2019-11-15

1 Iteration basics

An iterative solver for $Ax = b$ is produces a sequence of approximations $x^{(k)} \to x$. We always stop after finitely many steps, based on some convergence criterion, e.g.

- A residual estimate reached some threshold tolerance (relative to b or to the initial residual).
- An error estimate reached some threshold tolerance (usually relative to the initial error estimate).
- We reach a maximum iteration count.

We say we have solved the problem when some error-related tolerance is satisfied. We can often reason about the cost per step in a simple way, but estimating the steps to a solution can be quite complicated. It depends on the nature of the iteration, the structure of the problem, the norms used to judge convergence, and the problem tolerances.

The oldest and simplest iterations for solving linear systems are *stationary iterations* (a.k.a. *fixed point iterations*) and more generally *relaxation iterations*. In many cases, these iterations have been supplanted by more sophisticated methods (such as Krylov subspace methods), but they remain a useful building block. Moreover, what is old has a way of becoming new again; many of the classic iterations from the 1950s and 1960s are seeing new life in applications to machine learning and large scale optimization problems.

2 Stationary iterations

Stationary iterations are so named because the solution to a linear system is expressed as a stationary point (fixed point) of

$$x^{(k+1)} = F(x^{(k)}).$$

A sufficient (though not necessary) condition for convergence is that the mapping is a contraction, i.e. there is an $\alpha < 1$ such that for all x, y in the vector space,

$$\|F(x) - F(y)\| \leq \alpha \|x - y\|.$$

The constant α is the rate of convergence.

If we are solving a linear equation $Ax = b$, it generally makes sense to write a fixed point iteration where the mapping F is affine. We can write any such iteration via a *splitting* of the matrix A, i.e. by writing $A = M - N$ with M nonsingular. Then we rewrite $Ax = b$ as

$$Mx = Nx + b,$$

and the fixed point iteration is

$$Mx^{(k+1)} = Nx^{(k)} + b,$$

which we may rewrite as

$$x^{(k+1)} = x^{(k)} + M^{-1}(b - Ax^{(k)}).$$

2.1 Error iteration and convergence

We derive an error iteration by subtracting the fixed point equation from the iteration equation

$$\begin{array}{r} Mx^{(k+1)} = Nx^{(k)} + b \\ -[Mx = Nx + b] \\ \hline Me^{(k+1)} = Ne^{(k)} \end{array}$$

or $e^{(k+1)} = Re^{(k)}$ where $R \equiv M^{-1}N$ is the *iteration matrix*. A sufficient condition for convergence is that $\|R\| < 1$ in some operator norm. The necessary and sufficient condition is that $\rho(R) < 1$, where the spectral radius $\rho(R)$ is defined as $\max |\lambda|$ over all eigenvalues λ of R.

The choice of M is key to the success of an iterative method. Ideally, we want it to be easy to solve linear systems with M (low set-up time for any initial factorizations, and a low cost per iteration to solve), but we also want R to have a small norm or spectral radius. Often, there is a direct tension between these two. For example, the "best" choice of M from the perspective of iteration count is $M = A$. But this is a silly thing to do: the iteration converges after one step, but that step is to solve $Ax = b$!

2.2 Complexity of stationary iterations

What is the cost to "solve" a system of linear equations using a stationary iteration? We never exactly solve the system, so we need a convergence criterion to address this problem. Let us instead ask the time to satisfy $\|e^{(k)}\| \leq \epsilon \|e^{(0)}\|$, where $\|e^{(0)}\|$ is the initial error. Supposing $\|R\| < 1$, we know

$$\|e^{(k)}\| \leq \|R\|^k \|e^{(0)}\|,$$

so the criterion should be met after $\lceil \log(\epsilon)/\log(\|R\|) \rceil$ steps. While norms on a finite-dimensional space are all equivalent, the constants involved may depend on the dimension of the space. Therefore, when we analyze the complexity of a stationary iteration, we must specify the family of norms (of either the error or the residual) that we are using to judge convergence.

