Week 4: Wednesday, Feb 15

A summary

From Monday, you should have learned:

1. Gaussian elimination can be seen as the computation of a matrix factorization $PA = LU$, where L is a unit lower triangular matrix L whose entries are the multipliers used in the elimination process; U is an upper triangular matrix; and P is a permutation matrix corresponding to row re-ordering during partial pivoting.

2. Solving a linear system by Gaussian elimination consists of two steps: factoring the matrix (which costs $O(n^3)$) and solving triangular systems with forward and backward substitution (which costs $O(n^2)$).

3. Most of the entries in a *sparse* matrix are zero. We can represent a sparse matrix compactly by only storing the location and values of the nonzero entries. Gaussian elimination on sparse matrices *sometimes* yields sparse factors, but the order of elimination matters. The MATLAB call

   [L,U,P,Q] = **lu**(A);

   factors a sparse matrix A as $PAQ = LU$, where P, L, and U are as before, and the permutation matrix Q is automatically computed in order to try to keep L and U sparse.

Today, we'll look at

1. Condition numbers and some basic error analysis for linear systems.

2. *Cholesky* factorization for symmetric, positive definite matrices.

3. How Cholesky factorization actually gets implemented.

Partial pivoting

I only said a little last time about the role of the permutation matrix P in the factorization. The reason that P is there is to help control the size of

the elements in L. For example, consider what happens when we factor the following matrix without pivoting:

$$A = \begin{bmatrix} \epsilon & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \epsilon^{-1} & 1 \end{bmatrix} \begin{bmatrix} \epsilon & 1 \\ 0 & 1-\epsilon^{-1} \end{bmatrix}.$$

If we round u_{22} to $-\epsilon^{-1}$, then we have

$$\begin{bmatrix} 1 & 0 \\ \epsilon^{-1} & 1 \end{bmatrix} \begin{bmatrix} \epsilon & 1 \\ 0 & -\epsilon^{-1} \end{bmatrix} = \begin{bmatrix} \epsilon & 1 \\ 1 & 0 \end{bmatrix} \neq A;$$

that is, a rounding error in the (huge) u_{22} entry causes a complete loss of information about the a_{22} component.

In this example, the l_{21} and u_{22} terms are both huge. Why does this matter? When L and U have huge entries and A does not, computing the product LU must inevitably involve huge cancellation effects, and we have already seen the danger of cancellation in previous lectures. The *partial pivoting* strategy usually used with Gaussian elimination permutes the rows of A so that the multipliers at each step (the coefficients of L) are at most one in magnitude. Even with this control on the elements of L, it is still possible that there might be "pivot growth": that is, elements of U might grow much larger than those in A. But while it is possible to construct test problems for which pivot growth is exponential, in practice such cases almost never happen.

Alas, even when GEPP works well, it can produce answers with large relative errors. In some sense, though, the fault lies not in our algorithms, but in our problems. In order to make this statement precise, we need to return to a theme from the first week of classes, and discuss condition numbers.

Warm-up: Error in matrix multiplication

Suppose $\hat{y} = (A+E)x$ is an approximation to $y = Ax$. What is the error in using $\hat{y}$ to approximate y? We can write an equation for the absolute error:

$$\hat{y} - y = Ex,$$

and using norms, we have

$$\|\hat{y} - y\| \leq \|E\|\|x\|.$$

This is all well and good, but we would really like an expression involving *relative* errors. To get such an expression, it's helpful to play around with norms a little more. Assuming A is invertible, we have

$$\|x\| = \|A^{-1}y\| \leq \|A^{-1}\|\|y\|,$$

so that

$$\frac{\|\hat{y} - y\|}{\|y\|} \leq \|A\|\|A^{-1}\|\frac{\|E\|}{\|A\|}.$$

That is, the quantity

$$\kappa(A) = \|A\|\|A^{-1}\|$$

serves as a *condition number* that relates the size of relative error in the computed result $\hat{y}$ to the size of relative error in the matrix A. Note that this condition number is a function of the *problem formulation*, and does not depend on the way that we implement matrix multiplication.

