QR factorisation and pseudoinverse of rank-deficient matrices

Peter Moylan

Glendale, NSW

peter@pmoylan.org http://www.pmoylan.org

April 2016

ABSTRACT: Standard algorithms for QR decomposition assume that the matrix in question has full rank. We present an alternative that can, without pivoting, handle the rank-deficient case, and that produces an *R* matrix that has linearly independent rows. One application of this approach is the efficient calculation of pseudoinverses.

1. Introduction

The QR factorisation of a (real or complex) $m \times n$ matrix A is defined to be the decomposition

$$A = QR$$

where Q has the property $Q^*Q = I$, and R is a (not necessarily square) upper triangular matrix. Here, the superscript star indicates the adjoint (complex conjugate transpose) of a matrix. The size of the matrices is not specified, except to the extent that the matrix product must make sense. In practice there are three important ways of defining a QR factorisation.

In the traditional definition Q is an $m \times m$ unitary matrix ($Q^*Q = QQ^* = I$), and R is an $m \times n$ upper triangular matrix. This is called the full QR decomposition. It turns out, however, that some of the most useful applications of QR factorisation are when $m \gg n$. In such cases the traditional definition leads to unreasonably large matrices.

If $m > n$, the traditional factorisation has the form

$$A = [Q_1 \quad Q_2]\begin{bmatrix} R_1 \\ 0 \end{bmatrix} = Q_1 R_1$$

where Q_1 still has the property $Q_1^*Q_1 = I$, and R_1 is still upper triangular. Now Q_1 is $m \times n$ and R_1 is $n \times n$. In this case we choose to say that the pair (Q_1,R_1) is still a QR factorisation of A, one that is computationally more convenient because the matrices are smaller. Because Q_1 is not square it is technically not unitary, but for our purposes that does not matter. In the literature this is called the *thin* QR factorisation or the *reduced* QR factorisation. It is unique if A has full column rank and we restrict attention to those solutions where the diagonal elements of R_1 are positive. Obviously Q_2 is not unique, but that does not matter because we have no interest in computing Q_2.

The goal of the present paper is to present an algorithm that can also handle rank-deficient matrices. Specifically, we will find a factorisation where the number of rows of R is equal to the rank of A. Clearly, no smaller decomposition is possible, so we will call this the *minimal* or *fully reduced* QR factorisation.

Our approach is not the same as in rank-revealing QR factorisations, see e.g. [Ming96]. Those use a modified full or thin QR factorisation, and then look at the R matrix to look for a small or zero subblock. The goal in the present paper is not to generate the negligible part of R at all. This give computational savings, and also ensures that the rows of R are linearly independent.

One motivation for using a QR factorisation is that triangular matrices are easy to invert. This does not appear to help in the rank-deficient case, but for some applications it turns out that a pseudoinverse can be used instead of an inverse. For this reason, we show at the end of the paper how to computer a pseudoinverse, cheaply, using the minimal QR factorisation as an intermediate step.

2. Existence and uniqueness of a minimal QR decomposition

In this section we prove the existence and uniqueness of a specified kind of *fully reduced* QR factorisation. In general a QR factorisation is not unique, but it becomes unique once we add some constraints.

Definition: An upper triangular matrix is in fully reduced row echelon form if it has no all-zero rows, if the first nonzero element in each row is real and positive, and if that first nonzero element lies strictly to the right of the first nonzero element of the row above it.

Clearly, such a matrix has linearly independent rows. In other words, the number of rows is equal to its rank. Note that the "no all-zero rows" condition implies that it also has at least as many columns as rows.

In what follows we will sometimes need the matrix A to be nonzero, so we should first dispose of the case $A = 0$. In that case the decomposition is non-unique, but one solution is

$$Q = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad\qquad R = \begin{bmatrix} 0 & 0 & \cdots & 0 \end{bmatrix}$$

This is, strictly speaking, not a minimal decomposition (since A has rank 0 and R has one row), but it is the best we can do.

