Matrix Decompositions

Paul Johnson

December 2012

1 Background

In order to understand these notes, you should have some experience with matrix algebra. Perhaps not an entire semester or year of matrix algebra is necessary, but comfort with columns, rows, and their multiplication, is assumed.

I assume you know about

- "inner products" of vectors.
- matrix multiplication (which is actually conducted by calculating the inner products of the rows and columns of the matrices).
- the transpose of a vector converts a row into a column (and vice versa)
- the transpose of a matrix is a "reflection" of that matrix on its main diagonal.

Other than that, I try to define the terms as I go.

Inverse and Identity

Recall that an inverse of X is matrix X^{-1} such that

$$X\,X^{-1} = I \tag{1}$$

I , the "identity matrix", is a matrix of 1's and 0's,

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

If you could grab the first row of X, and the first column of X^{-1}, the inner product of row and column would be 1. And the product of that first row with any other column of X^{-1} will be 0.

Disregard some Elementary Linear Algebra

In statistics, we often need to calculate the inverse of X. We do that in order to "solve" a system or "calculate" some statistic.

In elementary matrix algebra, we see a matrix A and vectors x and b,

$$Ax = b \tag{3}$$

solved by the introduction of an inverse matrix A^{-1}, a matrix such that $A^{-1}A = I$.

$$\begin{aligned} A^{-1}Ax &= A^{-1}b \\ Ix &= A^{-1}b \\ x &= A^{-1}b \end{aligned}$$

It is almost instinctual to think that, if we need to calculate x, we multiply $A^{-1}b$.

In theoretical treatments of statistics, that way of thinking is prevalent. We write about inverting matrices, but usually do not concern ourself with how that is actually done. And we don't worry about the problem of accuracy in digital computers. Or how long the calculations might take. Or whether or not we have memory to store the matrices that are required for calculations.

Here's the danger. Inverting a matrix might be time consuming AND inaccurate. Calculating the determinant of a matrix is famously inaccurate. Calculations to optimize model fits will generally be geared to stop looking for improvements when the changes in the results between steps is on the order of 0.0000001, or 10^{-7}. Numerical inaccuracy creates 'wobble' in parameter estimates, so that algorithms might seem to "converge" when they are not in fact finished, or they go on calculating trying to adapt to noise in the calculations when the results are actually good enough.

To avoid those problems, we use matrix decomposition tricks. Those tricks are the main topic I'm trying to cover in these notes. (But it seems I'm wasting more and more time writing down general matrix blather than I intended.)

Regression Application

In regression treatments, we are often told we need to calculate the inverse of a product, X^TX, where X is a data matrix of predictors.

The product X^TX is called the "normal matrix" because of the important role it plays in regression analysis and the solution of the "normal equations." In multiple regression, if the data generating process is $y = X\beta + e$, then the predictive model we work with is $X\hat{\beta}$, where $\hat{\beta}$ is the choice of an optimal estimate. The best estimate $\hat{\beta}$ is usually expressed as the vector that minimizes of squared errors $(y - X\hat{\beta})^T(y - X\hat{\beta})$. Using calculus, the first order conditions for the solution are a system of linear equations:

$$(X^TX)\hat{\beta} = X^Ty \tag{4}$$

or, if we could invert X^TX, we could get $\hat{\beta}$ "by itself" by:

$$(X^TX)^{-1}(X^TX)\hat{\beta} = (X^TX)^{-1}X^Ty \tag{5}$$

$$I\hat{\beta} = (X^TX)^{-1}X^Ty \tag{6}$$

$$\hat{\beta} = (X^TX)^{-1}X^Ty \tag{7}$$

On the surface, it seems, we need to invert X^TX to calculate $\hat{\beta}$.

If we are doing math with pencils and papers, we might be tempted to actually calculate the product, and then attempt to invert it. It is more tempting to write computer code that way. I've recently learned this is a mistake. I just got an email from an expert on optimization who looked at some code and warned me "don't calculate X^TX and then invert it. That's inaccurate. And unnecessary."

Inverting (X^TX) is ONE way to estimate $\hat{\beta}$, but it is not the most accurate way.

Regression software was written with inverse routines in the 1960s and 1970s, but progress in numerical linear algebra has led to a series of approaches that are more accurate and dependable. Instead, software uses methods that try to decompose the matrix in order to solve equation (4).

Before I forget, a few comments about X^TX. Suppose X is the data frame, which has one row per respondent and one column per parameter to be estimated, so its $N\,x\,p$.

- X^TX is a square matrix. It is smaller than X. It is $p\,x\,p$.

