2022-03-14

1 Direct to Iterative

For the first part of the semester, we discussed *direct* methods for solving linear systems and least squares problems. These methods typically involve a factorization, such as LU or QR, that reduces the problem to a triangular solve using forward or backward substitution. These methods run to completion in a fixed amount of time, and are backed by reliable software in packages like LAPACK or UMFPACK.

There are a few things you need to know to be an informed *user* (not developer) of direct methods:

- You need some facility with matrix algebra, so that you know how to manipulate matrix factorizations and "push parens" in order to compute efficiently.
- You need to understand the complexity of different factorizations, and a little about how to take advantage of common matrix structures (e.g. low-rank structure, symmetry, orthogonality, or sparsity) in order to effectively choose between factorizations and algorithms.
- You need to understand a little about conditioning and the relationship between forward and backward error. This is important not only for understanding rounding errors, but also for understanding how other errors (such as measurement errors) can affect a result.

It's also immensely helpful to understand a bit about how the methods work in practice. On the other hand, you are unlikely to have to build your own dense Gaussian elimination code with blocking for efficiency; you'll probably use a library routine instead. It's more important that you understand the ideas behind the factorizations, and how to apply those ideas to use the factorizations effectively in applications.

Compared to direct methods, iterative methods provide more room for clever, application-specific twists and turns. An iterative method for solving the linear system $Ax = b$ produces a series of guesses

$$\hat{x}^1, \hat{x}^2, \ldots \to x.$$

The goal of the iteration is not always to get the exact answer as fast as possible; it is to get a good enough answer, fast enough to be useful. The rate at which the iteration converges to the solution depends not only on the nature of the iterative method, but also on the structure in the problem. The picture is complicated by the fact that different iterations cost different amounts per step, so a "slowly convergent" iteration may in practice get an adequate solution more quickly than a "rapidly convergent" iteration, just because each step in the slowly convergent iteration is so cheap.

As with direct methods, though, sophisticated iterative methods are constructed from simpler building blocks. In this lecture, we set up one such building block: stationary iterations.

2 Stationary Iterations

A stationary iteration for the equation $Ax = b$ is typically associated with a *splitting* $A = M - N$, where M is a matrix that is easy to solve (i.e. a triangular or diagonal matrix) and N is everything else. In terms of the splitting, we can rewrite $Ax = b$ as

$$Mx = Nx + b,$$

which is the fixed point equation for the iteration

$$Mx^{k+1} = Nx^k + b.$$

If we subtract the fixed point equation from the iteration equation, we have the error iteration

$$Me^{k+1} = Ne^k$$

or

$$e^{k+1} = Re^k, \quad R = M^{-1}N.$$

We've already seen one example of such an iteration (iterative refinement with an approximate factorization); in other cases, we might choose M to be the diagonal part of A (Jacobi iteration) or the upper or lower triangle of A (Gauss-Seidel iteration). We will see in the next lecture that there is an alternate "matrix-free" picture of these iterations that makes sense in the context of some specific examples, but for analysis it is often best to think about the splitting picture.

3 Convergence: Norms and Eigenvalues

We consider two standard approaches to analyzing the convergence of a stationary iteration, both of which revolve around the error iteration matrix $R = M^{-1}N$. These approaches involve taking a norm inequality or using an eigenvalue decomposition. The first approach is often easier to reason about in practice, but the second is arguably more informative.

For the norm inequality, note that if $\|R\| < 1$ for some operator norm, then the error satisfies

$$\|e^{k+1}\| \leq \|R\|\|e^k\| \leq \|R\|^k\|e^0\|.$$

Because $\|R\|^k$ converges to zero, the iteration eventually converges. As an example, consider the case where A is *strictly row diagonally dominant* (i.e. the sum of the magnitudes of the off-diagonal elements in each row are less than the magnitude of the diagonal element), and let M be the diagonal part of A (Jacobi iteration). In that case, $\|R\|_\infty = \|M^{-1}N\|_\infty < 1$. Therefore, the infinity norm of the error is monontonically decreasing[1]

Bounding by one the infinity norm (or two norm, or one norm) of the iteration matrix R is *sufficient* to guarantee convergence, but not *necessary.* In order to completely characterize when stationary iterations converge, we need to turn to an eigenvalue decomposition. Suppose R is diagonalizable, and write the eigendecomposition as

$$R = V\Lambda V^{-1}.$$

Now, note that $R^k = V\Lambda^k V^{-1}$, and therefore

$$\|e^k\| = \|R^k e^0\| = \|V\Lambda^k V^{-1}e^0\| \leq \kappa(V)\rho(R)^k\|e^0\|,$$

where $\rho(R)$ is the *spectral radius* of R, i.e.

