Notes for CS542G (Iterative Solvers for Linear Systems)

Robert Bridson

November 20, 2007

1 The Basics

We're now looking at efficient ways to solve the linear system of equations

$$Ax = b$$

where in this course, we are particularly interested in the case where A is sparse and SPD (symmetric positive-definite). This is the case for the linear systems we have derived for solving the Poisson equation either from finite differences or the Galerkin Finite Element Method.

In the one dimensional case, we saw that A was just tridiagonal for second-order accurate finite differences, or piecewise linear finite elements. This is a very special class of matrices, which we can solve using Cholesky in the optimal $O(n)$ time—in fact, LAPACK provides a fast routine for us.

However, in two or higher dimensions, A has a much more complicated structure and cannot be so easily factored with Cholesky: the resulting lower triangular factor L must (as has been proven) have significantly more nonzeros than A. Making this work efficiently is nontrivial, and has some fairly severe theoretical limits on time and space efficiency for some problems (in particular, solving PDE's on 3D meshes).

An alternative is to look at iterative methods. Here we'll take an initial guess at the solution of the linear system, usually $x_0 = 0$, and in each iteration improve on it to get a sequence of guesses, or iterates, x_k which hopefully will converge to the correct answer.

Before getting to methods of constructing these iterates, we need to have an idea of when to stop: how to measure convergence. Compared to general nonlinear optimization, where it's hard to say in general if a method has converged or not, we have it pretty easy here. Define the **residual** $r = b - Ax$ for a guess x; we have the exact solution if and only if $r = 0$. More to the point, looking at the norm of r is a fairly reliable way of determining if we are close to a solution. In fact, going back to our earlier error analysis of linear systems, the relative error in the solution can be bounded in terms of the residual:

$$\frac{\|x - x_{\text{exact}}\|}{\|x_{\text{exact}}\|} \leq \kappa(A) \frac{\|r\|}{\|b\|}$$

where $\kappa(A)$ is the condition number of the matrix. Assuming that the person who set up the linear system is confident $\kappa(A)$ isn't too large (otherwise, no numerical method can reasonably hope to get an accurate solution!) then it should be fine to say we are converged when $\|r\|$ is small enough. Typically the condition is something like $\|r\| \leq 10^{-9}\|b\|$.

Another look at the SPD problem, which will be useful later, is that the linear system is equivalent to minimizing the following quadratic:

$$\min_x \frac{1}{2} x^T A x - x^T b$$

Taking the gradient of this with respect to x gives us the negative residual $-r$ in fact, and so the solution (where $r = 0$) is a minimum of the quadratic. This of course relies crucially on A being positive definite, as otherwise it might not have a unique minimum.

Another question we'll put to rest now is the question of the initial guess, x_0. One of the possible attractions for iterative methods is that if for some reason you have a good initial guess, you should be able to save a lot of work. However, it's usually convenient for all these methods to be able to assume that the initial guess is all zero: $x_0 = 0$. In fact, if we do have a nonzero initial guess, we can change the linear system to solve for the Δx we need to add to that first guess:

$$\begin{aligned} Ax = b \qquad &\Leftrightarrow \qquad A(x_0 + \Delta x) = b \\ &\Leftrightarrow \qquad A\Delta x = b - Ax_0 \\ &\Leftrightarrow \qquad A\Delta x = r_0 \end{aligned}$$

So now we've reduced it to a new linear system for which the initial guess is $\Delta x_0 = 0$, and the right-hand side is actually the initial residual from the original system. Therefore, we'll from now on assume that the initial guess is zero.

2 Stationary Methods

The first class of iterative methods we'll look at for solving linear systems are called **stationary**. This technically means the iterative procedure is actually a linear operator: the function which maps the right hand side b to the k'th guess x_k is linear.

2.1 Jacobi

Jacobi's method can be understood by considering the equations one at a time. The i'th equation—i.e. the i'th row from $Ax = b$—is this:

$$A_{i1}x_1 + A_{i2}x_2 + \ldots + A_{ii}x_i + \ldots + A_{in}x_n = b_i$$

(Here I've switched to using subscripts for indices into a vector, not labelling which iteration we're on.) If we already have a reasonable guess for the solution, we can improve it by solving this equation for an improved new x_i:

$$\begin{aligned} A_{i1}x_1^{\text{old}} + A_{i2}x_2^{\text{old}} + \ldots + A_{ii}x_i^{\text{new}} + \ldots + A_{in}x_n^{\text{old}} &= b_i \\ \Rightarrow \qquad x_i^{\text{new}} &= \frac{b_i - \sum_{j \neq i} A_{ij}x_j^{\text{old}}}{A_{ii}} \end{aligned}$$