- $X^T X$ is a "cross products matrix", or a "variance-covariance matrix".
- $X^T X$ is a symmetric matrix (same values above and below the main diagonal).
- $X^T X$ is positive semi definite, meaning that for any vector w,

  $$w^T\, X^T X\, w \geq 0 \tag{8}$$

  That's like saying that the sum of squares of (Xw) is always positive, which seems "intuitive and obvious". However, in matrix algebra, lots of "intuitive and obvious" ideas are not correct, so this one bears pointing out.

  Background: call $w^T w$ the sum of squares of w. Note that Xw is a column vector. The sum of squares of Xw would be

  $$(Xw)^T Xw$$

  $$w^T X^T Xw \tag{9}$$

  Hence, $X^T X$ is positive semi-definite.
- Although $X^T X$ is smaller than X, we are afraid of calculating it. First, it is an expensive calculation. Multiplying $X^T X$ requires $N\,x\,N$ multiplications (each element of each row with each element of each column). Second, it is an inaccurate calculation, in the sense that the floating point approximation implicit in each digital calculation of each individual product is "congealed" into $X^T X$. Hence the advice "don't invert the normal matrix."

2 Misty Water-Colored Memories (of Linear Algebra)

To understand why decompositions are helpful, it is necessary to remember these facts from fundamental linear algebra.

2.0.1 Transpose of a product.

The transpose of a product is the product of the transposed the individual matrices, in reverse order.

$$(XY)^T = Y^T X^T \tag{10}$$

$$(X^T X)^T = X^T X \tag{11}$$

Thus, if a matrix can be decomposed into a product of smaller matrices, then we can "work" with that product. Another claim along those lines is the following.

2.0.2 Inverse of a product.

The inverse of a product is the product of the inverses of the individual matrices, in reverse order.

$$(XY)^{-1} = Y^{-1} X^{-1} \tag{12}$$

$$(XYZA)^{-1} = A^{-1} Z^{-1} Y^{-1} X^{-1} \tag{13}$$

2.0.3 Orthogonal Matrices.

Here's the big result: The inverse of an orthogonal matrix is its transpose. If you know that already, skip to the next section. Otherwise, labor on.

Here's what orthogonal means. Take any column. Calculate the inner product of that column with itself. (That's a sum of squares, incidentally). If the matrix is orthogonal, the result will be 1. Explicitly, for any column x_i , $x_i^T \cdot x_i = 1$. And if one takes two different columns, their inner product is 0. $x_i^T \cdot x_j = 0$ if $i \neq j$.

If that property holds for each column, then the definition of orthogonality boils down to the statement that

$$X^T X = I. \tag{14}$$

A reader should work out a little example to verify that. Given a column x_i, the product $x_i^T x_i = 1$, while the product of one column with another column, $x_i^T \cdot x_j$ is 0.

Orthogonal matrices have many special properties. Here's a big one.

Orthogonal matrices are very easy to invert. An inverse is as easy to get as a transpose.

Multiply $(X^T X)$ on the right by X^{-1}.

$$\begin{aligned} (X^T X)X^{-1} &= I\,X^{-1} \\ X^T(XX^{-1}) &= X^{-1} \qquad (15) \\ X^T &= X^{-1} \qquad (16) \end{aligned}$$

Surprise! The inverse of an orthogonal matrix is equal to its transpose.

There's no digital danger in calculating the inverse of an orthogonal matrix. The inverse is REALLY EASY to calculate–its just a transpose. It barely even counts as calculation, in my opinion. Its book-keeping.

2.0.4 Transpose of an Inverse equals Inverse of Transpose

I can't remember why I know this is true, but

$$\left(X^T\right)^{-1} = \left(X^{-1}\right)^T \tag{17}$$

2.0.5 Triangular Matrices.

A triangular matrix has 0's below or above the main diagonal.

$$\underset{upper\,triangular}{\begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ 0 & x_{22} & x_{23} & x_{24} & x_{25} \\ 0 & 0 & x_{33} & x_{34} & x_{35} \\ 0 & 0 & 0 & x_{44} & x_{45} \\ 0 & 0 & 0 & 0 & x_{55} \end{bmatrix}} \quad \underset{lower\,triangular}{\begin{bmatrix} x_{11} & 0 & 0 & 0 & 0 \\ x_{21} & x_{22} & 0 & 0 & 0 \\ x_{31} & x_{32} & x_{33} & 0 & 0 \\ x_{41} & x_{42} & x_{43} & x_{44} & 0 \\ x_{51} & x_{52} & x_{53} & x_{54} & x_{55} \end{bmatrix}} \tag{18}$$

Recall from the first week of matrix algebra that solving a linear system by Gaussian elimination requires us to calculate weighted row sums in order to arrive at an upper triangular matrix, and from there we "backsolve" to obtain the solution of a linear system.