For any nonzero A, we have the following result.

Theorem. Any nonzero real or complex matrix A has a unique decomposition $A = QR$ with the properties that R is in fully reduced row echelon form, and $Q^*Q = I$.

Proof. We proceed by induction on the number of columns of A. If A is nonzero and has only one column, the result is obvious by inspection. (R is then a 1×1 matrix.) Otherwise, let a_1 be the first column of A, so that

$$A = \begin{bmatrix} a_1 & A_2 \end{bmatrix}$$

If $a_1 = 0$, then by assumption $A_2 \neq 0$, and A_2 has fewer columns than A. By the inductive hypotheses, there exists a unique factorisation $A_2 = Q_2 R_2$ with Q_2 and R_2 in the required form. Then we have

$$A = [0 \quad Q_2 R_2] = Q_2 [0 \quad R_2]$$

This satisfies the requirements of the theorem statement.

If $a_1 \neq 0$, any QR factorisation, if it exists, necessarily has the form

$$A = [a_1 \quad A_2] = [q_1 \quad Q_2] \begin{bmatrix} r_1 & r_a \\ 0 & R_2 \end{bmatrix} = [q_1 r_1 \quad q_1 r_a + Q_2 R_2]$$

where q_1 is the first column of Q and r_1 is a scalar. The fact that $Q^* Q = I$ means that $q_1^* q_1 = 1$ and $q_1^* Q_2 = 0$ and $Q_2^* Q_2 = I$. This implies

$$r_1 = \sqrt{a_1^* a_1} \qquad\qquad q_1 = \frac{1}{r_1} a_1$$

and these values are unique because of the constraint that r_1 be real and positive. There are now two subcases to consider. For reasons that will become clear below, define

$$r_a = q_1^* A_2 \qquad\qquad A_{2new} = A_2 - q_1 r_a$$

If $A_{2new} = 0$ (which, it is easy to show, occurs iff A has rank 1), we have

$$A = [a_1 \quad A_2] = [q_1 r_1 \quad q_1 r_a] = q_1 [r_1 \quad r_a]$$

Observe that in this case the R matrix has a single row, which is trivially in row echelon form. This solution is unique, because any other QR factorisation would have more rows in R than the rank of A.

Finally, consider the general case where $a_1 \neq 0$ and $A_{2new} \neq 0$. By the inductive hypothesis, and the fact that A_{2new} has fewer columns than A, there exist unique matrices Q_2 and R_2 such that $A_{2new} = Q_2 R_2$, $Q_2^* Q_2 = I$, and R_2 is in fully reduced row echelon form. Define

$$Q = [q_1 \quad Q_2] \qquad\qquad R = \begin{bmatrix} r_1 & r_a \\ 0 & R_2 \end{bmatrix}$$

Then

$$QR = [q_1 \quad Q_2] \begin{bmatrix} r_1 & r_a \\ 0 & R_2 \end{bmatrix} = [q_1 r_1 \quad q_1 r_a + A_{2new}] = [a_1 \quad A_2] = A$$

Clearly R has the desired properties, so it only remains to be shown that $Q^* Q = I$. By construction, $q_1^* q_1 = 1$ and $Q_2^* Q_2 = I$, so only the cross product is in question. Observe that

$$q_1^* Q_2 R_2 = q_1^* A_2 - q_1^* q_1 r_a = q_1^* A_2 - r_a = 0$$

and $q_1^* Q_2 = 0$ follows because the rows of R_2 are linearly independent. □

3. A practical implementation

The proof in the last section is a constructive one, so to turn it into software we need only copy the steps of the proof. The result appears to be a recursive algorithm, but in practice no recursion is involved: we are simply stepping through the columns of A, at the same time filling in columns of Q and rows and columns of R, and modifying the remaining columns of A. The calculation is complete when we have finished processing either the last row or the last column of the modified A.

