**Laboratorio # 6 La factorización QR**

**TABLE OF CONTENTS**

* [Introduction](http://www.fim.utp.ac.pa/Members/fernando.castillo/metodos-numericos/laboratorio-parte-2/laboratorio-6#INTRODUCTION)
* [Orthogonal Matrices](http://www.fim.utp.ac.pa/Members/fernando.castillo/metodos-numericos/laboratorio-parte-2/laboratorio-6#ORTHOGONAL_MATRICES)
* [The Gram-Schmidt Method](http://www.fim.utp.ac.pa/Members/fernando.castillo/metodos-numericos/laboratorio-parte-2/laboratorio-6#THE_GRAM_SCHMIDT_METHOD)
* [Gram-Schmidt Factorization](http://www.fim.utp.ac.pa/Members/fernando.castillo/metodos-numericos/laboratorio-parte-2/laboratorio-6#GRAM_SCHMIDT_FACTORIZATION)
* [The QR Factorization Process](http://www.fim.utp.ac.pa/Members/fernando.castillo/metodos-numericos/laboratorio-parte-2/laboratorio-6#THE_QR_FACTORIZATION_PROCESS)
* [Householder Matrices](http://www.fim.utp.ac.pa/Members/fernando.castillo/metodos-numericos/laboratorio-parte-2/laboratorio-6#HOUSEHOLDER_MATRICES)
* [Householder Factorization](http://www.fim.utp.ac.pa/Members/fernando.castillo/metodos-numericos/laboratorio-parte-2/laboratorio-6#HOUSEHOLDER_FACTORIZATION)
* [The QR Method for Linear Systems](http://www.fim.utp.ac.pa/Members/fernando.castillo/metodos-numericos/laboratorio-parte-2/laboratorio-6#QR_METHOD_FOR_LINEAR_SYSTEMS)
* [Loose Ends](http://www.fim.utp.ac.pa/Members/fernando.castillo/metodos-numericos/laboratorio-parte-2/laboratorio-6#LOOSE_ENDS)
* [ASSIGNMENT](http://www.fim.utp.ac.pa/Members/fernando.castillo/metodos-numericos/laboratorio-parte-2/laboratorio-6#ASSIGNMENT)
* [Comments](http://www.fim.utp.ac.pa/Members/fernando.castillo/metodos-numericos/laboratorio-parte-2/laboratorio-6#COMMENTS)

**Introduction**

We know we can use the **PLU** factorization to solve a linear system, ... provided that the system is square, and that it is nonsingular, and that it is not too badly conditioned. And in fact, this is usually the case.

However, if we want to handle problems with a bad condition number, or that are singular, or even rectangular, we are going to need to come up with a different approach. In this lab, we will look at two versions of the **QR** factorization:

**A = Q \* R**

where **Q** is an "orthogonal" matrix, and **R** is upper triangular.

We will see that this factorization can be used to solve the same problems that the **PLU** factorization handles, but can be extended to do several other tasks for which the **PLU** factorization is useless.

In situations where we are concerned about controlling roundoff error, the **QR** factorization can be more accurate than the **PLU** factorization, and is even a little easier to use when solving linear systems, since the inverse of **Q** is simply **QT**.

The **QR** factorization also gives us a way to solve rectangular linear systems, and is the key idea behind a method for reliably computing all the eigenvalues and eigenvectors of a matrix.

**Orthogonal Matrices**

**Definition**: An *orthogonal matrix* is a real matrix whose inverse is equal to its transpose.

By convention, an orthogonal matrix is usually denoted by the symbol **Q**. The definition of an orthogonal matrix immediately implies that:

**Q\*QT=QT\*Q=I**

From the definition of an orthogonal matrix, and from the fact that the **L2** vector norm of **x** can be defined by:

**||x||2 = sqrt ( xT \* x )**

and the fact that

**(A\*x)T=xT\*AT**

you should be able to deduce an amazing fact about orthogonal matrices, namely:

**||Q\*x||2 = ||x||2**

If I multiply **x** by **Q** and its **L2** norm doesn't change, then **Q\*x** must lie on the circle whose radius is **x**. In other words, the only thing that has happened is that **Q\*x** has spun around the origin by some angle. That means that an orthogonal matrix represents a *rotation* or *reflection*.