The main benefit of triangular matrices is that they reduce the number of computations necessary to carry out matrix multiplication.

3 Decomposition: The Big idea

Inverting matrices is time consuming and inaccurate. We want to avoid it.

Decomposition approaches help. Perhaps we get results more quickly, but not always. But we do get them more accurately. And sometimes there is theoretical insight in the re-formulation of problems by matrix decomposition.

Suppose we could decompose X. That means re-write X as a product of separate matrices.

$$X = VDU^T \tag{19}$$

From the inverse of a product rule,

$$X^{-1} = U^{T^{-1}}D^{-1}V^{-1} \tag{20}$$

We could simplify it further if we had some "special knowledge" about V, D, and U. If the structures of U, D, and V were somehow simple and manageable, then we would be avoiding a lot of computation.

There are MANY different types of matrix decomposition. It can almost make your head spin.

4 SVD: The Singular Value Decomposition

Mandel, John. (Feb. 1982). "Use of The Singular Value Decomposition in Regression Analysis." The American Statistician 36(1): 15-24.

The SVD is on my mind this week because I've just spent about 30 hours writing code to optimize the calculation of the SVD to try to accelerate an R program. The SVD can play an important role in the calculation of ordinary inverses, but it is truly vital in calculation of estimates for principal components analysis as well as the "generalized" (Moore-Penrose) inverse for non-square or singular matrices.

Suppose X is an $m\,x\,n$ matrix. $X = VDU^T$ is a "singular value decomposition of X." There's a theorem that shows that such a decomposition always exists.

Those individual pieces have magic properties.

1. D is $m\,x\,n$. But it has a very simple structure. It has a diagonal matrix "tucked" into its top, and the bottom is padded with 0's. It

$$D = \begin{bmatrix} \lambda_1 & 0 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 & 0 \\ 0 & 0 & \lambda_3 & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & \lambda_n \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{21}$$

Sometimes, the last values of λ are zero, so we may have the right columns of D filled with 0's.

If D is square–because X is square–look how easy it would be to get the inverse:

$$D^{-1} = \begin{bmatrix} 1/\lambda_1 & 0 & 0 & 0 & 0 \\ 0 & 1/\lambda_2 & 0 & 0 & 0 \\ 0 & 0 & 1/\lambda_3 & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & 1/\lambda_n \end{bmatrix} \tag{22}$$

The eager reader will anticipate here, noticing the usage of the symbol λ_m. That's a hint that these values are familiar, they are eigenvalues, and they are all positive (or 0) if X is positive semidefinite.

2. V is square, $n\,x\,n$. The columns of V are orthogonal with one another. That means for a column i, $v_i^T v_i = 1$ but otherwise the product of two columns is 0. The columns of V are the eigenvectors of X^TX.

3. U is square, $m\,x\,m$ and orthogonal. It is made up of the eigenvectors that correspond to the values $(\lambda_1, \lambda_2, \ldots, \lambda_r)$. The columns of U are eigenvectors of XX^T.

The SVD is something of a "Swiss army knife" of matrix algebra. We can use it to decompose a square matrix, one for which we need an inverse, X^{-1}. However, its special power is that it can be used to calculate a generalized inverse of a rectangular, but not square, matrix. A generalized inverse X^+ has the property that $X = (X^+X)\,X$. It works like an inverse, then. This has various uses in statistical modeling.

Here's one way to think of the SVD. The 3 matrices multiplied together are equal to X, but we can group them into two sets to obtain some insight

$$X = (VDU^T) = (VD)U^T \, or \, V(DU^T) \tag{23}$$

This insight helps me to understand the statistical method called principal components, for example. Consider the first two terms together, VD, as a new, better version of X. VD has orthogonal (uncorrelated) columns. The matrix U^T is a weighting transformation that is applied to VD in order to generate X.

Principal components analysis is built around the idea that we need to understand the column-centered values of the X matrix, so the SVD in principal components is actually the SVD of $X - \mu$:

$$column\, centered\, X = X - \mu = (VD)U^T \tag{24}$$

I'm grouping together VD to try to help the reader see something important. Since the columns of V are orthogonal–they are "uncorrelated" in the statistical sense. D is just a simple weighting factor. So the VD is the "good" "uncorrelated" information from the data set. The SVD distills the information from the centered X matrix, setting aside the uncorrelated part VD and then taking note that to reproduce $X - \mu$, then we multiply by U^T.