Pivoting can be added to the algorithm, if desired. The only change this requires is to find, at each step, the largest of the columns not yet processed, and if necessary to swap that with the current column. This, however, does mean that the factorisation now has the form $A\Pi = QR$ or equivalently $A = QR\Pi^T$, where Π is a permutation matrix. Depending on the application, this might not be desirable, because $R\Pi^T$ will not be a triangular matrix.

Note that the special case $A_{2new} = 0$ does not require any special handling, other than detecting that that matrix is zero. At that point in the calculation r_1 and r_a have already had their values assigned, so all that remains to be done is to exit from the calculation loop.

Let A be an $m \times n$ matrix. The fully reduced decomposition is most beneficial when $m \gg n$, but it is also valid for the case $m \leq n$. We start the calculation by implicitly assuming that R has $\min(m, n)$ rows, but we omit one or more rows each time we meet a special case ($a_1 = 0$ or $A_{2new} = 0$). This does imply discarding partial rows of R that we have set in previous steps, but no information is lost, because the discarded entries are merely some of the zero entries below the main diagonal.

An in-place calculation is possible, where each column of Q that we calculate replaces a column of A. If we are doing an in-place calculation, we need to watch out for one detail. In the case $a_1 = 0$, where we step to the next column of A without adding a column to Q, we might have to be prepared to shift the remainder of A one column left. Alternatively, and perhaps more efficiently, we just need to keep track of the fact that the current column of Q is not necessarily the same as the current column of A.

4. QR decomposition using Householder reflections

It is commonly stated that a QR decomposition via Householder reflections has better numerical stability than the modified Gram-Schmidt process already described. We are therefore motivated to see whether the Householder approach can be modified to produce a minimal decomposition.

As shown in Appendix 2, a Householder transformation H can act on a vector x to produce

$$Hx = \begin{bmatrix} \alpha \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

where α is a scalar. If x is the first column of an $m \times n$ matrix A, then we have

$$HA = \begin{bmatrix} \alpha & r_a \\ 0 & R_2 \end{bmatrix}$$

which is the first step towards producing an upper triangular matrix. Repeating this operation on smaller and smaller submatrices, we get

$$H_n H_{n-1} \dots H_2 H_1 A = R$$

and therefore

$$A = QR$$

where

$$Q = H_1^* H_2^* \dots H_n^*$$

Observe that Q is an $m \times m$ matrix. That means that, regardless of the rank of A, we have a full QR decomposition, which is not really acceptable if $m \gg n$. In this case, the last $m - n$ rows of R are zero, so we can get a thin QR decomposition by deleting the last $m - n$ columns of Q.

This, unfortunately, does not translate into deleting rows or columns of the Householder matrices H_i. We have no alternative but to retain an $m \times m$ matrix Q until the end of the calculation; only at the end can we discard the redundant columns. For a long thin matrix A, this can create serious storage problems.

But can we modify the Householder approach to get a minimal factorisation? Luckily, we can. Partway through the calculation, we have

$$A = Q_i R_i = Q_i \begin{bmatrix} R_1 & R_a \\ 0 & R_{22} \end{bmatrix}$$

Uniqueness of the solution for R means that R_1 and R_a are precisely the matrices obtained by our Section 2 calculations. That means that we can use the Section 2 insights to work out how to modify the Householder approach to produce a minimal decomposition.

Let the upper left corner of R_{22} be at row and column (r_0, c_0). In the full-rank case we have $r_0 = c_0$, but for our algorithm this is no longer true. From Section 2 we know that we have to deal with two special cases:

(a) If the first column of R_{22} is zero, then we should increment c_0 without incrementing r_0 for the next step.
(b) Otherwise, if the updated R_{22} is zero, then we have finished the calculation.

It turns out that we do not need to check for case (b). If the case (b) condition is satisfied, then on the following iterations we will immediately run into case (a), so the calculations will be terminated in any case. Whether we do this check is purely a matter of software efficiency, and does not affect the final answer.

