Week 13: Friday, Nov 20

Logistics

1. There is a typo in the hint in HW 5 p3; it should be addition, not subtraction.

Arnoldi iteration

There is a generalization of the Lanczos iteration to nonsymmetric matrices: the Arnoldi iteration, a basic version of which is here:

```
function [Q,H] = lec34arnoldi(A,q1,m)
%
% Run m steps of basic Arnoldi iteration

n = length(A);
Q = zeros(n,m+1);
H = zeros(m+1,m);
Q(:,1) = q1;

for j = 1:m
  r = A*Q(:,j);
  for k = 1:j
    H(k,j) = Q(:,k)'*r;
    r      = r-Q(:,k)*H(k,j);
  end
  H(j+1,j) = norm(r);
  Q(:,j+1) = r/H(j+1,j);
end
```

Like the Lanczos iteration, the Arnoldi iteration builds up an orthonormal basis for a Krylov subspace, generating q_{j+1} by orthogonalizing Aq_j against the previous vectors (by modified Gram-Schmidt). Unlike with the Lanczos iteration, producing q_{j+1} requires that we store and orthogonalize against *all* the previous basis vectors $q_1, \ldots, q_j$. This quickly becomes expensive, so in practice one never does too many steps of Arnoldi iteration before *restarting*.

Restarting Arnoldi

If we have generated a Krylov subspace $\mathcal{K}_m(A, q_1)$ via the Arnoldi procedure, we can *restart* by choosing some vector $\hat{q}_1 \in \mathcal{K}_m(A, q_1)$ and running Arnoldi again starting at $\hat{q}_1 = f(A)q_1$, where f is a *filter polynomial* chosen to "damp out" components of $f(A)q_1$ in eigendirections that we do not want. This is known as *explicit restarting*. We choose the vector $\hat{q}_1$ so that it is large in the directions we find interesting and small in the other directions. For example, if A has real eigenvalues and we are interested in the largest ones, we might choose $\hat{q}_1$ to lie in the span of the largest few Ritz vectors.

Unfortunately, a forward instability in Arnoldi iteration tends to causes explicit restarting to produce slow convergence to a few extremal eigenvalues (the method remains backward stable, so slow convergence is more an issue than misconvergence). An alternate idea is *implicitly* restarting. To implicitly restart, we note that if f is a degree p filter polynomial, then $\mathcal{K}_{m-p}(A, f(A)q_1) \subseteq \mathcal{K}_m(A, q_1)$, and so we can implicitly perform the first $m - p$ steps of Arnoldi iteration starting with $f(A)q_1$ by manipulating an existing Krylov subspace basis. Specifically, we apply p shifted Hessenberg QR steps to the partially-constructed Hessenberg matrix H in the Arnoldi procedure, and note that the leading $m - p$ columns of the resulting basis depend on only the leading m columns of the original basis. This *implicitly restarted Arnoldi method* is one of the most popular methods for computing eigenvalues and vectors of large, sparse, nonsymmetric matrices, and it is what is implemented in `eigs` in MATLAB, which is based on the ARPACK package.

Odds and ends

Like Lanczos, the Arnoldi method converges most quickly to eigenvalues at the outside of the spectrum. As with the power method, we can find interior eigenvalues using Arnoldi or Lanczos iterations by applying a *shift-invert* transform that moves the desired eigenvalues to the exterior of the spectrum. This is actually done in MATLAB's sparse eigensolver `eigs` (which is based on the implicitly restarted Lanczos/Arnoldi code ARPACK). There is a catch, though: direct sparse LU factorization of a shifted matrix may be very slow.

The symmetric Lanczos and Arnoldi procedures are far from the only methods of computing eigenvalues of large, sparse matrices. In addition to

block versions of both algorithms, there are methods based on nonsymmetric Lanczos, and other methods (such as Jacobi-Davidson) that avoid Krylov subspaces entirely. What these methods share in common is that they build up subspaces (Krylov or others) in which eigenvector approximations are sought, and usually find a "good" approximation to an eigenpair by solving some reduced problem over the subspace (e.g. by finding Ritz pairs via a block Rayleigh quotient).

