'k.' )
axis('square'), axis([-10,10,-10,10]),set(gca,'Fontsize',16)
xlabel('Real Part'),ylabel('Imaginary Part')
```

So today a $1,000 \times 1,000$ matrix is not large, even though it and its matrix of eigenvectors have a million entries each. See Fig. 7.3.

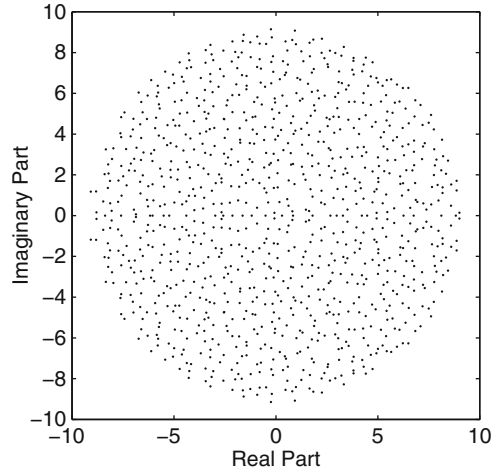

Fig. 7.3 Nine hundred ninety-nine eigenvalues of a random $1,000 \times 1,000$ real matrix. The odd eigenvalue is about 500.3294 (because all entries of this matrix are positive, the Perron–Frobenius theorem applies, and thus there is a unique eigenvalue with largest magnitude, which is real). Note the conjugate symmetry, and the confinement to a disk with radius about 10

For many applications, however, we might not need all $1,000$ eigenvalues and eigenvectors, but perhaps just the six largest, or six smallest. Consider the following situation. Suppose we execute

```
a=rand(1000);
eigs(a)
```

in MATLAB and receive the following warning:

```
Warning: Only 5 of the 6 requested eigenvalues converged.
In eigs>processEUPDinfo at 1474
In eigs at 367
```

This command had some sort of iteration failure—it only found five of the six largest eigenvalues. We will see in a moment a possible way to work around this failure. But before, notice that if we execute

```
eigs(a,6,0)
```

we successfully and quickly find the six smallest eigenvalues. Note that `eigs` is not `eig`. The "s" is for "sparse," although it works (as in this case) on a dense matrix. The following simple kludge avoids the convergence failure in this example:

```
eigs( a - 10.032*speye(1000) )
ans + 10.032
```

That is, we simply shifted the matrix a random amount, and this was enough to kick the iteration over its difficulties. Then we correctly find the eigenvalues:

$$10^2 \begin{bmatrix} 5.0033 \\ -0.0908 - 0.0118i \\ -0.0908 + 0.0118i \\ -0.0882 + 0.0119i \\ -0.0882 - 0.0119i \\ -0.0880 + 0.0016i \end{bmatrix}.$$

This is, of course, not entirely satisfactory, but we shall pursue this in a bit of detail shortly.

For large sparse matrices, special methods of iterating are needed: The construction of an upper Hessenberg intermediate matrix is already too expensive, so the QR iteration (as is) is also too expensive. The techniques of choice are Arnoldi iteration (as implemented in ARPACK and in MATLAB's `eigs` routine) and other special-purpose routines, such as Rayleigh quotient iteration for the symmetric eigenproblem. Before moving on to this method, we consider the so-called Krylov subspaces

$$\begin{bmatrix} \mathbf{v} & \mathbf{A}\mathbf{v} & \mathbf{A}^2\mathbf{v} & \mathbf{A}^3\mathbf{v} & \dots & \mathbf{A}^k\mathbf{v} \end{bmatrix},$$

which can be generated using only k matrix–vector multiplications. The power method considered only the latest $\mathbf{A}^k\mathbf{v}$ (and perhaps the previous). In exact arithmetic, as noted before, the characteristic polynomial can be constructed from the finite sequence $[\mathbf{v}, \mathbf{A}\mathbf{v}, \dots, \mathbf{A}^n\mathbf{v}]$ because these vectors must be linearly dependent; but in the presence of rounding errors, we are much better off using other techniques; if we're at all lucky, we will get good eigenvalue information with k iterations for $k \ll n$.