So does the Householder approach give a more accurate answer? If running a test suite, we should check $A - QR$ (which should be zero), and $Q^*Q - I$ (which should be zero). Our tests on random matrices shows that the Householder approach gives better values of $Q^*Q - I$, but worse values of $A - QR$. In other words, the Householder approach is better at producing orthogonal columns of the Q matrix, but is not as good at producing a QR factorisation. If our goal is to produce a good R matrix, then the Householder approach is less desirable.

5. LQ decomposition

An LQ decomposition of a matrix A has the form

$$A = LQ$$

where L is lower triangular and Q has the property that $QQ^* = I$. As in the QR case, we can have full, thin, or fully reduced version. The details are exactly as in the QR case, except that we swap the labelling of rows and columns.

6. Pseudoinverses

A pseudoinverse calculation is typically done via a singular value decomposition [SVD], but this is computationally expensive. In this section we show that the minimal QR decomposition can be used as a computationally cheap way to compute the result. The pseudoinverse (Moore-Penrose inverse) $M^{\#}$ of a real or complex matrix M is defined uniquely by the properties [Pen55]:

- $M^{\#}M$ and $MM^{\#}$ are both Hermitian
- $MM^{\#}M = M$
- $M^{\#}MM^{\#} = M^{\#}$

The pseudoinverse arises in connection with solving equations like

$$Mx = b$$

The "solution" $x = M^{\#}b$ has the properties

1. If one or more exact solutions exist, then the given solution is the one of least norm.
2. If no exact solution exists, then the pseudoinverse "solution" is the one that comes closest to being a solution in that it minimises $\|Mx - b\|$.
3. If M is invertible then $M^{\#} = M^{-1}$.

The following properties can be confirmed by substitution into the above conditions for a pseudoinverse.

- The operation of taking the Hermitian (complex conjugate transpose) of a matrix commutes with the pseudoinverse operation. That is, $(M^*)^{\#} = (M^{\#})^*$.
- If M has full column rank, then $M^{\#} = (M^*M)^{-1}M^*$.
- If M has full row rank, then $M^{\#} = M^*(MM^*)^{-1}$.

7. Calculating the pseudoinverse

As always, we need to assume that we are dealing with nonzero matrices. The pseudoinverse of an all-zero matrix is equal to its transpose, but we have to separate this out as a special case.

Let $A = QR$, where $Q^*Q = I$. Then it is easily shown, directly from the definition of a pseudoinverse, that $A^{\#} = R^{\#}Q^*$. (There is no similar result for an arbitrary matrix product.) That leaves us with the problem of computing $R^{\#}$. Because of the way we do the decomposition, R has full row rank and is upper triangular.

If that R is square, it is nonsingular, so the pseudoinverse is equal to the inverse. Calculating the inverse of an upper triangular matrix is a well-known low-cost algorithm.

More generally, if A is $m \times n$ and has rank q, then R is $q \times n$. If $q < n$ then the "upper triangular" property turns out to be of no help, because the pseudoinverse of a non-square upper triangular matrix is usually not a triangular matrix. However, the full rank property means that

$$R^{\#} = R^*(RR^*)^{-1}$$

At first sight, this appears to complicate matters by requiring us to calculate a full matrix inverse. The matrix RR^* is Hermitian, and the usual recommended way of inverting an

Hermitian matrix is via the Cholesky decomposition [Cho]. A Cholesky decomposition works by factoring the matrix in a certain way. Below, we show a cheaper method.

Suppose we do a QR factorisation of the $n \times q$ matrix R^*. The result is

$$R^* = Q_1 R_1$$

where R_1 is now a *square* $q \times q$ upper triangular matrix, and Q_1 is $n \times q$, i.e. it has more rows than columns. Then

$$(RR^*)^{-1} = (R_1^* Q_1^* Q_1 R_1)^{-1} = (R_1^* R_1)^{-1} = R_1^{-1}(R_1^{-1})^*$$

$$R^\# = R^*(RR^*)^{-1} = Q_1 R_1 R_1^{-1}(R_1^{-1})^* = Q_1(R_1^{-1})^*$$

This is a significant gain, because now the only inversion we have to deal with is the inversion of a triangular matrix. The cost is that we have had to do a new QR factorisation, but this is still cheaper than doing a full matrix inversion.

