Notes for 2016-03-21

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.

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.

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.

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

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

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 frequently-

recurring 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 = 1:N
  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 = 1:N
  u(j) = (h^2*f(j) + u(j-1) + u(j+2))/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.

Linear Solves and Quadratic Minimization

We have already briefly described an argument that Jacobi iteration converges for strictly row diagonally dominant matrices. We now discuss an argument that Gauss-Seidel converges (or at least part of such an argument). In the process, we will see a useful way of reformulating the solution of symmetric positive definite linear systems that will prepare us for our upcoming discussion of conjugate gradient methods.

Let A be a symmetric positive definite matrix, and consider the "energy" function

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

The stationary point for this function is the point at which the derivative in any direction is zero. That is, for any direction vector u,

$$\begin{aligned} 0 &= \left.\frac{d}{d\epsilon}\right|_{\epsilon=0} \phi(x+\epsilon u) \\ &= \frac{1}{2}u^T Ax + \frac{1}{2}x^T Au - u^T b \\ &= u^T(Ax-b) \end{aligned}$$

Except in pathological instances, a directional derivative can be written as the dot product of a direction vector and a gradient; in this case, we have

$$\nabla\phi = Ax - b.$$

Hence, minimizing ϕ is equivalent to solving $Ax = b$ [2].

Now that we have a characterization of the solution of $Ax = b$ in terms of an optimization problem, what can we do with it? One simple approach is to think of a sweep through all the unknowns, adjusting each variable in term to minimize the energy; that is, we compute a correction Δx_j to node j such that

$$\Delta x_j = \operatorname{argmin}_z \phi(x + ze_j)$$

Note that

$$\frac{d}{dz}\phi(x+ze_j) = e_j^T(A(x+ze_j)-b),$$

and the update $x_j := x_j + \Delta x_j$ is equivalent to choosing a new x_j to set this derivative equal to zero. But this is exactly what the Gauss-Seidel update does! Hence, we can see Gauss-Seidel in two different ways: as a stationary method for solving a linear system, or as an optimization method that constantly makes progress toward a solution that minimizes the energy [3]. The latter perspective can be turned (with a little work) into a convergence proof for Gauss-Seidel on positive-definite linear systems.

Extrapolation: A Hint of Things to Come

Stationary iterations are simple. Methods like Jacobi or Gauss-Seidel are easy to program, and it's (relatively) easy to analyze their convergence. But

[2] If you are unconvinced that this is a minimum, work through the algebra to show that $\phi(A^{-1}b + w) = \frac{1}{2}w^T Aw$ for any w.

[3] Later in the class, we'll see this as coordinate-descent with exact line search.

these methods are also often slow. We'll talk next time about more powerful *Krylov subspace* methods that use stationary iterations as a building block.

There are many ways to motivate Krylov subspace methods. We'll pick one motivating idea that extends beyond the land of linear solvers and into other applications as well. The key to this idea is the observation that the error in our iteration follows a simple pattern:

$$x^{(k)} - x = e^{(k)} = R^k e^{(0)}, \quad R = M^{-1}N.$$

For large k, the behavior of the error is dominated by the largest eigenvalue and the associated eigenvector[4], i.e.

$$e^{(k+1)} \approx \lambda_1 e^{(k)}.$$

Note that this means

$$x^{(k)} - x^{(k+1)} = e^{(k)} - e^{(k+1)} \approx (1 - \lambda_1) e^{(k)}.$$

If we have an estimate for λ_1, we can write

$$x = x^{(k)} - e^{(k)} \approx x^{(k)} - \frac{x^{(k)} - x^{(k+1)}}{1 - \lambda_1}.$$

That is, we might hope to get a better estimate of x than is provided by $x^{(k)}$ or $x^{(k+1)}$ individually by taking an appropriate linear combination of $x^{(k)}$ and $x^{(k+1)}$. This idea generalizes: if we have a sequence of approximations $x^{(0)}, \ldots, x^{(k)}$, why not ask for the "best" approximation that can be written as a linear combination of the $x^{(j)}$? This is the notion underlying methods such as the conjugate iteration iteration, which we will discuss shortly.

The Big Picture

We now start with our discussion of Krylov subspace methods in general, and the famous method of conjugate gradients (CG) in particular. Though this is the iterative method of choice for most positive definite systems, it may be as famously confusing as it is famous[5]. In order to avoid getting lost in the weeds, it seems worthwhile to start with a roadmap:

[4] If you don't understand this now, you will when we talk about the power method in a week or so!