**Exercise**: copy the file *random\_orthogonal.m* from the web page. It computes a random orthogonal matrix of given dimension. Each time you call it, you should get a different matrix, so you can use this routine for experiments. Using your wits, and the experimental data, verify, contradict, or improve the following statements about a random orthogonal matrix **Q**:

1. the identity matrix is orthogonal; the matrix [0 1; 1 0] is orthogonal; the matrix [c s; -s c] is orthogonal, where **c** and **s** are the cosine and sine of some angle; any diagonal matrix is orthogonal.
2. an orthogonal matrix must be symmetric.
3. the product of two orthogonal matrices is also orthogonal.
4. the inverse of an orthogonal matrix is also orthogonal.
5. the **L2** vector norm of the first column of **Q** must be 1.
6. the vectors formed by the first and last rows of **Q** must be perpendicular.
7. the vectors formed by the first row and the second column of **Q** must be perpendicular.
8. every entry of **Q** must be between -1 and 1.
9. the determinant of **Q** is always 1.
10. the eigenvalues of **Q** are always +1 or -1.
11. the singular values of **Q** are always +1 or -1.
12. the **L1**, **L2** and **Linf** norms of **Q** are always 1.
13. If **||x||2=7**, then **||Q\*x||2** is \_\_\_\_\_
14. If **A** is a matrix, and **||A||2=7**, then **||Q\*A||2** is \_\_\_\_\_
15. the **L2** condition number of **Q** is \_\_\_\_\_
16. in any column of **Q**, at most 1 entry can be 1.0.
17. in any column of **Q**, at most 1 entry can be 0.0.

See comments #1 at the end of the lab.

**A Set of Problems**

Suppose we have **N** vectors **xi**, each of length **M**. Here are some common and related problems:

* Can we determine if the vectors are linearly independent?
* Can we determine the dimension of the space the vectors span?
* Can we construct an orthonormal basis for the space spanned by the vectors?
* Given a new vector **xN+1**, can we determine whether this vector lies in the space spanned by the other vectors?
* If a new vector lies in the space, what specific linear combination of the old vectors is equal to the new one?
* If we create an **M** by **N** matrix using the vectors as columns, what is the rank of the matrix?
* How can we solve a *rectangular* system of linear equations?

Since we actually plan to do these calculations *numerically* we also need to be concerned about how we deal with numerical error. For instance, in exact arithmetic, it's enough to say a matrix is not singular. In numerical calculations, we would like to be able to say that a particular matrix, while not singular, is "nearly singular".

**The Gram Schmidt Method**

The *Gram Schmidt method* can be thought of as a process which analyzes a set of vectors **X**, producing an "equivalent" (and possibly smaller) set of vectors **Q** which span the same space, have unit **L2** norm, and are pairwise orthogonal.

Note that if we can produce such a set of vectors, then we can easily answer many of the questions in our set of problems. In particular, the size of the set **Q** tells us whether the set **X** was linearly independent, and the dimension of the space spanned and the rank of a matrix constructed from the vectors. And of course, the vectors in **Q** are the orthonormal basis we were seeking. So you should believe that being able to compute the vectors **Q** is a valuable ability. In fact, the **QR** factorization can help us with all the problems on our list, primarily because it's possible to look at every number in the factorization and understand what that number is telling us about the linear system.

In words, the Gram-Schmidt process goes like this:

* Start with no vectors in **Q**;
* Consider **x**, the next vector in **X**;
* For every vector **q** already in **Q**, compute the component of **q** that is in **x** and subtract it from **x**;
* Compute the norm of (what's left of) **x**. If the norm is zero (or too small), discard **x**; otherwise, divide **x** by its norm and move it from **X** to **Q**.
* You now have an orthonormal set of vectors spanning the same space as **X**.

