Week 6: Wednesday, Mar 7

From Stationary Methods to Krylov Subspaces

Last time, we discussed *stationary methods* for the iterative solution of linear systems of equations, which can generally be written in the form

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

Stationary methods are simple, and they make good building blocks for more sophisticated methods, but for most purposes they have been superceded by faster-converging *Krylov subspace* methods. The book describes one of these, the conjugate gradient method (CG), in a little detail. In these notes, we will describe the same iteration, but from a slightly different perspective.

When all you have is a hammer...

Suppose that we want to solve $Ax = b$, but the only information we have about A is a subroutine that can apply A to a vector. What should we do? If the only operation at hand is matrix mutiplication, and the only vector staring us in the face is b, a natural approach would be to take b and start multiplying by A to get b, Ab, A^2b, etc. Taking linear combinations of these vectors gives us a *Krylov subspace*:

$$\mathcal{K}_k(A, b) = \operatorname{span}\{b, Ab, A^2b, \ldots, A^{k-1}b\}.$$

Given the examples that we've seen so far, you might reasonably wonder why we're assuming that we can do so little with A. In fact, there are many cases where working with the entries of A is a pain, but evaluating a matrix-vector product can be done efficiently. One set of examples comes from image and signal processing, where many linear operations can be applied efficiently using Fourier transforms. Another example, which we may encounter again after the break when we talk about Newton methods for solving systems of linear equations, is approximate multiplication by a Jacobian matrix using finite differences. Since this example reinforces some calculus concepts that have occurred repeatedly, let's go through it in a little detail.

Suppose $f : \mathbb{R}^n \to \mathbb{R}^n$ has two continuous derivatives. At a point x, the *Jacobian matrix* $J(x)$ is a matrix of partial derivatives of f:

$$J_{ij} = \frac{\partial f_i}{\partial x_j}(x).$$

The Jacobian matrix can be used to calculate *directional derivatives* using the chain rule; if u is some direction, then

$$\frac{\partial f}{\partial u}(x) = \left.\frac{d}{dt}\right|_{t=0} f(x + tu) = J(x)u.$$

In a calculus course, you might learn that multiplying the Jacobian matrix by a direction vector is a way to compute a directional derivative. Here, we take the opposite approach: to approximate the product of a Jacobian matrix and a vector, we approximate a directional derivative:

$$J(x)u = \left.\frac{d}{dt}\right|_{t=0} f(x + tu) \approx \frac{f(x + hu) - f(x)}{h}.$$

If f is a "black box" function that is hard to differentiate analytically, this numerical approach to computing matrix-vector products with the Jacobian is sometimes incredibly useful.

From Linear Systems to Optimization

There are two basic ingredients to Krylov subspace methods:

1. Build a sequence of Krylov subspaces.
2. Find approximate solutions in those subspaces.

The conjugate gradient method (CG) pulls out approximate solutions to $Ax = b$ when A is a symmetric positive definite matrix by turning the problem of linear system solving into an equivalent optimization problem. That is, the functional

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

has a single critical point at

$$0 = \nabla\phi(x) = Ax - b.$$

By the second derivative test, this critical point is a global minimum for ϕ. The conjugate gradient method finds an approximate solution $x^{(k)} \in \mathcal{K}_k(A, b)$ by minimizing $\phi(x)$ over the Krylov subspace. In exact arithmetic, this is guaranteed to converge to the true solution in at most n steps, but in practice we usually get very good approximations in far fewer than n steps.

The *what* of the CG algorithm, then, is straightforward: at step k, the method produces an approximate solution $x^{(k)}$ that minimizes $\phi(x)$ over the kth Krylov subspace. The *how* of the algorithm involves a lot of beautiful mathematical connections and a little bit of magic. I will not discuss the implementation of CG further, except to point to the very popular article by Jonathan Shewchuck on "The Conjugate Gradient Method without the Agonizing Pain," which provides a gentle introduction to the particulars[1].

