Notes for 2015-03-13

From Lanczos to CG

In the last lecture, we developed the Lanczos iteration, which for a symmetric matrix A implicitly generates the decomposition

$$AQ^{(k)} = Q^{(k)}T^{(k)} + \beta_k q_{k+1}$$

where $T^{(k)}$ is a tridiagonal matrix with $\alpha_1, \ldots, \alpha_k$ on the diagonal and $\beta_1, \ldots, \beta_{k-1}$ on the subdiagonal and superdiagonal. The columns of $Q^{(k)}$ form an orthonormal basis for the Krylov subspace $\mathcal{K}_k(A, b)$, and are a numerically superior alternative to the power basis. We now turn to using this decomposition to solve linear systems.

The *conjugate gradient* algorithm can be characterized as a method that chooses an approximation $\tilde{x}^{(k)} \in \mathcal{K}_k(A, b)$ by minimizing the energy function

$$\phi(z) = \frac{1}{2} z^T A z - z^T b$$

over the subspace. Writing $\tilde{x}^{(k)} = Q^{(k)}u$, and using the fact that

$$(Q^{(k)})^T A Q^{(k)} = T^{(k)}$$
$$(Q^{(k)})^T b = \|b\| e_1$$

we have

$$\phi(Q^{(k)}u) = \frac{1}{2} u^T T^{(k)} u - \|b\| u^T e_1.$$

The stationary equations in terms of u are then

$$T^{(k)}u = \|b\| e_1.$$

In principle, we could solve find the CG solution by forming and solving this tridiagonal system at each step, then taking an appropriate linear combination of the Lanczos basis vectors. Unfortunately, this would require that we keep around the Lanczos vectors, which eventually may take quite a bit of storage. This is essentially what happens in methods like GMRES, but for the method of conjugate gradients, we have not yet exhausted our supply of cleverness. It turns out that we can derive a short recurrence relating the solutions at consecutive steps and their residuals. There are several different ways to this recurrence: one can work from a factorization of the nested tridiagonal matrices $T^{(k)}$, or work out the recurrence based on the optimization interpretation of the problem (this leads to the name "conjugate gradients").

Practical Matters

CG is popular for several reasons:

- The only thing we need A for is to form matrix-vector products. An implicit representation of A may be sufficient for this (and is in some cases more convenient than an explicit version).
- Because it involves a short recurrence, CG can be run for many steps without an excessive amount of storage or other overheads involving looking over many past vectors. This is not true of all other CG methods.
- Convergence is faster than with stationary methods.

This last point requires some clarification, and will take up most of the rest of our discussion.

When we started talking about Krylov subspaces, we described searching over the spaces $\mathcal{K}_m(M^{-1}A, M^{-1}b)$, where M is the matrix appearing in the splitting for some stationary iteration. When we derived CG, though, the matrix M disappeared. In fact, it disappeared only to keep the presentation as uncluttered as I could manage. In practice, CG and other Krylov subspace methods are typically used with a *preconditioner* M, and one looks over $\mathcal{K}_m(M^{-1}A, M^{-1}b)$ for solutions. The preconditioner should have the following properties

- As in a splitting for a stationary method, it should be easy to apply M^{-1}. There is a tradeoff here: the quality of the subspace and the efficiency with which M^{-1} can be applied may be in conflict. Note that solving systems involving M is the only way in which M is used (just as applying A is the only way in which A is used).
- The preconditioner does not need to correspond to a convergent stationary iteration, but M^{-1} should "look like" A^{-1} in that $M^{-1}A$ should have eigenvalues in clusters. If all the eigenvalues are the same, preconditioned CG (or other preconditioned Krylov subspace methods) will converge in one step.
- For CG, the preconditioner must be symmetric and positive definite.

Are preconditioners really necessary? For problems coming from PDE discretizations or computations on networks, there is sometimes an intuitive way to see why the answer is "yes" if one wants fast convergence. Consider, for example, the model tridiagonal matrix T in $\mathbb{R}^{N\times N}$, and imagine we want to solve $Tx = e_1$. The last component of x is not that tiny, but notice that all the vectors in a Krylov subspace $\mathcal{K}_m(T, e_1)$ are zero for every index after m. Therefore, there is no way a method that explores these Krylov subspaces will be close to converged for $m < N$ iterations. Given that we know how to solve this tridiagonal system in $O(N)$ time with Gaussian elimination, this is a bit disheartening! The issue is that method simply doesn't move information through the mesh fast enough to achieve rapid convergence.

One way of deriving preconditioners is to look at the splittings used in classical stationary iterations. Another approach is incomplete factorization – carry out Gaussian elimination or Cholesky, but throw away nonzeros that are small or appear in inconvenient places. However, the best preconditioners are often specific to particular applications. Some examples are:

- For discretizations of PDEs with variable coefficients, we might use a preconditioner based on a much more regular PDE (e.g. with constant coefficients and a regular geometry). The preconditioner solve could potentially be applied by fast transform methods.

- Multigrid preconditioners take advantage of the fact that one can often discretize a continuous problem using coarser or finer meshes, and the coarse meshes can be used to suppress parts of the error that are hard to reach by applying standard stationary methods to fine grids.

- Domain decomposition preconditioners split the problem into subproblems. If the subproblems are treated completely independently, domain decomposition looks like a block version of Jacobi iteration; but if the subproblems overlap, or if they are coupled together in some other way, one gets any of a variety of interesting methods (additive and multiplicative Schwarz, FETI, BDDC, etc).

Frameworks like PETSc and Trilinos provide both standard iterative solvers like CG and a menu of different preconditioners. Choosing the right preconditioner is in general hard, and choosing the parameters that determine the detailed behavior is hard.