We can simplify the formula a little if we throw in the old value for x_i too:

$$\begin{aligned} x_i^{\text{new}} &= \frac{b_i - \sum_j A_{ij}x_j^{\text{old}} + A_{ii}x_i^{\text{old}}}{A_{ii}} \\ &= x_i^{\text{old}} + \frac{b_i - \sum_j A_{ij}x_j^{\text{old}}}{A_{ii}} \\ &= x_i^{\text{old}} + \frac{1}{A_{ii}}r_i \end{aligned}$$

That's the residual popping up in the last formula. In fact, we can use this now to write the iteration for the whole vector. Let $D = \text{diag}(A)$ be the diagonal part of A: a diagonal matrix with $A_{11}, \ldots, A_{nn}$ down the diagonal. Then Jacobi iteration is:

$$x^{\text{new}} = x^{\text{old}} + D^{-1}r$$

Or, now using subscripts for the iteration again,

$$x_{k+1} = x_k + D^{-1}r_k$$

The big question that looms over this is: does this converge? Do we get closer to the solution as we iterate? Let's take a look at what happens to the residual:

$$\begin{aligned} r_{k+1} &= b - Ax_{k+1} \\ &= b - A(x_k + D^{-1}r_k) \\ &= b - Ax_k - AD^{-1}r_k \\ &= r_k - AD^{-1}r_k \\ &= (I - AD^{-1})r_k \end{aligned}$$

Aha—we now see the k'th residual is just $(I - AD^{-1})^k b$, which converges to zero only if all the eigenvalues of the iteration matrix $I - AD^{-1}$ have magnitude less than one. Unfortunately, this is not always guaranteed to hold—and in general, it's harder to check if this condition holds (which might require finding eigenvalues) that just to run Jacobi and see if it works.

To get a better feel for what Jacobi might be doing, let's take a look at a model problem, our 2nd order finite difference method for solving the Poisson equation, in one dimension. Assuming Dirichlet boundary conditions, and after rescaling by Δx^2, we end up with the following linear system:

$$\begin{pmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & \ddots & \ddots & \ddots \\ & & -1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

Entering this into MATLAB for a reasonably large n, and taking a look at the eigenvectors it finds, we can make the guess that the eigenvectors are sine waves, i.e. guess that the j'th eigenvector has entry i equal to $\sin(\pi ij/(n+1))$, where both i and j range over 1 to n. Sure enough, if we plug this into the eigen equation $Au = \lambda u$

$$-1\left(\sin(\pi(i-1)j/(n+1))\right) + 2\left(\sin(\pi ij/(n+1))\right) - 1\left(\sin(\pi(i+1)j/(n+1))\right) = \lambda_j \sin(\pi ij/(n+1))$$

and use some handy trigonometric formulas, we find it does indeed work, and the eigenvalue is:

$$\lambda_j = 2 - 2\cos(\pi j/(n+1))$$

In particular, the eigenvalues range strictly between 0 and 4. The minimum eigenvalue corresponds to the lowest frequency sine wave; the maximum eigenvalue corresponds to the highest frequency. As an aside, something we'll come back to later, it's not too hard to see that the smallest eigenvalue is $O(1/n^2)$, from the Taylor series expansion of cosine, and since the largest eigenaluve of $O(1)$ the condition number of A is $\kappa(A) = O(n^2)$.

(As another aside: the trig formulas can be a royal pain to work with. One convenient shortcut which gets us most of the way there is to forget about boundary conditions, and just guess the complex exponential $\exp(\sqrt{-1}\omega i)$ for some frequency ω. Exponentials are much simpler to work with, and of course, sine and cosine really are just linear comobinations of complex exponentials.)

Back to Jacobi: for this model problem D is just $2I$, twice the identity. So the iteration matrix is just $I - \frac{1}{2}A$, which must have eigenvalues strictly between -1 and 1. Thus Jacobi converges! Moreover, we can see that the eigencomponents of the residual decay at different rates: for medium frequency components, where $\lambda \approx 2$, Jacobi is extremely efficient. But for low frequency components, where $\lambda \approx 0$, Jacobi hardly changes things at all, and for high frequency components, where $\lambda \approx 4$, Jacobi essentially just flips the sign back and forth also with very slow decay. Referring back to n, the low and high freqency components are multiplied by $\pm 1 - O(1/n^2)$, necessitating $O(n^2)$ iterations to converge.

2.2 Gauss-Seidel

One of the obvious criticisms that could be made of Jacobi is that when we derived the new guess at the i'th entry of the solution, we just used old values for all the other entries. If we're computing them one after the other (as opposed to independently in

parallel), why not use updated solution values as they become available? This gives Gauss-Seidel, a considerably more powerful iterative method.