The cost per step depends on the time to solve a linear system with M and the time to form a residual. For many of the basic stationary iterations, the time per step is $O(\text{nnz}(A))$, where $\text{nnz}(A)$ is the number of nonzero elements in the matrix A. The number of steps, though, depends very strongly on not just the number of nonzeros, but more detailed properties of A. Therefore, we generally cannot describe the asymptotic complexity of an iterative method except in the context of a very specific family of matrices (such as the 2D Poisson model problem).

3 The classical iterations

One of the simplest stationary iterations is *Richardson iteration*, in which M is chosen to be proportional to the identity:

$$\begin{aligned} x^{(k+1)} &= x^{(k)} + \omega(b - Ax^{(k)}) \\ &= (I - \omega A)x^{(k)} + \omega b. \end{aligned}$$

The iteration matrix in this case is simply $R = I - \omega A$. If A is symmetric and positive definite, we can always make Richardson iteration converge with an appropriate ω, though the convergence may be heart-breakingly slow.

Let $A = D - L - U$, where D is diagonal, L is strictly lower triangular, and U is strictly upper triangular. Jacobi iteration takes $M = D$. When we discuss multigrid, we will also see *damped Jacobi*, for which $M = \omega^{-1} D$ with $\omega < 1$. Damped Jacobi is equivalent to moving in the Jacobi direction by

a fraction ω of the usual step length. Like Richardson, we can always make (damped) Jacobi converge for SPD matrices; the method also converges for A strictly diagonally dominant.

The *Gauss-Seidel* iteration incorporates a little more of A into M, taking $M = D - L$. For A symmetric and positive definite, this generally yields about twice the rate of convergence of Jacobi; and it is not necessary to damp the method to obtain convergence. However, Gauss-Seidel is less friendly to parallel computing because the triangular solve involves computing in a strict order.

3.1 Splitting and sweeping

While we typically analyze stationary methods in terms of a splitting, that is not always how we implement them. We can think of either Jacobi or Gauss-Seidel as a *sweep* over the variables, in which we update the value of variable i using the ith equation and using a guess for all the other variables. In the Jacobi iteration, the guess for the other variables comes from the previous step; in Gauss-Seidel, the guess for the other variables involves whatever our most up-to-date information might be. We illustrate this style of programming with two codes, each of which compute a single sweep:

```
% [Unew] = sweep_jacobi(U, F)
%
% Run one Jacobi sweep for 2D Poisson
%
function [Un] = sweep_jacobi(U, F);

  n  = size(U,1)-2;
  h2 = 1/(n+1)^2;
  Un = U;

  for j = 2:n+1
    for i = 2:n+1
      Un(i,j) = (U(i-1,j) + U(i+1,j) + ...
                 U(i,j-1) + U(i,j+1) + h2*F(i,j))/4;
    end
  end

end
```

```
% [U] = sweep_gs(U, F)
%
```

```
% Run one Gauss-Seidel sweep for 2D Poisson
%
function [U] = sweep_gs(U, F);

  n  = size(U,1)-2;
  h2 = 1/(n+1)^2;

  for j = 2:n+1
    for i = 2:n+1
      U(i,j) = (U(i-1,j) + U(i+1,j) + ...
                U(i,j-1) + U(i,j+1) + h2*F(i,j))/4;
    end
  end

end
```

3.2 Over-relaxation

In the Jacobi iteration, we take $M = D$; in Gauss-Seidel, we take $M = D - L$. In general, Gauss-Seidel works better than Jacobi. So if we go even further in the "Gauss-Seidel" direction, perhaps we will do better still? This is the idea behind *successive over-relaxation*, which uses the splitting $M = D - \omega L$ for $\omega > 1$. The case $\omega < 1$ is called *under-relaxation*.

The iteration converges for positive definite A for any $\omega \in (0, 2)$. The optimal choice is problem-dependent; but it is rarely of interest any more, since SOR is mostly used to accelerate more sophisticated iterative methods. Indeed, the most widely-used variant of SOR involves a forward sweep and a backward sweep; this SSOR iteration applied to an SPD A matrix yields an SPD splitting matrix M, and can therefore be used to accelerate the conjugate gradient method (which depends on this structure).