Iterative solvers for linear systems

The idea of drawing approximations from a Krylov subspace applies to problems other than eigenvalue problems as well. Many of the most popular iterative methods for solving large linear systems involve extracting approximate solutions from a sequence of Krylov subspaces. For example, suppose A is symmetric and positive definite and $Q_m \in \mathbb{R}^{n\times m}$ is a basis for a m-dimensional Krylov subspace generated by A and b. Then one method of approximating solutions to $Ax = b$ is to find $\hat{x}_m = Q_m y_m$ such that $Q_m^T(A\hat{x}_m - b) = 0$:

$$Q_m^T A Q_m y = T_m y = Q_m^T b.$$

This is one way to derive the *method of conjugate gradients*, which we will describe in more detail shortly. We could also construct solutions that satisfy different approximation conditions; for example, we could choose $\hat{x}_m$ so that $\|A\hat{x}_m - b\|_2$ was minimal over all admissible choices, which would lead us to an algorithm called MINRES. In each case, we use a two-part strategy: build Krylov subspaces that we believe will eventually contain good approximations, and a method to extract good approximate solutions from the space.

Krylov subspace methods are not the only iterative methods available. Earlier in the semester, we already saw one idea for iteratively solving linear systems $Ax = b$. The iteration

$$x_{k+1} = x_k + \hat{A}^{-1}(b - Ax_k),$$

converges to the solution x, provided $\hat{A}^{-1}$ is some "good enough" approximation to A^{-1}. This is a simple *stationary* iterative method. In general, stationary methods involve a constant-coefficient vector recurrence

$$Mx_{k+1} = Nx_k + b$$

where $A = M - N$ and it is "easy" to solve with M. Equivalently, we could write

$$x_{k+1} = x_k + M^{-1}(b - Ax_k)$$

The choice of M and N is known as a *splitting* of A. The iteration converges if the *spectral radius* of $M^{-1}N$ (i.e. the largest eigenvalue magnitude) is less than one. A weaker sufficient condition is that $\|M^{-1}N\| < 1$ in some consistent norm.

Now, consider the first few iterates of a stationary method (from $x_0 = 0$):

$$\begin{aligned} x_1 &= M^{-1}b, \\ x_2 &= x_1 + (x_1 - M^{-1}Ax_1), \\ x_3 &= x_2 + (x_1 - M^{-1}Ax_2). \end{aligned}$$

In general, x_k lies in $\mathcal{K}_k(M^{-1}A, M^{-1}b)$! Just as we expect Lanczos and Arnoldi procedures to better approximate eigenvalues than single-vector power iteration, we might expect a procedure that considers the *entire* Krylov subspace $\mathcal{K}_k(M^{-1}A, M^{-1}b)$ to produce better approximate linear system solutions than a single-vector stationary method. However, just as the choice of M (the splitting) is critical to the convergence of a stationary method, it is also critical to the speedy convergence of the related Krylov subspace methods. In the context of Krylov subspace methods, we call M a *preconditioner*, which is introduced by replacing the initial problem $Ax = b$ with an equivalent problem $M^{-1}Ax = M^{-1}b$.

Standard choices of splittings include taking M to be the diagonal of A (Jacobi iteration), or the lower triangle of A (Gauss-Seidel iteration). The Jacobi iteration corresponds to computing the jth element of x_{k+1} to satisfy the jth equation in the system $Ax = b$ provided all the other elements are given in x_k; and the Gauss-Seidel iteration corresponds to the same computation, but using updated values for the first $j - 1$ components when computing the jth component of x_{k-1}. There are block versions of each of these algorithms, as well as more sophisticated variants such as successive overrelaxation (SOR).

Corresponding to the Jacobi, Gauss-Seidel, and SOR iterative methods are Jacobi, Gauss-Seidel, and SOR preconditioners. Other general classes of preconditioners involve sparse approximate factorizations or inverses. Descriptions of common preconditioned Krylov subspace methods are given in Templates for the Solution of Linear Systems by Barrett et al (free online). Often, though, the best known preconditioners are application-specific.