The conclusion is that we can calculate the pseudoinverse of any arbitrary matrix using at most two QR factorisations and inverting a triangular matrix (which is easy). This appears to be computationally more efficient than, for example, using a singular value decomposition.

Computational accuracy can be improved by using pivoting in the course of the QR calculation. This is a minor change to the calculation, and results in finding a permutation matrix Π such that $A\Pi = QR$. With this change, we have $A = QR\Pi^T$ and $A^\# = \Pi R^\# Q^*$. That is, it is the same calculation plus a final permutation.

8. Relationship to singular value decomposition

It is not hard to see that the method just described is effectively a QR decomposition followed by an LQ decomposition, and in fact we can save one adjoint calculation by doing it that way. If we have a factorisation $A = PLQ$, where $P^*P = I$ and $QQ^* = I$ and L is lower triangular, then it is easy to show that $A^\# = Q^* L^{-1} P^*$.

If our goal is to compute a pseudoinverse then we need go no further, because the triangular matrix L is easy to invert. If, however, we choose to do a similar decomposition of L, we get a sequence of factorisations

$$A = P_i L_i Q_i$$

where each L_i is the result of doing a PLQ decomposition of L_{i-1}. It turns out that at each step L_i is "more diagonal" than its predecessor, in the sense that the off-diagonal elements are becoming smaller, so that eventually we converge towards a diagonal L_i.

This is, in fact, a known algorithm for finding a singular value decomposition, although not the most efficient [SVD]. We do, however, depart from the usual definition in the following way. If A is $m \times n$ and has rank q, then according to the conventional approach it has $\min(m, n)$ singular values, of which q are nonzero. Because we are using *minimal* QR and LQ decompositions, instead of the more usual thin decompositions, our L_i are nonsingular at every step. As a result, we calculate only the nonzero singular values. Of course, no information is lost, because we know that the missing ones are all zero. We can restore them, if this is really necessary, by adding zero rows and/or columns to L, and padding the P and Q matrices with extra blocks that, because they are multiplied by zero, have no effect on the product. For most applications this will not be necessary, because the positive singular values are the interesting ones.

9. Conclusions

It is possible, with a fairly simple change to the conventional Gram-Schmidt approach, to do a QR decomposition where the R matrix is guaranteed to have full row rank. One consequence of this is that we can calculate pseudoinverses far more efficiently than by using a singular value decomposition.

The software for the algorithms described here are available for download [Moy16].

AUTHOR'S NOTE: As a retired academic I have no access to on-line research journals, so cannot find out which parts of this paper, if any, are original. I gather that some universities let retired staff retain their library privileges, but that is not the case for my university. I apologise for the poor referencing.

References

[Cho] Cholesky Decomposition, https://en.wikipedia.org/wiki/Cholesky_decomposition

[Ming96] Ming Gu and Stanley C. Eisenstat, "Efficient algorithms for computing a strong rank-revealing QR factorisation, SIAM J Sci. Comput. 17 (4), 848-868, July 1996.

[Moy16] P.J. Moylan, Numerical Analysis Software, http://pmoylan.org/pages/m2/NumAnal.html

[Pen55] R. Penrose, "A generalized inverse for matrices", Mathematical Proceedings of the Cambridge Philosophical Society, 51, pp 406-413, 1955. doi:10.1017/S0305004100030401.

[SVD] Singular Value Decomposition, https://en.wikipedia.org/wiki/Singular_value_decomposition

Appendix 1: Inversion of triangular matrices

The following results are well-known, but are collected here for convenience.