Rayleigh quotient iteration—or RQI—is easily described (see Problem 6.16). Given an initial guess for an eigenvector $\mathbf{x}_0$, form

$$\mu = \frac{\mathbf{x}_0^H \mathbf{A}\mathbf{x}_0}{\mathbf{x}_0^H \mathbf{x}_0},$$

the Rayleigh quotient. We make the crucial simplification of assuming $\mathbf{A} \in \mathbb{R}^{n\times n}$ and $\mathbf{A}^H = \mathbf{A}$; that is, $\mathbf{A}$ is symmetric. More, let $\mathbf{A}$ be positive-definite, and sparse (or at least fast to make matrix–vector products $\mathbf{y} = \mathbf{A}\mathbf{v}$ with). Finally, we suppose eigenvalues are simple. Once we have μ, which is the best least-squares approximation to an eigenvalue corresponding to $\mathbf{x}_0$, we now use it to improve $\mathbf{x}_0$. Solve

$$(\mathbf{A} - \mu\mathbf{I})\mathbf{z} = \mathbf{x}_0, \tag{7.3}$$

and put $\mathbf{x}_1 = \mathbf{z}/\|\mathbf{z}\|$. You may use any convenient method to solve Eq. (7.3); since $\mathbf{A}$ is sparse (or $\mathbf{Av}$ is easy), you may choose a sparse iterative method. You may choose not to solve it very accurately; after all, $\mathbf{x}_1$ will just be another approximate eigenvector, and we're going to do the iteration again. When do we stop? If

$$\|\mathbf{A}\mathbf{x}_i - \mu_i \mathbf{x}_i\| < \varepsilon\,,$$

then we know that μ_i is an exact eigenvalue for $\mathbf{A} + \Delta\mathbf{A}$ with $\|\Delta\mathbf{A}\| \leq \varepsilon\|\mathbf{A}\|$. Hence, this is a reliable test for convergence, from a backward error point of view. Since symmetric matrices have perfectly conditioned eigenvalues (normwise), this may be satisfactory from the forward point of view, too. Thus, we get Algorithm 7.1.

Algorithm 7.1 Rayleigh quotient iteration

Require: A vector $\mathbf{x}_0$, a method to compute $\mathbf{y} = \mathbf{Av}$, a method to solve $(\mathbf{A} - \mu\mathbf{I})\mathbf{z} = \mathbf{b}$
  for $i = 1, 2, \ldots$ until converged **do**
    $\mu_{i-1} = \mathbf{x}_{i-1}^T(\mathbf{A}\mathbf{x}_{i-1})/(\mathbf{x}_{i-1}^T\mathbf{x}_{i-1})$
    Solve $(\mathbf{A} - \mu_{i-1}\mathbf{I})\mathbf{z} = \mathbf{x}_{i-1}$
    $\mathbf{x}_i = \mathbf{z}/\|\mathbf{z}\|$
  end for

We may want to find generalizations of this method; for example, we wish to find more than one eigenvector at a time. Suppose $\mathbf{x}_0 \in \mathbb{R}^{n\times k}$ ($k \ll n$). Then if $\mathbf{x}_0^T\mathbf{x}_0 = \mathbf{I}$,

$$\mathbf{H} = \mathbf{x}_0^T\mathbf{A}\mathbf{x}_0 \in \mathbb{R}^{k\times k}$$

shares some interesting features with the 1×1 case. The eigenvalues of $\mathbf{H}$, called Ritz values, are approximations to eigenvalues of $\mathbf{A}$, in some sense. Alternatively, one can think of the following iteration:

for $i = 1, 2, \ldots$ until converged **do**
  $\mathbf{H} = \mathbf{x}_{i-1}^T\mathbf{A}\mathbf{x}_{i-1}$
  $\mu = \mathrm{diag}(\mathbf{H})$
  for $j = 1, 2, \ldots, k$ **do**
    Solve $(\mathbf{A} - \mu_{jj}\mathbf{I})\,\mathbf{z}_j = (\mathbf{x}_{i-1})_j$
    $(\mathbf{x}_i)_j = \mathbf{z}_j$
  end for
  $(\mathbf{X}_j, \mathbf{R}) = \mathtt{qr}(\mathbf{X}_j)$
end for

This essentially does k independent Rayleigh iterations at once; the `qr` step just makes sure the eigenvalues are kept separate.

