A Condensed Summary of Matrix Decompositions We've Covered

Chris Tiee

February 3, 2016

In effort to lessen the bewilderment factor due to the great variety of matrix factorizations you've been learning, here's a summary of what we've covered so far (sort of the condensed first few weeks of the course). It will suggest, approximately, when and where each factorization is useful, and the dos and don'ts of matrix computation.

1 The Setup

There are basically two fundamental operations in linear algebra:

1. **Matrix multiplication** (including the special case of multiplying matrix by a vector, which is usually going to be a column, namely, finding b in $Ax = b$, given x).
   - This has a flop count of $2mnr$ for $m \times n$ and $n \times r$ matrices.
   - For the special case of a matrix-vector multiplication ($r = 1$) it is $2mn$, and in the case that A is square, $2n^2$.
   - For the special case of banded or sparse matrices, the flop count is $2ns$ where s is the bandwidth or simply the max number of nonzero elements per row.
2. **Solving a linear system** $Ax = b$: given b, find x. This is theoretically equivalent to calculating $A^{-1}b = x$, but as I've repeated many times over, you should not ever actually try to explicitly calculate A^{-1} (see the section at the end on "evil" operations).
   - This is theoretically equivalent to calculating $A^{-1}b = x$, but as I've repeated many times over, you should *not* ever actually try to explicitly calculate A^{-1} (see the section at the end on "evil" operations).
   - Most of the computational effort in solving linear systems is in computing the relevant matrix decompositions, described next. The factorization methods are centered around reducing things to a triangular form, for which forward and/or back substitution can be done, each of which has a flop count of n^2.
   - Sparse versions generally have a better flop count only in the case the matrix is banded. (Banded triangular: flop count is $2ns$) The problem with general sparse matrices (few nonzero elements per row, but no restriction of the distance from the diagonal), is that the decompositions produce *fill-ins*, namely zero elements that become nonzero. There are a number of algorithms that scramble things around to minimize this, but *we're skipping them* in favor of what we *really* do with large sparse systems, and will occupy us for a good couple of weeks: iterative methods.

2 Matrix Decompositions

These are the matrix decompositions we've studied so far:

1. **Cholesky Factorization**: $A = R^T R$, with R upper triangular, Uniqueness is enforced by making R have positive diagonal entries.

   - It only applies to square symmetric and positive-definite (SPD) matrices, namely A such that $x^T Ax > 0$ for $x \neq 0$. This situation occurs more often than one might think, due to applications where A is interpreted as some kind of energy (read: 1/2 something squared).
   - SPD matrices also arise even in more theoretical contexts where we're dealing with non-symmetric matrices, such as in least squares: given any rectangular A, of full rank, $A^T A$ is SPD (and thus has a Cholesky factorization). This Cholesky factor is useful in other decompositions as well.
   - You solve $Ax = b$ by first solving
     $$R^T y = b$$
     for y by forward substitution, and then solving
     $$Rx = y$$
     for x by back substitution. We write, for this process, $x = R \setminus R^T \setminus b$ (notation borrowed from MATLAB).
   - The flop count is about $\frac{1}{3}n^3$.
   - This method is very numerically stable.

2. **LU Decomposition:** $A = LU$ with L lower triangular and U upper triangular. Uniqueness is enforced by requiring L to have diagonal elements equal to 1.

   - This is the matrix computation version of row reduction. In fact, the coefficients of L are the multipliers you use in row reduction. Just like Cholesky, we factor it into a lower triangular–upper triangular pair. Except, this time, they don't have to be related to one another. And, A need not be symmetric or positive-definite (but it does have to be square and invertible).
   - You solve $Ax = b$ by first solving
     $$Ly = b$$
     for y by forward substitution, and then solving
     $$Ux = y$$
     for x by back substitution. We write, for this process, $x = U \setminus L \setminus b$.
   - The flop count of this is $\frac{2}{3}n^3$, twice as much as Cholesky when it can be applied.
   - This method is *numerically unstable*: in practice, the LU decomposition is supplemented with row exchanges, called pivoting. The key is to always choose, as the next row, something whose first coefficient is largest in magnitude. We didn't cover this in class, but you have to know this in practice. Matlab's `lu` command automatically takes care of this.

- This completely (and with pivoting, reliably) replaces matrix inversion as a practical tool (see next section on "evil" operations)

3. **(Condensed) QR Decomposition:** $A = QR$, where A can be a "skinny" rectangular matrix, Q is isometric, and R is upper triangular.