For a partitioned upper triangular (and nonsingular) matrix, the equation $\begin{bmatrix} A & B \\ 0 & C \end{bmatrix}\begin{bmatrix} D & E \\ F & G \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix}$ expands to $\begin{bmatrix} AD + BF & AE + BG \\ CF & CG \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix}$, which implies $G = C^{-1}$ and $F = 0$. From this observation, there is an obvious inductive argument that shows that the inverse (if it exists) of an upper triangular matrix is itself upper triangular, and the diagonal elements of the inverse are the reciprocals of the corresponding diagonal elements of the original matrix.

Let R be an upper triangular matrix, and let S be its inverse. The property $SR = I$ expands out to

$$(SR)_{ij} = \sum_{k=1}^{N} S_{ik}R_{kj} = \delta_{ij}$$

and because we know that some of the terms are zero, this reduces to

$$\sum_{k=i}^{j} S_{ik}R_{kj} = \delta_{ij}$$

For row i, we already know that $S_{ii} = 1/R_{ii}$ and that $S_{ij} = 0$ for all $j < i$. For $j > i$,

$$S_{ij}R_{jj} + \sum_{k=i}^{j-1} S_{ik}R_{kj} = 0$$

or

$$S_{ij} = -\frac{1}{R_{jj}}\sum_{k=i}^{j-1} S_{ik}R_{kj}$$

If we move from left to right through the array, the right side of this equation involves only quantities we have already calculated.

An alternative approach, starting from the equation $RS = I$, leads to the formula

$$S_{ij} = -\frac{1}{R_{ii}}\sum_{k=i+1}^{j} R_{ik}S_{kj}$$

for $j > i$, which is appropriate if we want to move backwards through the rows. These two methods are of equal complexity and apparently equal accuracy, so either method should work equally well.

The results for a lower triangular matrix are similar. If L is a nonsingular lower triangular matrix and M is its inverse, then M is also lower triangular, with

$$M_{ii} = \frac{1}{L_{ii}}$$

and

$$M_{ij} = -\frac{1}{L_{ii}} \sum_{k=j}^{i-1} L_{ik} M_{kj}$$

for $i > j$. This is in a form suitable for working through the rows from first to last. There is a second formula which also works, but is less convenient because it requires working backwards through the columns.

Appendix 2: Householder transformations

This is again known material, but it is included here because most of the available references cover only the case of real matrices, and the extension to the complex case is not entirely obvious.

A Householder transformation on a matrix is multiplication on the left by a matrix

$$H = I - \gamma v v^*$$

where γ is a real or complex scalar and v is a unit-norm vector ($v^* v = 1$). It is easy to show that $H^* H = H H^* = I$ iff $\gamma = 1 + \beta$, where β is a real or complex number such that $|\beta| = 1$. In the real case the only possible choices are $\gamma = 0$ (meaning no change) or $\gamma = 2$. In the complex case the choice of γ is a little trickier.

The property $H^* H = I$ implies that $\|Hx\| = \|x\|$ for any vector x. This says that by suitable choice of H we can rotate or reflect the vector, but we cannot change its size. In the context of QR calculations, we are interested in producing the result $Hx = \alpha y$, where y is a unit vector. In matrix notation, that means a vector with 1 in its first position and zero entries elsewhere. The norm-preserving property implies that $|\alpha| = \|x\|$, which fixes the magnitude of the scalar α but not its phase. For our application there is no freedom of choice in the phase either. If we are using the Householder transformation to do a QR factorisation, then α must be real and nonnegative. This forces the choice $\alpha = \|x\|$.

Geometrical considerations suggest that we should choose $v = u/\|u\|$, where

$$u = x - \alpha y$$

Then

$$Hx = x - \gamma v v^* x = x - \frac{\gamma u^* x}{\|u\|^2} u = x - \frac{u^* x + \beta u^* x}{\|u\|^2} u$$

The result that we want is $Hx = \alpha y = x - u$, which requires that

$$u^* x + \beta u^* x = \|u\|^2$$

and a short calculation reduces this to

$$\beta = \frac{x^* u}{u^* x}$$

This is all we need to complete the calculation.