We might also wish to solve unsymmetric problems. The difficulties here are worse, as we must solve for left eigenvectors, too; this is called broken iteration, or Ostrowski iteration for some variations. In the symmetric case, convergence is often cubic; for the nonsymmetric case, this is true only sometimes. More seriously, if all we can do with $\mathbf{A}$ is make $\mathbf{Av}$, how do we make $\mathbf{y}^H\mathbf{A}$? This can be done without

constructing $\mathbf{A}$ explicitly [which costs $O(n^2)$], but it can be awkward.[2] Still, we have a method:

Require: For $\mathbf{x}_0, \mathbf{y}_0 \in \mathbb{C}^n$, a way to compute $\mathbf{Av}$ and a way to solve both $(\mathbf{A} - \mu\mathbf{I})\mathbf{z} = \mathbf{x}$ and $\left(\mathbf{A}^H - \overline{\mu}\mathbf{I}\right)\mathbf{w}^H = \mathbf{y}^H$

for $i = 1, 2, \ldots$ until converged **do**

$\mu_{i-1} = (\mathbf{y}_{i-1}^H \mathbf{A}\mathbf{x}_{i-1})/(\mathbf{y}_{i-1}^H \mathbf{x}_{i-1})$ (N.B. fails if $\mathbf{y}_{i-1}^H \mathbf{x}_{i-1}$ is too small)

Solve $(\mathbf{A} - \mu_{i-1}\mathbf{I})\mathbf{z} = \mathbf{x}_{i-1}$

$\mathbf{x}_i = \mathbf{z}/\|\mathbf{z}\|$

Solve $(\mathbf{A}^H - \overline{\mu}\mathbf{I})\mathbf{w} = \mathbf{y}_{i-1}$

$\mathbf{y}_i = \mathbf{w}/\|\mathbf{w}\|$

end for

Convergence in residual happens if

$$\|\mathbf{A}\mathbf{x}_i - \mu_i \mathbf{x}_i\| \leq \varepsilon$$

as before, but note that now the eigenvalue may be very ill-conditioned, in which case $\mu_i \in \Lambda_\varepsilon(\mathbf{A})$ does not mean $|\lambda - \mu_i| = O(\varepsilon)$ for a modest multiple of ε.[3]

Again, when to *stop* the iteration? Since the residuals are being computed at each stage, one can in principle stop if the residuals get small enough that the backward error interpretation of $\mathbf{r}$, namely, that we have solved $\mathbf{Ax} = \mathbf{b} - \mathbf{r}$, suggests that the residual is negligible. However, rounding errors (especially if the matrix $\mathbf{S}$ is not normal) can prevent the residuals from getting as small as we like.[4]

Example 7.4. The popular Jenkins–Traub method (Jenkins and Traub 1970) for finding roots of polynomials expressed in the monomial basis has at its core an iteration related to the Rayleigh quotient iteration on the companion matrix for the polynomial. In this example, we use RQI on the companion matrix of a polynomial to find some of its roots, as follows. Recall that a companion matrix for a monic polynomial $p(z) = a_0 + a_1 z + \cdots + z^n$ can be written as a sparse matrix, all zero except for the first subdiagonal, which is just 1s, and the final column, which is the negative of the polynomial coefficients. It is a short exercise to see that if z is a root of $p(z)$, then the vector $[1, z, z^2, \ldots, z^{n-1}]$ is a left eigenvector of $\mathbf{C}$, and a corresponding right eigenvector is $[\alpha_1(z), \alpha_2(z), \ldots, \alpha_n(z)]$, where $\alpha_n(z) = 1$, $\alpha_{n-1}(z) = a_{n-1} + z$, $\alpha_{n-2}(z) = a_{n-2} + z(a_{n-1} + z)$, and so on up until $\alpha_1(z) = a_1 + z(a_2 + z(a_3 + \cdots)$, which must also equal $-a_0/z$ if $z \neq 0$ (and, of course, $a_0 = 0$ if $z = 0$). These are the successive evaluations of the polynomial that one gets by executing Horner's method. That is, for *this* kind of matrix, a guess at an eigenvalue λ will automatically give us a pair of approximate left and right eigenvectors. It is simple to form the Rayleigh quotient $(\mathbf{x}^H\mathbf{C}\mathbf{x})/(\mathbf{x}^H\mathbf{x})$ or the Ostrowski quotient $(\mathbf{y}^H\mathbf{C}\mathbf{x})/(\mathbf{y}^H\mathbf{x})$ from these to give us a hopefully improved estimate of the eigenvalue (which then can be

[2] See Bostan et al. (2003). For a history of the transposition principle, see http://cr.yp.to/transposition.html.

[3] Please consult Demmel (1997) or Hogben (2006) for more information on general techniques such as the implicitly restarted Arnoldi iteration.

[4] For more on this, see the discussion in Higham (2002).

fed back into the eigenvector formulae to use on the next iteration). This works, and it's faster than solving (which also works, and works more generally).