[5] See, e.g., "Introduction to the Conjugate Gradient Method Without the Agonizing Pain" by Jonathan Shewchuk.

- We begin with the observation that if vectors $x^{(0)}, \ldots, x^{(k)}$ are increasingly good approximations to x, then some linear combination of these vectors may produce an even better approximation. If the original sequence is produced by a stationary iteration, these vectors span a *Krylov subspace*.

- One can generally show that a big enough Krylov subspace will contain a good approximation to x. Alas, this does not tell us how to find which vector in the space is best (or even good)! Attempting to minimize the norm of the error is usually impossible, but it is possible to minimize the residual (leading to GMRES or MINRES), an energy function (CG), or some other error-related quantity.

- The basic framework of a Krylov subspace plus a method of choosing approximations from the space allows us to describe some theoretical properties of several iterations without telling us why (or if) we can implement them efficiently and stably. A key practical point is the computation of well-conditioned bases for the Krylov subspaces, e.g., using the *Lanczos* algorithm (symmetric case) or the *Arnoldi* algorithm (nonsymmetric case).

From Stationary Methods to Krylov Subspaces

Earlier in these notes, we tried to motivate the idea that we can improve the convergence of a stationary method by replacing the sequence of guesses

$$x^0, x^1, \ldots \to x$$

with *linear combinations*

$$\tilde{x}^k = \sum_{j=1}^{k} \alpha_{kj} x^j.$$

We could always choose α_{kj} to be one for $k = j$ and zero otherwise, in which case we have the original stationary method; but by choosing the coefficients more carefully, we might do better.

We've so far written stationary methods as

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

This is equivalent to

$$x^{j+1} = x^j + M^{-1}r^j, \quad r^j \equiv b - Ax^j,$$

or

$$x^{j+1} = Rx^j + M^{-1}b$$

where $R = I - M^{-1}A$ is the iteration matrix we've seen in our previous analysis. If $x^0 = M^{-1}b$, this gives

$$x^j = \sum_{i=1}^{j} R^i M^{-1} b.$$

If we look at this expression closely, we might notice that the space spanned by the first k iterates of the stationary method is all vectors of the form

$$\sum_{i=0}^{j} c_i R^i M^{-1} b.$$

If we look a little harder, we might observe that this is equivalent to the space of all vectors of the form

$$\sum_{i=0}^{j} c_i (M^{-1}A)^i M^{-1} b = p(M^{-1}A)M^{-1}b$$

where $p(z) = \sum_{i=1}^{j} c_i z^i$ is a polynomial of degree at most j.

In general, the d-dimensional Krylov subspace generated by a matrix A and vector B is

$$\begin{aligned} \mathcal{K}_d(A, b) &= \text{span}\{b, Ab, A^2b, \ldots, A^{d-1}b\} \\ &= \{p(A)b : p \in \mathcal{P}_{d-1}\}. \end{aligned}$$

As we have just observed, the iterates of a stationary method form a basis for nested Krylov subspaces generated by $M^{-1}A$ and $M^{-1}b$. If the stationary method converges, we know the Krylov subspaces will eventually contain pretty good approximations to $A^{-1}b$. Let's now spell this out a little more carefully.

The Power of Polynomials

We showed a moment ago that the first m iterates of a stationary method form a basis for the space

$$\mathcal{K}_{m+1}(R, M^{-1}b) = \mathcal{K}_{m+1}(M^{-1}A, M^{-1}b)$$

What can we say about the quality of this space for approximation? As it turns out, this turns into a question about polynomial approximations. We will not spell out all the details (nor will this appear on any exams or homework problems for this class), but it's worth spending a few moments giving an idea of what happens.

We have seen that the iterates of the stationary method are

$$x^{(k)} = x + e^{(k)} = x + R^k e^{(0)}$$

We would like to take a linear combination

$$\tilde{x}^{(m)} = \sum_{k=0}^{m} \gamma_{mk} x^{(k)} = p_m(1)x + p_m(R)e^{(0)}$$

where $p_m(z) = \sum_{k=0}^{d} \gamma_{mk} z^k$. Moreover, if R is diagonalizable with $R = V\Lambda V^{-1}$, then

$$p_m(R) = V p(\Lambda) V^{-1}.$$

For any p_m with $p_m(1) = 1$, we have

$$\begin{aligned} \|\tilde{e}^{(m)}\| &= \|\tilde{x}^{(m)} - x\| \\ &= \|p_m(R)e^{(0)}\| = \|V p(\Lambda) V^{-1} e^{(0)}\| \\ &\leq \kappa(V) \max_{\lambda_j} |p(\lambda_j)| \|e^{(0)}\|. \end{aligned}$$

Hence, we would really like to choose the polynomial that is one at 1 and as small as possible on each of the eigenvalues.