   - This can be computed using the Gram-Schmidt orthogonalization procedure: start off with A as a set of column vectors; then the orthonormal columns produced from the procedure gives the matrix Q. Saving the multipliers (a lot like how row reduction gives LU) gives you R.
   - The flop count is $\frac{4}{3}n^3$ for a square matrix.
   - This is numerically unstable. In practice, the error accumulates enough to make the "orthogonal" vectors not truly orthogonal: $\|Q^TQ - I\|_2$ gets farther and farther away from 0 as more vectors are computed.
   - This can be corrected by taking the columns of Q as a new A, and applying Gram-Schmidt again. That doubles the operations count to $\frac{8}{3}n^3$ and is not recommended for QR—instead, do the noncondensed using reflectors, and throw out the last $n-m$ columns of Q there.
   - Nevertheless, this is conceptually important, and tells you what the QR factorization really is.
   - This can be used to solve least squares problems: $Rx = Q^Tb$ yields the least squares solution to $Ax = b$. If square, this solves $Ax = b$ directly, but having a flop count of $\frac{4}{3}n^3$, twice that of LU, in turn twice that of Cholesky, we should only use QR when we have to, namely if $n \neq m$.

4. **Noncondensed QR Decomposition:** $A = QR$, where A can be a "skinny" rectangular matrix, Q is a square orthogonal matrix, and R is rectangular and upper triangular.

   - This is computed using Householder reflections, whose technique is to apply an orthogonal transformation that simultaneously zeroes out a bunch of elements. We did not cover this in class (and you won't be expected to know it for the exam, only that "this what you do").
   - This method is numerically stable.
   - The flop count is $2nm^2 - \frac{2}{3}m^3$ which for n close to m is $\frac{4}{3}n^3$, twice that of LU.
   - It is also possible to do this using rotators, but that increases the operations count and doesn't guarantee uniqueness, because it may lead to a situation where you must have a negative entry on the diagonal (bonus question: prove it).
   - You can use this decomposition to solve least squares problems (but this is done effectively by throwing out the last $n-m$ columns of Q and the bottom part of R to get the condensed version).

5. **The Singular Value Decomposition**, $A = U\Sigma V^T$.

   - This is the One Matrix Decomposition to Rule Them All.
   - Expensive to compute, but by far the most informative, providing a wealth of information about rank, range basis, domain basis, orthogonal complements, etc.

- Will solve the stubborn cases, namely, the least squares problem for rank-deficient matrix.

Expect this list to be updated as the quarter progresses! Also if you have any questions or discover a mistake in the above, let me know right away.

3 The Following Are Evil and Should Not Be Used

- **Matrix inversion**. Famously, **if you remember nothing else in this class, remember that you never have to calculate the inverse matrix A^{-1} explicitly**. For the curious, however, an efficient general matrix inversion method using the methods we've learned is: find the LU decomposition, and use forward and back substitution on each column of an identity matrix: find $(A^{-1})_{:i} = U \setminus L \setminus e_i$ where e_i is the ith standard basis element (or: $Ly = e_i$ and $Ux = y$): x is then the ith column. This has a flop count of $\frac{2}{3}n^3$, and is basically dominated by computing L and U. But with L and U in hand, this takes care of solving any $Ax = b$ you could ever want, even for "mass production" (if you compute and save A^{-1}, often stated as the one and only reason you'd want A^{-1}, and then use it to solve for a million x's given a million different b's, then since the flop count for each multiplication is $2n^2$, it is exactly the same effort to use forward and back sub, each with op count n^2, on each b). As we shall soon see in the next few weeks, sometimes even an LU decomposition is computationally infeasible.

- **Determinants** (I apologize for problem 1.7.10 with all its determinants). Don't use Cramer's Rule for solving linear systems for anything larger than a 2×2 system. Don't even bother to use it to check if a matrix is invertible (simply use LU factorization until a failure due to a zero diagonal element). Determinants have a super-exponential running time $O(n!)$ which is far, far worse than anything else we've studied (including matrix inversion). Determinants were at one time regarded as panacea for solution of linear equations, simply because "Oh look! We've reduced a whole matrix to a single number!" and it has literally taken *centuries* for it to be declared finally unfit for general linear-algebraic pedagogy. There *are* situations in which they are useful theoretically and conceptually (namely, concepts involving volumes, something that comes up a lot in my research, actually), but matrix computations is not one of them.