Here is a sketch of the Gram Schmidt process as an algorithm. Assume that **n** is the number of **x** vectors:

**nq = 0 for j = 1 to n for i = 1 to nq rij = qi' \* xj   
 xj = xj - qi \* rij end rjj = sqrt ( xj' \* xj )  
  
 if ( rjj ~= 0 )   
 nq = nq + 1  
 qnq = xj / rjj end end**

You should be able to match this algorithm to the previous verbal description. How is the **L2** norm of a vector being computed?

**M-file** - implement the Gram-Schmidt process in an M-file called *gram\_schmidt.m. Assume the vectors are stored as columns of a matrix called* ***X****. Your function should have the form:*

***function Q = gram\_schmidt ( X )***

*You can reference the vector formed by the* ***j****-th column of* ***X*** *by using the notation* ***X(:,j)****.*

***Exercise*** *- Test your Gram-Schmidt code using the following input:*

*X = ( 2 -1 0 )  
 ( -1 2 -1 )  
 ( 0 -1 2 )  
 ( 0 0 -1 )*

*If your code is working correctly, you should compute approximately:*

*Q = ( 0.89 0.36 0.20 )  
 ( -0.45 0.72 0.39 )  
 ( 0.00 -0.60 0.59 )  
 ( 0.00 0.00 -0.68 )*

*You should verify that the columns of* ***Q*** *have* ***L2*** *norm 1, and are pairwise orthogonal. What is an easy way to check this?*

*Explain the following statements about the matrix* ***Q*** *you just computed:*

* *The columns of* ***Q*** *have* ***L2*** *norm 1, and are pairwise orthogonal, so* ***Q*** *must be an orthogonal matrix, right?*
* *..and moreover,* ***QT\*Q = I****, true?*
* *..and yet, it is not the case that* ***Q\*QT = I****...*
* *...therefore,* ***Q*** *can't be an orthogonal matrix?*

*See comments #2 at the end of the lab.*

***Gram-Schmidt Factorization***

*We need to take a closer look at the Gram-Schmidt process. Recall how the process of Gauss elimination could actually be regarded as a process of factorization. This insight enabled us to solve many other problems. In the same way, the Gram-Schmidt process is actually carrying out a different factorization that will give us the key to other problems.*

*Just to keep our heads on straight, let me point out that we're about to stop thinking of* ***X*** *as a bunch of vectors, and instead regard it as a matrix. Since our easily-confused brain likes matrices to be called* ***A****, that's what we'll call our set of vectors from now on.*

*Now, in the Gram-Schmidt algorithm, the numbers that we called* ***rij*** *and* ***rjj****, which we computed, used, and discarded, actually record important information. They can be regarded as the nonzero elements of an upper triangular matrix* ***R****. The Gram-Schmidt process actually produces a factorization of the matrix* ***A*** *of the form:*

***A = Q \* R***

*Here, the matrix* ***Q*** *has the same* ***M*** *by* ***N*** *"shape" as* ***A****, so it's only square if* ***A*** *is. The matrix* ***R*** *will be square (****N*** *by* ***N****), and upper triangular.*

*Now that we're trying to produce a factorization of a matrix, we need to modify the Gram-Schmidt algorithm of the previous section. Every time we consider a vector* ***xj*** *at the beginning of a loop, we will now always produce a vector* ***qj*** *at the end of the loop. Hence, if* ***rjj****, the norm of vector* ***xj****, is zero, we will simply set* ***qj*** *to the zero vector.*

***M-file****: make a copy of your previous M-file and call it gs\_factor.m. Modify it to compute the* ***Q*** *and* ***R*** *factors of a rectangular matrix* ***A****. It should have the form*

***function [ Q, R ] = gs\_factor ( A )***

***Exercise****: when you think your code is correct, use it to compute the* ***QR*** *factorization of the Frank matrix of order 4. To verify your results, the following statements should be checked:*

1. *Is it true that* ***QT\*Q=I****?\_\_\_\_\_*
2. *Is it true that* ***Q\*QT=I****?\_\_\_\_\_*
3. *Is* ***Q*** *orthogonal in the strict sense?\_\_\_\_\_*
4. *Is the matrix* ***R*** *upper triangular?\_\_\_\_\_*
5. *Is it true that* ***A=Q\*R****?\_\_\_\_\_*