If you remember that matrix multiplication is done column-by-column, then it may help to remember that U^T is a set of columns. So we can individually re-produce each column of X by multiplying the matrix VD by a column of U^T.

The calculation so far adds some insight for me, but it does not really "simplify" anything. Yet. Now suppose that the matrix D has some really small values on the main diagonal. The values are organized from high to low (top-left to bottom-right).

$$D = \begin{bmatrix} \lambda_1 & & & & \\ & \lambda_2 & & & \\ & & \lambda_3 & & \\ & & & nearly\, 0 & \\ & & & & nearly\, 0 \end{bmatrix} \tag{25}$$

If the bottom-right values are 0, or so small that they can be disregarded, then we might just "erase" them, literally set them equal to 0. After that, when we calculate

$$(VD)U^T \tag{26}$$

we are effectively throwing away the bottom rows of U^T by giving them zero weight. If we proceed through the matrix D, setting more and more of the diagonal values to 0, then we are trying to re-produce X by relying on less-and-less information. That's what principal components analysis is all about.

5 QR Decomposition

Although the SVD always works, it is also always computationally intensive. R does not use SVD for ordinary regression, it instead uses the QR decomposition. The SVD is only needed for cases in which the columns of X are linearly dependent (collinear).

Until very recently, regression textbooks would just plop out the answer

$$\hat{\beta} = (X^TX)^{-1}X^Ty \tag{27}$$

and assume the reader (or a statistical program) would implement calculations to get the right answer. I never thought about it very much, until I started reading the code for R's function lm.fit and I realized there's no matrix inversion going on in there. Its a QR decomposition. On the other hand, for very serious cases, say situations in which we think $X^T X$ is nearly singular, then we find the SVD is actually put to use, as in the MASS package's code for ridge regression.

The use of the QR algorithm is not a secret in the R documentation, of course, but I simply ignored it (as I expect most "users" do). However, when things go wrong in statistical calculations, then we often have to remember how these calculations are done.

This is the point at which I would refer the reader to Simon Woods's excellent book, Generalized Additive Models, my favorite stats book. It has an absolutely beautiful explanation of how the QR decomposition is put to use in estimating linear regression models.

Suppose X is $m\,x\,n$. The QR decomposition is as follows

$$X = Q\left[\begin{array}{c} R \\ 0 \end{array}\right] \tag{28}$$

Q is square, R is upper triangular. And we are encouraged to note that there is a big block of 0's in the bottom right side.

This is the "full sized" version of QR. Q is an orthogonal matrix that is $m\,x\,m$ in size. The thing on the right is reminiscent of the center matrix in the SVD. Its an upper griangular R in the top, padded by rows and rows of 0's on the bottom. Unlike the SVD, where the top part was diagonal, now the top part is upper triangular. And, unlike Q, which is a large $m\,x\,m$, R is only n wide. The bottom part of the stack, $\left[\begin{array}{c} R \\ 0 \end{array}\right]$, is $m-n$ rows of 0's:

$$\left[\begin{array}{c} R \\ 0 \end{array}\right] = \left[\begin{array}{ccccc} r_{11} & r_{12} & r_{13} & r_{14} & r_{1n} \\ 0 & r_{22} & r_{23} & r_{24} & r_{2n} \\ 0 & 0 & r_{33} & r_{34} & r_{3n} \\ 0 & 0 & 0 & \ddots & r_{(m-1)n} \\ 0 & 0 & 0 & 0 & r_{mm} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right] \tag{29}$$

I called that the "full sized" version of QR because we often don't need all that information. The fact that the bottom rows of the right part are all zeros can simplify the whole thing quite a bit.

Since the multiplication on the right is like this

$$X = \left[\begin{array}{ccccc} q_{11} & q_{12} & q_{14} & & q_{1m} \\ q_{21} & & & & \\ & & & & \\ & & \ddots & & \\ & & & & \\ & & & & \\ q_{m1} & & & & q_{mm} \end{array}\right]\left[\begin{array}{ccccc} r_{11} & r_{12} & r_{13} & r_{14} & r_{1n} \\ 0 & r_{22} & r_{23} & r_{24} & r_{2n} \\ 0 & 0 & r_{33} & r_{34} & r_{3n} \\ 0 & 0 & 0 & \ddots & r_{(n-1)n} \\ 0 & 0 & 0 & 0 & r_{nn} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right] \tag{30}$$

we see readily that the last columns of Q don't matter. In the multiplication process, those rows of 0's on the bottom of R end up "erasing" the last $m-n$ columns of the full sized Q.