If all eigenvalues λ_j of R are real, then we have

$$\max_{\lambda_j} |p(\lambda_j)| \leq \max_{|z|<\rho(R)} |p(z)|,$$

and a reasonable way to choose polynomials is to minimize $|p_m(z)|$ on $[-\rho(R), \rho(R)]$ subject to the constraint $p_m(1) = 1$. The solution to this problem is the *scaled*

Chebyshev polynomials, with which we can show that the optimal p_m satisfies

$$\begin{aligned} p_m(z) &\leq \frac{2}{1 + m\sqrt{2/(1-\rho(R))}} \\ &= 2(1 - m\sqrt{2(1-\rho(R))}) + O(m^2(1-\rho(R)). \end{aligned}$$

While the number of steps for the basic stationary iteration to reduce the error by a fixed amount scales roughly like $(1-\rho(R))^{-1}$, the number of steps to reduce the bound on the optimal error scales like $(1-\rho(R))^{-1/2}$.

While the Chebyshev bounds are correct, and involve a beautiful bit of approximation theory, they are limited. For one thing, they fall apart on non-normal matrices with complex spectra. Even in the SPD case, these bounds are often highly pessimistic in practice. When the eigenvalues of R come in clusters, a relatively low degree polynomial can do a good job of approximating λ^{-1} at each of the clusters, and hence a relatively low-dimensional Krylov subspace may provide excellent solutions.

All of this is to say that the detailed convergence theory for Krylov subspace methods can be quite complicated, but understanding a little about the eigenvalues of the problem can provide at least a modicum of insight.

For theoretical work, we are fine writing Krylov subspaces as polynomials in R applied to $M^{-1}b$. In practical computations, though, we need a basis. Because the iterates of the stationary method are converge to x, they tend to form a very ill-conditioned basis. We would like to keep the same Krylov subspace, but have a different basis – say, for instance, an orthonormal basis. We turn to this task next.

The Lanczos Idea

What is good about the "power basis" for a Krylov subspace? That is, why might we like to write

$$\mathcal{K}_m(A,b) = \text{span}\{b, Ab, A^2b, \ldots, A^{m-1}b\}$$

rather than choosing a different basis? Though it's terrible for numerical stability, there are two features of the basis that are attractive:

- The power bases span nested Krylov subspaces. Given the vectors $b, \ldots, A^{m-1}b$ spanning $\mathcal{K}_m(A,b)$, we only need one more vector (A^db) to span $\mathcal{K}_{m+1}(A,b)$.

- Each successive vector can be computed from the previous vector with a single multiplication by A. There is no other overhead.

While we dislike the power basis from the perspective of stability, we would like to keep these attractive features for any alternative basis.

We've already described one approach to converting the vectors in a power basis into a more convenient set of vectors. Define the matrix

$$X^{(m)} = \begin{bmatrix} b & Ab & A^2b & \dots & A^{m-1}b \end{bmatrix}$$

and consider the economy QR decomposition

$$X^{(m)} = Q^{(m)} R^{(m)}, \quad Q^{(m)} = \begin{bmatrix} q_1 & q_2 & \dots & q_m \end{bmatrix}.$$

The columns of $Q^{(m)}$ are orthonormal and for any $k \leq m$, the first k columns of $Q^{(m)}$ span the same space as the first k columns of $X^{(m)}$. But forming $X^{(m)}$ and running QR is unattractive for two reasons. First, a dense QR decomposition may involve a nontrivial amount of extra work, particularly as m gets large; and second, simply forming the rounded version of $X^{(m)}$ is enough to get us into numerical trouble, even if we were able to run QR with no additional rounding errors. We need a better approach.