***Householder Matrices***

*Another approach to the* ***QR*** *factorization of a matrix proceeds through a series of partial factorizations* ***A=Qk\*Rk****, where the first* ***Q*** *is the identity matrix, and the first* ***R*** *is the matrix* ***A****. When we begin the* ***k****-th step of factorization, our factor* ***Rk-1*** *is only upper triangular in columns 1 to* ***k-1****. Our goal on the* ***k****-th step is to find a better factor* ***Rk*** *which is upper triangular through column* ***k****. If we can do this process* ***n-1*** *times, we're done.*

*Suppose, then, that we've partially factored the matrix* ***A****, up to column* ***k-1****. In other words, we have a factorization*

***A = Qk-1 \* Rk-1***

*for which the matrix* ***Qk-1*** *is orthogonal, but for which the matrix* ***Rk-1*** *is only upper triangular for the first* ***k-1*** *columns.*

*To proceed from our partial factorization, we're going to consider taking some orthogonal matrix* ***Hk****, and "inserting" it into the factorization as follows:*

***A = Qk-1 \* Rk-1 = Qk-1 \* HT \* H \* Rk-1***

*Then, if we define*

***Qk = Qk-1 \* HT  
Rk = H \* Rk-1***

*it will again be the case that:*

***A = Qk \* Rk***

*and we're guaranteed that* ***Qk*** *is still orthogonal. The interesting question is, if we pick the right* ***H****, can we "improve" the appearance of* ***Rk****, so that it is actually upper triangular all the way through column* ***k****.*

*In fact, we can do this by picking an appropriate Householder matrix. The formulas for a Householder matrix are a little complicated, and we won't go into them here. However, what you should understand is that, for the problem we've just described, we can determine a Householder matrix* ***H*** *with the property that* ***Rk=H\*Rk-1*** *is "slightly more upper triangular" than* ***Rk-1****.*

***M-file*** *- copy the file householder.m from the web page. This function accepts the name of a matrix* ***R*** *and the index of a column* ***k****, and returns a Householder matrix* ***H*** *such that* ***H\*R*** *has only zero entries below the diagonal in column* ***k****.*

***Exercise*** *- define the matrix* ***A*** *to be the Frank matrix of order 5. Compute* ***H1****, the Householder matrix that knocks out the subdiagonal entries in column 1 of* ***A****, and then compute* ***A1=H1****\*A. Does the result have the proper form?*

*Now let's compute* ***H2****, the Householder matrix that knocks out the subdiagonal entries in column 2 of* ***A****, and compute* ***A2=H2****\*A. This matrix should have subdiagonal zeros in column 2. You should be convinced that you can zero out any subdiagonal column you want.*

*Now let's try to zero out all the subdiagonal entries. Proceed as follows:*

*A = frank ( 5 )  
 Q = eye ( 5 )  
 R = A  
  
 H = householder ( R, 1 )  
 Q = Q \* H'  
 R = H \* R  
 Q \* R  
  
 H = householder ( R, 2 )  
 Q = Q \* H'  
 R = H \* R  
 Q \* R  
 ...*

*As you watch the calculation, you should see that the* ***R*** *matrix is gradually being zeroed out below the diagonal, and that the product of* ***Q*** *and* ***R*** *still equals* ***A****. Once we have completed the 4-th step, we're done.*

***Householder Factorization***

*For a rectangular* ***M*** *by* ***N*** *matrix* ***A****, the Householder* ***QR*** *factorization has the form*

***A = Q \* R***

*where the matrix* ***Q*** *is* ***M*** *by* ***M*** *(hence square and truly orthogonal) and the matrix* ***R*** *is* ***M*** *by* ***N****, and upper triangular (or upper trapezoidal if you want to be more accurate.)*