Consider Newton's example, $p(z) = z^3 - 2z - 5$. A companion matrix for this is

$$\mathbf{C} = \begin{bmatrix} 0 & 0 & 5 \\ 1 & 0 & 2 \\ 0 & 1 & 0 \end{bmatrix}.$$

If we start with an initial approximation $z_0 = -1 + i$ and use the formulae above for Ostrowski iteration, we get convergence in five iterations. If instead we *solve* for our approximate eigenvectors at each step via $\mathbf{C} - z^{(i)})\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)}$, and similarly for the left eigenvector, neither of which is hard because this matrix is sparse, then this is more like a normal Rayleigh quotient case where we don't know what the eigenvectors look like. In both cases the convergence appears to be quadratic, but the Rayleigh quotient only converges if solving for the new eigenvector happens each time. That is, with the formulae for the left and right eigenvectors instead of solving, only Ostrowski (also called "broken") iteration converges, but Rayleigh quotient iteration converges if the new eigenvectors are solved for.

Once a root has been found, it is necessary to *deflate* the matrix (or the polynomial); we do not discuss this in any detail here, although note that this is entirely possible within the framework of matrices—using either the left or right eigenvectors, one can in theory find a matrix one dimension smaller that has all the remaining roots as eigenvalues. Let

$$\mathbf{X} = \begin{bmatrix} \alpha_1 & 0 & 0 \\ \alpha_2 & 1 & 0 \\ \alpha_3 & 0 & 1 \end{bmatrix},$$

where the first column is the right eigenvector corresponding to the root z that we have found. Note that $\alpha_1 = {}^{-a_0}/_z$, which we assume is nonzero, so that $\mathbf{X}$ is invertible. Then $\mathbf{X}^{-1}\mathbf{C}\mathbf{X}$ has $[z,0,0]^T$ as its first column, and the remaining two eigenvalues of $\mathbf{C}$ are the two eigenvalues of the 2×2 block in the second two rows and columns. Similarly, one could deflate instead with the left eigenvector (which works even if $a_0 = 0$, though trivially since the matrix is already deflated in that case).

This is mathematically equivalent to synthetic division if the right eigenvector is used, and the deflated matrix is also a companion matrix; if the left eigenvector is used, then a different matrix is obtained. However, there is a tendency for rounding errors to accumulate in this process when one works with polynomials of high degree.

One can use a code such as this to implement this idea:

```
%% Rayleigh Quotient Iteration for a Companion Matrix
%
% Newton's example polynomial was $p(z) = z^3 - 2z - 5 = 0$.
%
C = [0 0 1.67608204095197550; 1 0 2; 0 1 -0.66478359180960489;];
x0 = -6 + 5i;
```

```
x = @(z) [-C(2,end)+z*(C(3,end)+z); C(3,end)+z; 1];
niters = 19;
xi  = zeros( niters, 1 );
xia = zeros( niters, 1 );
% Now solve at each step for new eigenvector.
xi(1) = x0;
xia(1)= x0;
x1 = x(x0); % Initial eigenvector
xa = x1;
x1 = x1/norm(x1,2);
for i=2:niters,
    x1 = (C-xi(i-1)*eye(3))\x1;
    x1 = x1 / norm(x1,2) ;
    xi(i) = (x1' * C * x1 ); %(x1'*x1) =1
    xia(i) = (xa' * C * xa )/(xa'*xa);
    xa = x(xi(i)); % analytic eigenvector formula
end
ers = xi(:) - xi(end);
close( figure(1) )
figure(1), semilogy( abs(ers), 'ko' ), set(gca,'fontsize',16),
    hold on
ersa = xia(:)-xi(end);
semilogy( abs( ersa ), 'kS' )
```

It is straightforward to adapt this code for other similar problems. ◁

Problems

7.1. Add an iterative refinement step to your solution of Problem 6.6. Note that evaluation of the residual is comparable in cost to the solution of the system, so this is a significantly costly step in this case. Does this help?

7.2. Consider the following system:

$$\begin{aligned} 2x_1 - x_2 &= 1 \\ -x_{j-1} + 2x_j - x_{j+1} &= j, \qquad j = 2, \ldots, n-1 \\ -x_{n-1} + 2x_2 &= n \end{aligned}$$

with $n = 100$. Parts 1–2 are from Moler (2004, prob. 2.19).