3.3 Red-black ordering

In Jacobi iteration, we can compute the updates for each equation independently of all other updates — order does not matter, and so the method is ripe for parallelism within one sweep[1] In general, though, Gauss-Seidel and over-relaxation methods depend on the order in which we update variables.

[1]There is actually not enough work per sweep to make this worthwhile with ordinary Jacobi, usually, but it is worthwhile if we deal with the block variants.

The *red-black* ordering (or more general *multi-color ordering*) trick involves re-ordering the unknowns in our matrix by "color," where each unknown is assigned a color such that no neighbor in the graph of the matrix has the same color. In the 2D Poisson case, this can be achieved with two colors, usually dubbed "red" and "black," applied in a checkerboard pattern.

3.4 Block iterations

So far, we have restricted our attention to *point relaxation* methods that update a single variable at a time. *Block* versions of Jacobi and Gauss-Seidel have exactly the same flavor as the regular versions, but they update a subset of variables simultaneously. These methods correspond to a splitting with M equal to the block diagonal or block lower triangular part of A.

The block Jacobi and Gauss-Seidel methods update disjoint subsets of variables. The *Schwarz* methods act on *overlapping* subsets It turns out that a little overlap can have a surprisingly large benefit. The book *Domain Decomposition* by Smith, Gropp, and Keyes provides a nice overview.

4 Convergence of stationary iterations

For general non-symmetric (and nonsingular) matrices, none of the classical iterations is guaranteed to converge. But there are a few classes of problems for which we can say something about the convergence of the classical iterations, and we survey some of these now.

4.1 Strictly row diagonally-dominant problems

Suppose A is strictly diagonally dominant. Then by definition, the iteration matrix for Jacobi iteration ($R = D^{-1}(L+U)$) must satisfy $\|R\|_\infty < 1$, and therefore Jacobi iteration converges in this norm. A bound on the rate of convergence has to do with the strength of the diagonal dominance. Moreover, one can show (though we will not) that in this case

$$\|(D-L)^{-1}U\|_\infty \leq \|D^{-1}(L+U)\|_\infty < 1,$$

so Gauss-Seidel converges at least as quickly as Jacobi. The Richardson iteration is also guaranteed to converge, at least so long as $\omega < 1/(\max_i |a_i i|)$, since this is sufficient to guarantee that all the Gershgorin disks of $I - \omega A$ will remain within the unit circle.

4.2 Symmetric and positive definite problems

4.2.1 Richardson iteration

If A is SPD with eigenvalues $0 < \lambda_1 < \ldots < \lambda_n$, then Richardson iteration satisfies

$$\|R\|_2 = \max(|1 - \omega\lambda_1|, |1 - \omega\lambda_n|);$$

and the rate of convergence is optimal when $\omega = 2/(\lambda_1 + \lambda_n)$, which yields

$$\|R\|_2 = 1 - \frac{2\lambda_1}{\lambda_1 + \lambda_n} = 1 - \frac{2}{\kappa(A) + 1}$$

If A is ill-conditioned, the iteration may be painfully slow.

4.2.2 Jacobi iteration

The error iteration for Jacobi is

$$e^{(k+1)} = D^{-1}(L + U)e^{(k)} = D^{-1}(D - A)e^{(k)}.$$

If A is SPD, then so is D, and therefore it induces a norm; scaling the error iteration by $D^{1/2}$ gives

$$\hat{e}^{(k+1)} = D^{-1/2}(D - A)D^{-1/2}\hat{e}^{(k)},$$

where $\hat{e}^{(k)} = D^{1/2}e^{(k)}$ and

$$\|\hat{e}^{(k)}\|_2 = \|e^{(k)}\|_D.$$

Therefore

$$\|e^{(k+1)}\|_D \leq \|D^{-1/2}(D - A)D^{-1/2}\|_2 \|e^{(k)}\|_D.$$

For A is symmetric and positive definite, we then have

$$\|D^{-1/2}(D - A)D^{-1/2}\|_2 = \max(|1 - \lambda_1|, |1 - \lambda_n|),$$

where $0 < \lambda_1 < \ldots < \lambda_n$ are the eigenvalues of the pencil (A, D). We have convergence when $\lambda_n < 2$, i.e. $2D - A$ is symmetric and positive definite. Damped Jacobi, on the other hand, can always be made to converge for a sufficiently large damping level ω.