$$\rho(R) = \max_{\lambda \text{ an eig}} |\lambda|,$$

and $\kappa(V) = \|V\|\|V^{-1}\|$. For a diagonalizable matrix, convergence of the iteration happens if and only if the spectral radius of R is less than one. *But*

[1]In finite-dimensional spaces, there is a property of "equivalence of norms" that says that convergence in one norm implies convergence in any other norm; however, this does *not* mean that monotone convergence in one norm implies monotone convergence in any other norm.

that statement ignores the condition number of the eigenvector matrix! For highly "non-normal" matrices in which the condition number is large, the iteration may appear to make virtually no progress for many steps before eventually it begins to converge at the rate predicted by the spectral radius. This is consistent with the bounds that we can prove, but often surprises people who have not seen it before.

4 Splittings and Sweeps

Splitting is the right linear algebraic framework for discussing convergence of stationary methods, but it is not the way they are usually programmed. The connection between a matrix splitting and a "sweep" of a stationary iteration like Gauss-Seidel or Jacobi iteration is not always immediately obvious, and so it is probably worth spending a moment or two explaining in more detail.

For the sake of concreteness, let's consider a standard model problem: a discretization of a Poisson equation on a line. That is, we approximate

$$-\frac{d^2u}{dx^2} = f, \quad u(0) = u(1) = 0$$

using the second-order finite difference approximation

$$\frac{d^2u}{dx^2} \approx \frac{u(x-h) - 2u(x) + u(x+h)}{h^2}$$

where h is a small step size. We discretize the problem by meshing $[0, 1]$ with evenly spaced points $x_j = jh$ for $j = 0$ to $N + 1$ where $h = 1/(N + 1)$, then apply this approximation at each point. This procedure yields the equations

$$-u_{j-1} + 2u_j - u_{j+1} = h^2 f_j, \quad j = 1, \dots, N$$

or, in matrix notation

$$Tu = h^2 f$$

where u and f are vectors in $\mathbb{R}^N$ representing the sampled (approximate) solution and the sampled forcing function. The matrix T is a frequentlyrecurring model matrix, the tridiagonal

$$T = \begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{bmatrix}.$$

Suppose we forgot about how cheap Gaussian elimination is for tridiagonal matrices. How might we solve this system of equations? A natural thought is that we could make an initial guess at the solution, then refine the solution by "sweeping" over each node j and adjusting the value at that node (u_j) to be consistent with the values at neighboring nodes. In one sweep, we might compute a new set of values u^{new} from the old values u^{old}:

```
for j = 2:N-1
  unew[j] = (h^2*f[j] + uold[j-1] + uold[j+1])/2;
end
```

or we might update the values for each node in turn, using the most recent estimate for each update, i.e.

```
for j = 2:N-1
  u[j] = (h^2*f[j] + u[j-1] + u[j+1])/2;
end
```

These are, respectively, a step of *Jacobi* iteration and a step of *Gauss-Seidel* iteration, which are two standard stationary methods.

How should we relate the "sweep" picture to a matrix splitting? The update equation from step k to step $k+1$ in Jacobi is

$$-u_{j-1}^{(k)} + 2u_j^{(k+1)} - u_{j+1}^{(k)} = h^2 f_j,$$

while the Gauss-Seidel update is

$$-u_{j-1}^{(k+1)} + 2u_j^{(k+1)} - u_{j+1}^{(k)} = h^2 f_j.$$

In terms of splittings, this means that Jacobi corresponds to taking M to be the diagonal part of the matrix,

$$M = \begin{bmatrix} 2 & & & & & \\ & 2 & & & & \\ & & 2 & & & \\ & & & \ddots & & \\ & & & & 2 & \\ & & & & & 2 \end{bmatrix}, \quad N = \begin{bmatrix} 0 & 1 & & & & \\ 1 & 0 & 1 & & & \\ & 1 & 0 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & 0 & 1 \\ & & & & 1 & 0 \end{bmatrix},$$

while Gauss-Seidel corresponds to taking M to be the lower triangle of the matrix,

$$M = \begin{bmatrix} 2 & & & & & \\ -1 & 2 & & & & \\ & -1 & 2 & & & \\ & & \ddots & \ddots & & \\ & & & -1 & 2 & \\ & & & & -1 & 2 \end{bmatrix}, \quad N = \begin{bmatrix} 0 & 1 & & & & \\ & 0 & 1 & & & \\ & & 0 & 1 & & \\ & & & \ddots & \ddots & \\ & & & & 0 & 1 \\ & & & & & 0 \end{bmatrix}.$$

The point of this exercise is that *programming* stationary iterative methods and *analyzing* the same methods may lead naturally to different ways of thinking about the iterations. It's worthwhile practicing mapping back and forth between these two modes of thought.