As a result, we might as well throw away those rows of $0'$s and the $m-n$ columns of Q. So the more "petite" version (Q_f) would give

$$X = Q_f R \tag{31}$$

$$X = \left[\begin{array}{ccccc} q_{11} & q_{12} & q_{13} & & q_{1n} \\ q_{21} & q_{22} & q_{23} & & q_{2n} \\ q_{31} & q_{32} & & & \\ q_{41} & & & \ddots & \\ & \ddots & & & \\ & & & & \\ q_{m1} & & & & q_{mn} \end{array}\right]\left[\begin{array}{ccccc} r_{11} & r_{12} & r_{13} & r_{14} & r_{1n} \\ 0 & r_{22} & r_{23} & r_{24} & r_{2n} \\ 0 & 0 & r_{33} & r_{34} & r_{3n} \\ 0 & 0 & 0 & \ddots & r_{(n-1)n} \\ 0 & 0 & 0 & 0 & r_{nn} \end{array}\right] \tag{32}$$

When the software reports Q, it will often not report the full sized thing, but rather the smaller $m\,x\,n$ Q_f matrix. (In R, for example, run qr() and then put the result through qr.Q() to extract Q. The result will be $m\,x\,n$, unless the additional argument complete=TRUE is supplied to qr.Q()).

If X is a square matrix, of course, then there's no smaller Q to deal with. The Q matrix will be $m\,x\,m$ and R will be upper triangular.

How does that help us in a regression analysis? Well, the "brute force" instinct in me is to stomp through this like an elephant. We want to calculate

$$\hat{\beta} = (X^T X)^{-1} X^T y \tag{33}$$

So replace each X by the petite $Q_f R$.

$$\hat{\beta} = ((Q_f R)^T (Q_f R))^{-1} (Q_f R)^T y \tag{34}$$

If we use the rules for inverses and transposes mentioned above, we can algebraically reduce that:

$$\begin{aligned}
\hat{\beta} \;\; = \;\; & (R^T Q_f^T Q_f R))^{-1} (Q_f R)^T y & (35)\\
& (R^T R)^{-1} R^T Q_f^T y & (36)\\
& R^{-1} R^{T^{-1}} R^T Q_f^T y & (37)\\
& R^{-1} Q_f^T y & (38)
\end{aligned}$$

On pages 15-17, Woods has a more elegant, albeit less obvious, derivation.

In the ordinary math-based treatment of regression, we find the variance of $\hat{\beta}$ is

$$\widehat{Var(\hat{\beta})} = \sigma_e^2 (X^T X)^{-1} \tag{39}$$

The QR algorithm-based estimator is

$$\widehat{Var(\hat{\beta})} = \sigma_e^2 R^{-1} R^{-T} \tag{40}$$

The QR algorithm has been on my mind this week, even though I was working to accelerate code that uses an SVD. An algorithm to use the QR decomposition to accelerate the calculation of generalized inverses was presented in this article:

V. N. Katsikis, D. Pappas, Fast computing of the Moore-Penrose inverse ## matrix, Electronic Journal of Linear Algebra, 17(2008), 637-650.

My R implementation of that function tries to use the fast KPginv method to get a Moore-Penrose inverse, but if that fails, it uses the guaranteed-to-work svd() based calculation.

```
mpinv2 <- function(X, tol = sqrt(.Machine$double.eps)) {
    KPginv <- function(X){
        Xdim <- dim(X)
        if (Xdim[1] > Xdim[2]){
            C <- crossprod(X)
            ginv <- solve(C, t(X))
        } else {
            C <- tcrossprod(X)
            G <- solve(C, X)
            ginv <- t(G)
        }
        ginv
    }

    tryCatch (KPginv(X), error = function(err) {
        print("use original mpinv (consider MASS::ginv instead)")
        s <- svd(X)
        e <- s$d
        e[e > tol] <- 1/e[e > tol]
        s$v %*% diag(e, nrow = length(e)) %*% t(s$u)
    })
}
```

6 What's the Point?

When a problem is small, then we can write careless computer code and get right answers more or less quickly. We can invert matrices, without worrying too much.

However, when problems grow in size, we need to exert more care when writing computer programs. I recently learned that one of my colleagues started a calculation in September that did not finish until December. That may be necessary, but I'd have to inspect the code to be sure. He said it took a long time to "invert some really big matrices" and my spidey-senses were tingling with the hint of danger.