One helpful observation is that

$$\mathcal{R}(AQ^{(k)}) = \mathcal{R}(AX^{(k)}) \subseteq \mathcal{R}(X^{(k+1)}) = \mathcal{R}(Q^{(k+1)}).$$

That is, Aq_k can always be written as a linear combination of $q_1, \dots, q_{k+1}$. In matrix terms, this means we can write

$$AQ^{(k)} = Q^{(k+1)} \bar{H}^{(k)}$$

where

$$\bar{H}^{(m)} = \begin{bmatrix} h_{11} & h_{12} & h_{13} & \dots & h_{1,k-1} & h_{1k} \\ h_{21} & h_{22} & h_{23} & & h_{2,k-1} & h_{2k} \\ 0 & h_{32} & h_{33} & & h_{3,k-1} & h_{3k} \\ & 0 & h_{43} & & h_{4,k-1} & h_{4k} \\ & & \ddots & \ddots & \vdots & \vdots \\ & & & 0 & h_{k,k-1} & h_{kk} \\ & & & & 0 & h_{k+1,k} \end{bmatrix}.$$

A matrix with this structure (all elements below the first subdiagonal equal to zero) is called *upper Hessenberg*. Alternately, we write

$$AQ^{(k)} = Q^{(k)} H^{(k)} + q_{k+1} h_{k+1,k}$$

where $H^{(k)}$ is the square matrix consisting of all but the last row of $\bar{H}^{(k)}$. This formula is sometimes called an *Arnoldi decomposition*; it turns out to be crucial in the development of GMRES, one of the most popular iterative solvers for nonsymmetric linear systems. For the moment, though, we want to focus on the symmetric case, and so we will move on.

When A is symmetric, we have that

$$(Q^{(k)})^T A Q^{(k)} = H^{(k)}$$

is also symmetric, as well as being upper Hessenberg; in this case, the matrix $H^{(k)}$ is actually *tridiagonal*, and we write $H^{(k)}$ as $T^{(k)}$ to emphasize this fact. Conventionally, $T^{(k)}$ is written as

$$T^{(k)} = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \alpha_3 & \ddots & \\ & & \ddots & \ddots & \beta_{k-1} \\ & & & \beta_{k-1} & \alpha_k \end{bmatrix}.$$

Converting from matrix notation back into vector notation, we have

$$A q_k = \beta_{k-1} q_{k-1} + \alpha_k q_k + \beta_k q_{k+1},$$

which we can rearrange as

$$\beta_k q_{k+1} = A q_k - \alpha_k q_k - \beta_{k-1} q_{k-1}.$$

where

$$\begin{aligned} \alpha_k &= q_k^T A q_k \\ \beta_{k-1} &= q_k^T A q_{k-1} = q_{k-1}^T A q_k. \end{aligned}$$

Putting everything together, we have the *Lanczos iteration*, in which we obtain each successive vector q_{k+1} by forming Aq_k, orthogonalizing against the q_k and q_{k-1} by Gram-Schmidt, and normalizing. We saw this iteration earlier in the semester when we talked about eigenvalue problems and compared different methods of matrix tridiagonalization.

Presented as a *fait accompli*, the Lanczos iteration looks like magic. It seems even more like magic when one realizes that despite an instability in

the iteration (because of the use of Gram-Schmidt for orthonormalization at each step), the iteration still produces useful information in floating point. The Lanczos iteration is the basis for one of the most popular iterative methods for solving eigenvalue problems, and in that setting it is important to acknowledge and deal with the instability in the method. For the moment, though, we are still interested in solving linear systems, and the method of Conjugate Gradients (also built on Lanczos) turns out to still work great.

1 Addendum: Three-Term Recurrences

The Lanczos iteration allows us to generate a sequence of orthonormal vectors using a three-term recurrence. As it turns out, the same approach leads to three-term recurrences that generate families of orthogonal polynomials, including the Chebyshev polynomials mentioned in passing earlier and the Legendre polynomials that play a significant role in the development of Gaussian quadrature. I consider the details beyond of these connections to be beyond the scope of the current class. But the connections are too beautiful and numerous to not mention that they exist. I would hate for you to walk away from this class with the impression that the mathematical development of the Lanczos iteration is only some quirky trick in numerical linear algebra that gets you part of the way to CG.

From Lanczos to CG

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"). For a detailed discussion, my preferred reference is *Templates for the Solution of Linear Systems*, a short book available from SIAM which is also freely available online. The paper "Introduction to the Conjugate Gradient Method Without the Agonizing Pain" provides a longer and possibly gentler introduction.

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.