The Gauss-Seidel update can be written as a loop over i, from 1 up to n, solving the i'th equation for the i'th unknown:

$$\begin{aligned} A_{i1}x_1^{\text{new}} + A_{i2}x_2^{\text{new}} + \ldots + A_{ii}x_i^{\text{new}} + \ldots + A_{in}x_n^{\text{old}} &= b_i \\ \Rightarrow \qquad x_i^{\text{new}} &= \frac{b_i - \sum_{j<i} A_{ij}x_j^{\text{new}} - \sum j > i A_{ij}x_j^{\text{old}}}{A_{ii}} \end{aligned}$$

Compare this with Jacobi to see the difference. Just like with Jacobi we can write it in vector form, by also introducing the strictly lower triangular part of A, labelled L, and the strictly upper triangular of A, labelled U. Note that $A = L + D + U$. Rearranging the solution formula gives:

$$A_{i1}x_1^{\text{new}} + A_{i2}x_2^{\text{new}} + \ldots + A_{ii}x_i^{\text{new}} = b - A_{ii+1}x_i^{\text{old}} - \ldots - A_{in}x_n^{\text{old}}$$

or simply:

$$(L + D)x^{\text{new}} = b - Ux^{\text{old}}$$

and thus Gauss-Seidel iteration is:

$$\begin{aligned} x^{\text{new}} &= (L + D)^{-1}(b - Ux^{\text{old}}) \\ &= (L + D)^{-1}(b - (L + D + U)x^{\text{old}} + (L + D)x^{\text{old}}) \\ &= x^{\text{old}} + (L + D)^{-1}(b - Ax^{\text{old}}) \\ &= x^{\text{old}} + (L + D)^{-1}r \end{aligned}$$

One way to view this is to first observe that the "perfect" (though impractical) iteration $x^{\text{new}} = x^{\text{old}} + A^{-1}r$ would instantly solve the problem; Jacobi approximated A^{-1} with D^{-1}; Gauss-Seidel uses a better approximation $(L + D)^{-1}$.

Indeed, while we won't prove it in this course, it can be shown that Gauss-Seidel always works for SPD matrices A, making it much more robust than Jacobi. For problems where both converge Gauss-Seidel is usually, though not necessarily always, faster too. For the PDE problem we examined, it turns out to be roughly twice as fast—asymptotically still taking $O(n^2)$ iterations though.

Incidentally, one of the downsides to Gauss-Seidel is that it is apparently inherently sequential, whereas Jacobi was nice and parallel. However, if A is sparse, it may be possible to use some artful graph theory to expose parallelism in Gauss-Seidel. View the nonzero pattern of A as the adjacency matrix of a graph on n nodes, i.e. a graph where there is an edge between nodes i and j if and only if $A_{ij} \neq 0$. An independent set of nodes is a set where there are no edges between nodes in the set: you should be able to convince yourself that in Gauss-Seidel we can update an independent set in parallel. If we then colour the nodes of the graph so that each colour induces an independent set (i.e. the usual notion of graph colouring), we see we can solve each colour in parallel. For the finite difference discretization of the Poisson problem, it turns out you only need two colours, alternating as on a chessboard. This is called the "red-black" version of Gauss-Seidel. For reasons we won't get to in this course, red-black Gauss-Seidel can be significantly more effective, in some contexts, than regular Gauss-Seidel even when running in a single thread of execution.

2.3 Successive Over-Relaxation (SOR)

Recall from the preamble that, when we look at the linear system as a quadratic minimization problem, the residual was exactly the negative gradient. We can interpret both Jacobi and Gauss-Seidel as approximations to Newton's method:

$$\begin{aligned} x_{k+1} &= x_k - H^{-1}g \\ &= x_k - A^{-1}(-r) \end{aligned}$$

where $H = A$ is the Hessian and $g = -r$ is the gradient. In Jacobi the Hessian is approximated with just its diagonal part, and in Gauss-Seidel is approximating with the lower triangular part. In both cases we use the natural Newton step length of $\alpha = 1$.

However, we saw that Newton can be made a bit more robust or powerful by allowing more general step lengths. This gives us the idea behind Successive Over-Relaxation, SOR for short. (This peculiar name goes back to a tradition of calling iterative methods for linear systems "relaxation" methods, probably due to a physical interpretation in terms of underlying PDE problems, and the notion that Gauss-Seidel successively "relaxes" each equation.) Here we figure if the Gauss-Seidel step takes us towards the answer, going aggressively a little further might get us there faster:

$$x_{k+1} = x_k + \omega(L + D)^{-1} r_k$$

Here ω is a parameter; $\omega = 1$ corresponds to plain Gauss-Seidel. Usually faster convergence is possible for well-chosen $1 < \omega < 2$, but the actual range in which a significant improvement can be had is typically very small and hard to find in general. That said, for some PDE problems people have worked out optimal parameters, which lead to asymptotically faster convergence (i.e. more than just a constant factor improvement in the number of iterations).

We'll end stationary methods here, though I should point out there are plenty more that people have derived and can be extremely useful in practice. For example, there is a method called MultiGrid which, for the Poisson problem we've studied, gives an optimal solution (converging in $O(1)$ iterations, independent of n, with $O(n)$ work per iteration). However, MultiGrid and similarly powerful methods are rather more complicated than we have time for.