It is a straightforward (if tedious) exercise in rounding error analysis to show that if we compute $y = Ax$ in the usual way in floating point arithmetic, the computed result $\hat{y}$ will actually satisfy $\hat{y} = (A + E)x$, where $|E_{ij}| \lesssim n\epsilon_{\text{mach}}|A_{ij}|$. That is, $\hat{y}$ is the *exact* result for a slightly perturbed problem. The perturbation E is called a *backward error*. For the matrix norms we have discussed, this element-wise inequality implies the norm inequality $\|E\|/\|A\| \leq n\epsilon$. Thus, the relative error in matrix multiplication is bounded by

$$\frac{\|\hat{y} - y\|}{\|y\|} \leq \kappa(A) \cdot n\epsilon.$$

Since the numerical computation always has a small backward error, we say the algorithm is *backward stable* (or sometimes just *stable*). If the problem is additionally *well-conditioned* (so that $\kappa(A)$ is small), then the *forward relative error* $\|\hat{y} - y\|/\|y\|$ will be small. But if the condition number is large, the forward error may still be big.

From multiplication to linear solves

Now suppose that instead of computing $y = Ax$, we want to solve $Ax = b$. How sensitive is this problem to changes in A? We know already how to differentiate A^{-1} with respect to changes in A; using this knowledge, we can

write a first-order sensitivity formula relating small changes δA in the system matrix to small changes δx in the solution:

$$\delta x \approx -A^{-1}(\delta A)A^{-1}b = -A^{-1}(\delta A)x.$$

Taking norms gives

$$\|\delta x\| \lesssim \|A^{-1}\|\|\delta A\|\|x\|,$$

which we can rearrange to get

$$\frac{\|\delta x\|}{\|x\|} \lesssim \kappa(A)\frac{\|\delta A\|}{\|A\|}.$$

That is, the condition number $\kappa(A) = \|A\|\|A^{-1}\|$ once again relates relative error in the matrix to relative error in the result. Another very useful result is that

$$\frac{\|\hat{x} - x\|}{\|x\|} \lesssim \kappa(A)\frac{\|r\|}{\|b\|},$$

where $r = b - A\hat{x}$ is the *residual error*, or the extent to which the approximate solution $\hat{x}$ fails to satisfy the equations.

Gaussian elimination with partial pivoting is almost always backward stable in practice. There are some artificial examples where "pivot growth" breaks backward stability, but this never seems to occur in practice; and if it does occur, one can cheaply evaluate the relative residual in order to evaluate the solution. What this means in practice is that solving linear systems with Gaussian elimination with partial pivoting almost always results in a small relative residual (on the order of some modestly growing function in n times ϵ_{mach}, for example). However, a small relative residual only translates to a small relative error if the condition number is also not too big!

Cholesky factorization

For matrices that are symmetric and positive definite, the *Cholesky factorization*

$$A = LL^T$$

is an attractive alternative to Gaussian elimination. Here, the *Cholesky factor* L is a lower triangular matrix; by convention, the diagonal of L is chosen to be

positive. Sometimes, the Cholesky factorization is written in the equivalent form

$$A = R^T R$$

where R is upper triangular; this is the convention used by default in MATLAB. One way to see this factorization is as a generalization of the posive square root of a positive real number[1]

The Cholesky factorization is useful for solving linear systems, among other things. Cholesky factors also show up in statistical applications, such as sampling a multivariate normal with given covariance; and the existence of a (nonsingular) Cholesky factor is equivalent to A being positive definite, so Cholesky factorization is sometimes also used to check for positive definiteness. Even if we're only interested in linear systems, the Cholesky factorization has a very attractive feature compared to Gaussian elimination: it can be stably computed without pivoting. Because pivoting is sort of a pain, I'm going to leave the discussion of the algorithmic details of Gaussian elimination to the book, but I will walk through some ideas behind Cholesky factorization.

Let's start with the Cholesky factorization of a 2-by-2 matrix:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} \\ 0 & l_{22} \end{bmatrix}.$$

We can write this matrix equation as three scalar equations, which we can easily use to solve for the Cholesky factor

$$\begin{aligned} a_{11} &= l_{11}^2 & l_{11} &= \sqrt{a_{11}} \\ a_{12} &= l_{21} l_{11} & l_{21} &= a_{12}/l_{11} \\ a_{22} &= l_{22}^2 + l_{21}^2 & l_{22} &= \sqrt{a_{22} - l_{21}^2}. \end{aligned}$$

This picture actually generalizes. Now suppose we write down a *block* matrix formula:

$$\begin{bmatrix} a_{11} & a_{21}^T \\ a_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 \\ l_{21} & L_{22} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21}^T \\ 0 & L_{22} \end{bmatrix}.$$

[1] It's worth noting that the matrix square root of an SPD matrix A is actually a symmetric positive definite matrix B such that $A = B^2$. So while the Cholesky factor is *a* generalization of square roots to matrices, it is not the generalization that gets called "the matrix square root."