The same analysis holds for block Jacobi.

4.2.3 Gauss-Seidel iteration

To understand the Gauss-Seidel convergence, it is useful to look at the linear system $Ax^{(*)} = b$ as the minimizer of the convex quadratic

$$\phi(x) = \frac{1}{2}x^T Ax - x^T b.$$

Now consider a given x and consider what happens if we update to $x + se_i$ for some s. This gives the value

$$\phi(x + se_i) = \phi(x) + \frac{a_{ii}}{2}s^2 + se_i^T Ax - sb_i.$$

Minimizing with respect to s yields

$$a_{ii}s = b_i - e_i^T Ax$$

or

$$a_{ii}(x_i + s) = b_i - \sum_{j \neq i} a_{ij}x_j.$$

But this is precisely the Gauss-Seidel update! Hence, Gauss-Seidel for a positive definite system corresponds to optimization of ϕ by *cyclic coordinate descent*. The method decreases ϕ at each coordinate step, and each sweep is guaranteed to sufficiently reduce the objective so that we ultimately converge.

The same analysis holds for block Gauss-Seidel.

4.3 Convergence on the 2D model problem

In the case of the 2D model problem, recall that the eigenvalues are

$$\lambda_{i,j} = 2\left(2 - \cos(\pi ih) - \cos(\pi jh)\right)$$

The extreme eigenvalues are

$$\lambda_{1,1} = 2h^2\pi^2 + O(h^4)$$

and

$$\lambda_{n,n} = 4 - 2h^2\pi^2 + O(h^4).$$

The diagonal of $T_{n\times n}$ is simply $4I$, so the Jacobi iteration matrix looks like

$$R = \frac{1}{4}(4I - T_{n\times n}),$$

or which the eigenvalues are

$$\lambda_{i,j}(R) = -(\cos(\pi i h) + \cos(\pi j h))/2,$$

and the spectral radius is

$$\rho(R) = \cos(\pi h) = 1 - \frac{\pi^2 h^2}{2} + O(h^4)$$

Thus, the number of iterations to reduce the error by $1/e$ scales like

$$\frac{2}{\pi^2 h^2} = \frac{2}{\pi^2}(n+1)^2 = O(N);$$

and since each step takes $O(N)$ time, the total time to reduce the error by a constant factor scales like $O(N^2)$.

he successive overrelaxation iteration uses a splitting

$$M = \omega^{-1}(D - \omega\tilde{L}) = \omega^{-1}D^{-1}(I - \omega L),$$

which yields an iteration matrix

$$R_{SOR} = (I - \omega L)^{-1}((1-\omega)I + \omega U).$$

In general, this is rather awkward to deal with, since it is a nonsymmetric matrix. However, for the model problem with a particular ordering of unknowns (red-black ordering), one has that the eigenvalues μ of R_J correspond to the eigenvalues λ of R_{SOR} via

$$(\lambda + \omega - 1)^2 = \lambda\omega^2\mu^2.$$

For the case $\omega = 1$ (Gauss-Seidel), this degenerates to

$$\lambda = \mu^2,$$

and so $\rho(R_{GS}) = \rho(R_J)^2$. Consequently, each Gauss-Seidel iteration reduces the error by the same amount as two Jacobi iterations, i.e. Gauss-Seidel converges twice as fast on the model problem. This tends to be true for other problems similar to the model problem, too. However, going from Jacobi to Gauss-Seidel only improves the convergence rate by a constant factor; it doesn't improve the asymptotic complexity at all. However optimal ω (about

$2-O(h)$) gives us a spectral radius of $1-O(h)$ rather than $1-O(h^2)$, allowing us to accelerate convergence to $O(N^{3/2})$.

