Factorizations and Special Forms

We have seen that one important application of the Gaussian elimination algorithm is factorizations. Given an n by n matrix A, the Gauss elimination algorithm allows us to factor that matrix into the form

$$A = P^t LU$$

The only problem with this is that Gauss elimination is a relatively inefficient algorithm. To perform Gauss elimination on an n by n matrix we have to do on the order of n^3 arithmetic operations. For very large n this is simply too inefficient.

A common theme in computing is that if a general algorithm is too inefficient we can often find more efficient algorithms that apply in special cases. As we will see below, there are more efficient methods to factor matrices that can be applied in some important special cases.

Another special case that occurs in some situations is that if we can place certain special conditions on the matrix A then we can also demand that the matrices L and U in the factorization above also take special forms.

Positive Definite Matrices

An important class of matrices that plays a role in many applications is the class of symmetric positive definite matrices.

Definition An n by n matrix A is *positive definite* if it is symmetric (that is, $A^t = A$) and if

$$\mathbf{x}^t A\, \mathbf{x} > 0$$

for all n dimensional vectors $\mathbf{x} \neq 0$.

One problem with this definition is that it is difficult to apply in practice. There are a number alternative ways to characterize positive definite matrices. Here are two of them.

Alternative One A *leading principal submatrix* of a matrix A is a matrix of the form

$$A_k = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,k} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k,1} & a_{k,2} & \cdots & a_{k,k} \end{bmatrix}$$

A matrix A is positive definite if and only if each of its leading principal submatrices has a positive determinant.

Alternative Two The symmetric matrix A is positive definite if and only if Gaussian elimination without row interchanges can be performed on the linear system $A\,\mathbf{x} = \mathbf{b}$ with all pivot elements positive.

Other alternative characterizations of positive definite matrices are possible. To see these, please take a course in linear algebra.

Cholesky Decomposition

Because symmetric, positive definite matrices have some special properties, we might expect that something special may happen when we try to factor them. The following theorem gives that special property.

Theorem A symmetric positive definite matrix A can be factored into LL^t, where L is a lower triangular matrix.

Note that the matrix L in the theorem is only required to be lower triangular. In particular, unlike the L matrix in LU decomposition, this matrix is not required to have 1s on its diagonal. In general, the diagonal entries of this L will not be 1, as we will see below.

An interesting thing about this decomposition is that we can develop the algorithm to compute L by brute force. We begin by assuming that there is a lower triangular L such that $\mathrm{A} = LL^t$. We then try to determine by brute force what the elements of that L matrix might look like.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix} = \begin{bmatrix} l_{1,1} & 0 & \cdots & 0 \\ l_{2,1} & l_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n,1} & l_{n,2} & \cdots & l_{n,n} \end{bmatrix} \begin{bmatrix} l_{1,1} & l_{2,1} & \cdots & l_{n,1} \\ 0 & l_{2,2} & \cdots & l_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & l_{n,n} \end{bmatrix}$$

Here we can take advantage of the fact that many of the entries on the right are 0. Here is what happens when we start multiplying out the product on the right and equating terms from the right to elements of A.

$$a_{1,1} = \left(l_{1,1}\right)^2$$

$$a_{1,2} = a_{2,1} = l_{1,1}\, l_{2,1}$$

$$a_{2,2} = \left(l_{2,1}\right)^2 + \left(l_{2,2}\right)^2$$

$$a_{1,3} = a_{3,1} = l_{1,1}\, l_{3,1}$$

$$a_{2,3} = a_{3,2} = l_{2,1}\, l_{3,1} + l_{2,2}\, l_{3,2}$$

$$a_{3,3} = \left(l_{3,1}\right)^2 + \left(l_{3,2}\right)^2 + \left(l_{3,3}\right)^2$$

After a few rounds of this we discover two interesting facts. The first is that these equations that determine the entries $l_{i,j}$ are easy to unravel. Starting from the first equation above we have that

$$l_{1,1} = \sqrt{a_{1,1}}$$

$$l_{2,1} = \frac{a_{1,2}}{l_{1,1}}$$

$$l_{2,2} = \sqrt{a_{2,2} - \left(l_{2,1}\right)^2}$$

$$l_{3,1} = \frac{a_{3,1}}{l_{1,1}}$$

$$l_{3,2} = \frac{a_{3,2} - l_{2,1}\, l_{3,1}}{l_{2,2}}$$

$$l_{3,3} = \sqrt{a_{3,3} - \left(l_{3,1}\right)^2 - \left(l_{3,2}\right)^2}$$

The second thing we realize after a few iterations of this is that the formulas for the entries in L follow some simple, predictable patterns. These patterns allow us to write down general formulas for the entries of L and turn this into an algorithm, known as the *Cholesky algorithm*.