Here, we're thinking of a_{11} and l_{11} as scalars, a_{21} and l_{21} as column vectors, and A_{22} and L_{22} as matrices. Working out the multiplication, we again have three equations:

$$\begin{aligned} a_{11} &= l_{11}^2 & l_{11} &= \sqrt{a_{11}} \\ a_{21} &= l_{21} l_{11} & l_{21} &= a_{21} l_{11}^{-1} \\ A_{22} &= L_{22} L_{22}^T + l_{21} l_{21}^T & L_{22} L_{22}^T &= A_{22} - l_{21} l_{21}^T. \end{aligned}$$

We can compute the first column of the Cholesky factor by the first two of these equations, and the remaining equation tells us how to express the rest of L as the Cholesky factor for a smaller matrix. Here's the idea in MATLAB:

```
function L = lec08chol(A)

  n = length(A);
  L = zeros(n);

  for j = 1:n

      % Compute column j of L
      L(j,j) = sqrt(A(j,j));
      L(j+1:n,j) = A(j+1:n,j)/L(j,j);

      % Update the trailing submatrix (a "Schur complement")
      A(j+1:n,j+1:n) = A(j+1:n,j+1:n)-L(j+1:n,j)*L(j+1:n,j)';

  end
```

Actually, MATLAB uses an even more sophisticated algorithm based on a block factorization

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{12} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} L_{11} & L_{21} \\ 0 & L_{22} \end{bmatrix}.$$

The L_{11} part of the factor is the Cholesky factorization of A_{11}, which is computed by a small Cholesky factorization routine; the block $L_{21} = A_{21} L_{11}^{-1}$ is computed by triangular solves; and then L_{22} is computed by a block factorization of the Schur complement $A_{22} - L_{21} L_{21}^T$. This organization turns out to be very useful for writing *cache-efficient code* that is able to do a lot of work on a small part of the matrix before moving on to other parts of the computation.

Problems to ponder

1. Suppose you were given P, L, and U such that $PA = LU$. How would you solve $A^T x = b$?

2. Suppose $A = LU$. How could you compute $\det(A)$ efficiently using the L and U factors[2]?

3. Show that the product of two unit lower triangular matrices is again unit lower triangular.

4. I claimed that if A has a nonsingular Cholesky factor, then A is SPD. Why is that so?

5. Suppose
$$A = \begin{bmatrix} \epsilon & 1 \\ 1 & 1 \end{bmatrix}$$
What is the one-norm condition number $\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1$?

6. I claimed in class that
$$\frac{\|\hat{x} - x\|}{\|x\|} \leq \kappa(A) \frac{\|r\|}{\|b\|}.$$
Using the formula $r = A\hat{x} - b = A(\hat{x} - x)$ and properties of norms, argue why this must be true.

7. What is the Cholesky factor of A?
$$A = \begin{bmatrix} 4 & 4 & 2 \\ 4 & 20 & 34 \\ 2 & 34 & 74 \end{bmatrix}$$
What is the determinant?

[2] When I taught multivariable calculus, I actually started off with this method for computing determinants. It has a nice interpretation if you think about an elementary operation in Gaussian elimination as a *shear transformation*, which preserves volumes.

8. Write an $O(n)$ code to compute the Cholesky factor of an SPD tridiagonal matrix given the diagonal entries $a_1, \ldots, a_n$ and off-diagonal entries $b_1, b_2, \ldots, b_{n-1}$:

$$A = \begin{bmatrix} a_1 & b_1 & & & \\ b_1 & a_2 & b_2 & & \\ & b_2 & a_3 & b_3 & \\ & & \ddots & \ddots & \ddots \\ & & & b_{n-1} & a_n \end{bmatrix}$$

9. *Harder.* In order to test whether or not a matrix A is singular, one sometimes uses a *bordered linear system*. If $A \in \mathbb{R}^{n \times n}$, we choose $b, c \in \mathbb{R}^n$ and $d \in \mathbb{R}$ at random and try to solve the equation

$$\begin{bmatrix} A & b \\ c^T & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

   If the extended matrix is singular, then A almost certainly has a null space of at least dimension two; otherwise, with high probability, $y = 0$ iff A is singular. Why does this make sense?