The red-black ordering can be convenient for parallel implementation, because allowing the red nodes (or black nodes) to be processed in any orer gives more flexibility for different scheduling choices. But it is also a useful choice for analysis. For example, in the red-black ordering, the model problem looks like

$$A = \begin{bmatrix} 4I & B \\ B^T & 4I \end{bmatrix}$$

The preconditioner based on Jacobi iteration is

$$M_J = \begin{bmatrix} 4I & 0 \\ 0 & 4I \end{bmatrix},$$

which results in the iteration matrix

$$R_J = M_J^{-1}(M_J - A) = \frac{1}{4}\begin{bmatrix} 0 & B \\ B^T & 0 \end{bmatrix}.$$

The eigenvalues of R_J are thus plus or minus one quarter the singular values of B. Note that this much would have been the same for more general problems with the same structure!

I did not drag you in class through the rest of the analysis, and I would not expect you to repeat it on an exam. Nonetheless, it may be worth writing it out in order to satisfy the curious. The preconditioner for Gauss-Seidel is

$$M_{GS} = \begin{bmatrix} 4I & 0 \\ B^T & 4I \end{bmatrix};$$

and because of the relatively simple form of this matrix, we have

$$M_{GS}^{-1} = \frac{1}{4}\begin{bmatrix} I & 0 \\ B^T/4 & I \end{bmatrix}.$$

The iteration matrix for Gauss-Seidel is

$$R_{GS} = M_{GS}^{-1}(M_{GS} - A) = \begin{bmatrix} 0 & B/4 \\ 0 & -\frac{1}{16}B^TB \end{bmatrix},$$

which has several zero eigenvalues together with some eigenvalues that are minus 1/16 times the squared singular values of B^TB. Thus, as indicated

earlier, the spectral radius of R_{GS} is the square of the spectral radius of R_J (for the model problem).

The analysis for the general SOR case is slightly messier, but I'll include it here for completeness. The preconditioner is

$$M_{SOR} = \frac{1}{\omega} \begin{bmatrix} 4I & 0 \\ \omega B^T & 4I \end{bmatrix},$$

and the inverse is

$$M_{SOR}^{-1} = \frac{\omega}{4} \begin{bmatrix} I & 0 \\ -\omega B^T/4 & I \end{bmatrix},$$

The iteration matrix is

$$\begin{aligned} R_{SOR} &= \frac{1}{4} \begin{bmatrix} I & 0 \\ -\omega B^T/4 & I \end{bmatrix} \begin{bmatrix} 4(1-\omega)I & -\omega B \\ 0 & 4(1-\omega)I \end{bmatrix} \\ &= \begin{bmatrix} (1-\omega)I & -\omega B/4 \\ -(1-\omega)\omega B^T/4 & \omega^2 B^T B/16 + (1-\omega)I \end{bmatrix}. \end{aligned}$$

If λ is any eigenvalue of R_{SOR} except $1-\omega$, we can do partial Gaussian elimination on the eigenvalue equation

$$(R_{SOR} - \mu I)v = 0;$$

after eliminating the first block of variables, we have the residual system

$$\left(\frac{\omega^2}{16} B^T B - (\lambda + \omega - 1)I - \frac{(1-\omega)\omega^2}{16} B^T ((1-\omega-\lambda)I)^{-1} B \right) v_2 = 0,$$

Refactoring, we have

$$\left[\left(\frac{1-\omega}{\lambda+\omega-1} + 1 \right) \frac{\omega^2}{16} B^T B - (\lambda + \omega - 1)I \right] v_2 = 0.$$

From our earlier arguments, letting μ be an eigenvalue of the Jacobi matrix, we know that μ^2 is an eigenvalue of $B^T B/16$. The corresponding eigenvalues λ of R_{SOR} must therefore satisfy

$$\left(\frac{1-\omega}{\lambda+\omega-1} + 1 \right) \omega^2 \mu^2 - (\lambda - \omega - 1) = 0.$$

Multiplying through by $\lambda - \omega - 1$, we have

$$(1 - \omega + \lambda + \omega - 1)\omega^2 \mu^2 - (\lambda - \omega - 1)^2 = 0$$

or

$$\lambda \omega^2 \mu^2 = (\lambda - \omega - 1)^2,$$

which is the formula noted before.