$$l_{i,i} = \sqrt{a_{1,1} - \sum_{k=1}^{i-1} \left(l_{i,k}\right)^2}$$

$$l_{j,i} = \frac{a_{j,i} - \sum_{k=1}^{i-1} l_{j,k}\, l_{i,k}}{l_{i,i}}$$

Note that this algorithm is unfortunately still an $O(n^3)$ algorithm, although it does turn out to be a relatively simple and efficient $O(n^3)$ algorithm which is easier to implement than, say, LU decomposition.

Tridiagonal Matrices and Crout Factorization

Another even more special kind of matrix is the tridiagonal matrix. A matrix is *tridiagonal* if it has nonzero entries only on its principal diagonal and the diagonals immediately above and below the principal diagonal.

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & 0 & 0 & \cdots & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & 0 & \cdots & 0 \\ 0 & a_{3,2} & a_{3,3} & a_{3,4} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ 0 & \cdots & \cdots & 0 & a_{n,n-1} & a_{n,n} \end{bmatrix}$$

Because this matrix has so many 0 entries it may well be possible to give a simpler algorithm for, say, computing an LU factorization.

Once again, we can use a brute force argument to discover the correct algorithm. We start by assuming that the tridiagonal A can be factored into a lower triangular L and an upper triangular U with 1s on its principal diagonal. A little thought suffices to convince ourselves that L will only have two diagonals of nonzero entries, and that U will have only one diagonal of nonzero entries immediately above its principal diagonal of 1s.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & 0 & \cdots & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & 0 \\ 0 & a_{3,2} & a_{3,3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \cdots & a_{n,n-1} & a_{n,n} \end{bmatrix} = \begin{bmatrix} l_{1,1} & 0 & 0 & \cdots & 0 \\ l_{2,1} & l_{2,2} & 0 & \cdots & 0 \\ 0 & l_{3,2} & l_{3,3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \cdots & l_{n,n-1} & a_{n,n} \end{bmatrix} \begin{bmatrix} 1 & u_{1,2} & 0 & \cdots & 0 \\ 0 & 1 & u_{2,3} & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \cdots & 0 & 1 \end{bmatrix}$$

Once again, we start forming products and looking for patterns.

$$a_{1,1} = l_{1,1}$$

$$a_{1,2} = l_{1,1}\, u_{1,2}$$

$$a_{2,1} = l_{2,1}$$

$$a_{2,2} = l_{2,1}\, u_{1,2} + l_{2,2}$$

$$a_{2,3} = l_{2,2}\, u_{2,3}$$

$$a_{3,2} = l_{3,2}$$

$$a_{3,3} = l_{3,2}\, u_{2,3} + l_{3,3}$$

$$a_{3,4} = l_{3,3}\, u_{3,4}$$

These equations can be unraveled quite easily.

$$l_{1,1} = a_{1,1}$$

$$u_{1,2} = \frac{a_{1,2}}{l_{1,1}}$$

$$l_{2,1} = a_{2,1}$$

$$l_{2,2} = a_{2,2} - l_{2,1}\, u_{1,2}$$

$$u_{2,3} = \frac{a_{2,3}}{l_{2,2}}$$

$$l_{3,2} = a_{3,2}$$

$$l_{3,3} = a_{3,3} - l_{3,2}\, u_{2,3}$$

$$u_{3,4} = \frac{a_{3,4}}{l_{3,3}}$$

The pattern now is quite clear and simple.

$$u_{i,i} = 1$$

$$l_{1,1} = a_{1,1}$$

$$u_{1,2} = \frac{a_{1,2}}{l_{1,1}}$$

$$l_{i,i\text{-}1} = a_{i,i\text{-}1}$$

$$l_{i,i} = a_{i,i} - l_{i,i\text{-}1}\, u_{i\text{-}1,i}$$

$$u_{i,i+1} = \frac{a_{i,i+1}}{l_{i,i}}$$

This algorithm is known as *Crout factorization*.