Preconditioning

For the fastest convergence, Krylov subspace methods like conjugate gradients should be used with a *preconditioner*. That is, instead of solving $Ax = b$, we solve

$$M^{-1}Ax = M^{-1}b,$$

where applying M^{-1} should approximate A^{-1} in some very loose sense[2] be reasonably inexpensive. Thinking back to the last lecture, we recall that $M^{-1}A$ also showed up when we were analyzing the stationary iteration

$$y^{(k+1)} = y^{(k)} - M^{-1}(Ay^{(k)} - b)$$

Note that if we start this iteration at $y^{(0)} = 0$, then

$$y^{(k)} \in \mathcal{K}_k(M^{-1}A, M^{-1}b).$$

[1]I personally prefer the terser — more painful? — treatment in standard numerical linear algebra texts like those of Demmel or Golub and Van Loan, or even just the "template" given in the book *Templates for the Solution of Linear Systems* (which is freely available online). Then again, I grew up to be a numerical analyst. Your mileage may vary.

[2]Really, M^{-1} should be chosen so that the eigenvalues of $M^{-1}A$ are clustered, though this characterization tends to be more useful for analysis than for preconditioner design. I tried to walk through this in lecture, and realized about 3/4 of the way through my explanation that I'd lost everybody by pulling in too much about eigenvalues too quickly. Sorry about that. Don't worry, it won't be on any exams.

That is, the first k steps of a stationary iteration with the splitting matrix M form a basis for a *preconditioned* Krylov subspace $\mathcal{K}_k(M^{-1}A, M^{-1}b)$. Since Krylov subspace methods try to find the best possible solution within a subspace (where the definition of "best" varies from method to method), using M as a preconditioner for a Krylov subspace method typically yields better approximations than using M as the basis for a stationary method. However, the M matrices used in classical stationary methods such as the Jacobi, Gauss-Seidel, and SOR iterations are frequently used as default preconditioners. It is often possible to construct much better preconditioners for specific classes of matrices using a combination of hard analysis and physical intuition, but this is beyond the scope of what we will cover in this class.

Problems to ponder

1. A is *strictly diagonally dominant* if for each i,
$$a_{ii} > \sum_{j \neq i} |a_{ij}|.$$
Show that Jacobi iteration necessarily converges for strictly diagonally dominant matrices.

2. If A is a sparse rectangular matrix, MATLAB provides a so-called "Q-less" QR decomposition in which R is a sparse matrix and $Q = AR^{-1}$ is never formed explicitly. Suppose the computed factor $\hat{R}$ is contaminated by a small amount of noise. Describe a stationary method that nonetheless uses this computed factor to compute $\operatorname{argmin}_x \|Ax - b\|^2$ accurately in a few steps[3].

3. Show that if $\phi(x) = \frac{1}{2} x^T A x - x^T b$ and A is symmetric, then $\nabla \phi(x) = Ax - b$.

4. Given a guess x^{old}, consider the update $x^{\mathrm{new}} = x^{\mathrm{old}} + t e_i$. For what value of t is $\phi(x^{\mathrm{new}})$ minimized? For the one-dimensional Poisson model problem, show that updating the ith component of x according to this rule is equivalent to step i in a Gauss-Seidel sweep[4]

5. Suppose $x^* = \operatorname{argmin}_{x \in \mathcal{U}} \phi(x)$. Show that $Ax^* - b$ is orthogonal to every vector in $\mathcal{U}$.

6. Show that $\langle x^* - A^{-1}b, x^* \rangle_A = 0$ (use the previous question).

7. Show that minimizing $\phi(x^*)$ over all vectors in $\mathcal{U}$ also minimizes $\|x^* - A^{-1}b\|_A^2$ over all vectors in $\mathcal{U}$.

[3] If you are following along offline, type `help qr` in MATLAB — the online documentation describes exactly the iterative refinement strategy I have in mind.

[4] This is true for matrices other than the model problem, too; in general, Gauss-Seidel for a symmetric positive definite system will monotonically reduce the value of ϕ.