1. Use `diag` or `spdiags` to form the coefficient matrix and then use `lu`, \, and `tridisolve` to solve the system.
2. Use `condest` to estimate the condition of the coefficient matrix.
3. Solve the same problem as above, but changing 2 to be $\theta > 2$, say $\theta = 2.1$, and using the approach of `Seneca.m` to encode the matrix–vector product, use Jacobi iteration instead (note that $\mathbf{P}^{-1} = \theta \mathbf{I}$ and so $\mathbf{Px} = \frac{1}{\theta}\mathbf{x}$ is particularly easy). How large can the size of the problem be, before it takes MATLAB at least 60 s to solve the problem this way? How large can the problem be using a direct method?

(And, even more, the comparison is unfair; MATLAB's method is built-in, and Jacobi iteration must be "interpreted." Still, ...)

7.3. Implement in MATLAB the SOR method as described in the text. Be careful not to invert any matrices. Use your implementation with $\omega = 2 - O(1/n)$ to solve the linear system described in Problem 7.2 with $\theta = 2.1$.

7.4. Take $\mathbf{A} = \texttt{hilb}(8)$, the 8×8 Hilbert matrix. Use MGS to factor $\mathbf{A}$ approximately:

$$\mathbf{A} = \mathbf{QR}$$

with $\mathbf{Q}^T\mathbf{Q} \doteq \mathbf{I}$. In fact, $\mathbf{Q}^T\mathbf{Q} = \mathbf{I} + \mathbf{E}$, where $\|\mathbf{E}\| \leq \kappa(\mathbf{A}) \cdot c \cdot \mu_M$, where c is a modest constant and μ_M is the unit roundoff. Solve $\mathbf{Ax} = \mathbf{b}$ by using this $\mathbf{Q}$ and $\mathbf{R}$ in a factoring, as follows:

$$\begin{aligned} \mathbf{Qy} &= \mathbf{b} \\ \mathbf{Rx} &= \mathbf{y}\,, \end{aligned}$$

and use the solution process

$$\begin{aligned} \hat{\mathbf{y}} &= \mathbf{Q}^T\mathbf{b} \\ \hat{\mathbf{x}} &= \mathbf{R}\backslash\hat{\mathbf{y}}\,. \end{aligned}$$

Use one or two iterations of refinement to improve your solution. Discuss.

7.5. Consider the matrix from Example 7.2. Use the formula for the pseudospectral radius, namely, Eq. (5.13), and the estimate $\| ((\mathbf{A}) - z\mathbf{I})^{-1} \|_2 \geq |z - 8/9|^n$ [this is easy to see, because the $(n,1)$ entry of the resolvent is just that, and the 2-norm must be at least as large as any element of the matrix] to derive a reasonably tight lower bound on the maximum $\|\mathbf{S}^k\|_2$ when $n = 89$. Verify your bound by computation of $\mathbf{S}^k$ for $1 \leq k \leq 1600$. Hint: Take $\varepsilon = e/9^n$ and use $e^{1/n} > 1 + 1/n$. Ultimately, of course, $\|\mathbf{S}^k\|_2$ must go to zero as $k \to \infty$, but this analysis shows that it gets quite large along the way. This is why Richardson iteration is so slow for the system $(\mathbf{I} - \mathbf{S})\,\mathbf{x} = \mathbf{b}$.

7.6. The diagonal dominance of the matrix

$$\mathbf{A} = \begin{bmatrix} -10 & 1 & & & & \\ 1 & -10 & 1 & & & \\ & 1 & -10 & 1 & & \\ & & 1 & -10 & 1 & \\ & & & 1 & -10 & 1 \\ & & & & 1 & -10 \end{bmatrix}$$

tempts us to try Jacobi iteration $x_{n+1} = x_n + \mathbf{D}^{-1}(\mathbf{b} - \mathbf{A}x_n)$.

1. For $\mathbf{b} = [1, 1, 1, 1, 1, 1]^T$ and an initial guess of $x_0 = -[1, 1, 1, 1, 1, 1]^T/10$, carry out two iterations by hand. (The arithmetic for this problem is not out of reach:

The numbers were chosen to be nice enough to do on a midterm exam.) Can you estimate how accurate your final answer is?

2. Using symmetry and the eigenvalue formula for tridiagonal Toeplitz matrices $\lambda_k = -10 + 2\cos(\pi k/(n+1))$ (here $n = 6$), estimate the 2-norm condition number. The Skeel condition number $\text{cond}(\mathbf{A}) = \| |\mathbf{A}^{-1}||\mathbf{A}| \|$ can be shown to have exactly the same value. Using the phrases "structured condition number" and "structured backward error" in a sentence, explain what this means.