*If the matrix* ***A*** *is not square, then this definition is different from the Gram-Schmidt factorization we discussed before. The obvious difference is the shape of the factors. Here, it's the* ***Q*** *matrix that is square. The other difference, which you'll have to take on faith, is that the Householder factorization is generally more accurate, and easier to define compactly.*

*Householder QR Factorization Algorithm:*

1. *Q := I  
   R := A.*
2. *For* ***k*** *= 1 to* ***min(m-1,n)****:*
3. *Construct the Householder matrix* ***H*** *for column* ***k*** *.*
4. *Q := Q \* H',   
   R := H \* R.*
5. *next* ***k****.*

***M-file*** *- Make an M-file h\_factor.m. It should have the form*

***function [ Q, R ] = h\_factor ( A )***

*Use the routine householder.m that you copied earlier, in order to compute the* ***H*** *matrix that you need at each step.*

***Exercise*** *- Test your code by computing the* ***QR*** *factorization of the Hilbert matrix of order 4. You should get something like this:*

*Q = 0.84 -0.52 0.15 0.03 R = 1.19 0.67 0.47 0.37  
 0.42 0.44 -0.73 0.32 0 0.12 0.13 0.12  
 0.28 0.53 0.14 0.79 0 0 0.01 0.01  
 0.21 0.50 0.65 0.53 0 0 0 0.00*

*By the way, there is plenty of information in this factorization. To make the 4-th column of* ***A****, for instance we use 37 percent of column 1 of* ***Q****, 12 percent of column 2, 1 percent of column 3, and practically nothing of column 4. The fact the the diagonal entry is so small compared to the other entries in the 4-th column allows us to assign a sort of measure to how linearly dependent the 4-th column of* ***A*** *is on the previous 3 columns (since the space spanned by the first three columns of* ***A*** *is the same as that spanned by the first three columns of* ***Q****.*

***Exercise*** *- You may have written code that only works properly for a square matrix. In order to check that you have set things up to work for any rectangular matrix* ***A****, carry out the following tests:*

***A = rand ( 2, 4 )  
[Q,R] = h\_factor ( A )  
A2 = Q \* R;  
norm ( A - A2, inf )***

*and repeat the test for random matrices of sizes (3,4), (4,4), (5,4) and (6,4). If you get a large error on any of these computations, you'll need to go back and rethink your code.*

***The QR Method for Linear Systems***

*If we have computed the Householder* ***QR*** *factorization of a matrix without encountering any singularities, then it is easy to solve linear systems. We use the property of the* ***Q*** *factor that* ***QT\*Q=I****:*

***A \* x = b  
Q \* R \* x = b  
QT \* Q \* R \* x = QT \* b  
R \* x = QT \* b***

*so all we have to do is form the new right hand side* ***QT \* b*** *and then solve the upper triangular system. And that's easy because it's the same as the last step of our old* ***PLU*** *solution algorithm.*

***M-file*** *- Make a copy of your file plu\_solve.m, calling it h\_solve.m. It should have the form*

***function x = h\_solve ( Q, R, b )***

*Assume that the* ***QR*** *factors come from the h\_factor routine, so that, if* ***A*** *is rectangular, then so is* ***R****, while* ***Q*** *will always be square. Set up your code to compute* ***QT \* b*** *and then solve the upper triangular system* ***R \* x = QT \* b****. For now, don't worry too much about the possibility that the matrix is rectangular.*

***Exercise****: When you think your solver is working, test it out on a square system as follows:*

*n = 5*

*A = frank ( n )*

*x = [ 1 : n ]'*

*b = A \* x*

*[ Q, R ] = h\_factor ( A )*

*x2 = h\_solve ( Q, R, b )*

***Loose Ends***

*Instead of the Householder form of the factorization, we may use the Gram-Schmidt* ***QR*** *factorization. There is not much difference for the case where the system matrix is square. But if the system was rectangular, the matrix* ***Q*** *is rectangular, and what we do from there depends on whether the number of rows is greater or less than the number of columns.*

*For either method of factorization, a further complication occurs when the number of columns is greater than the number of rows. In this underdetermined situation, there are multiple solutions. Assuming we simply want one solution, any solution, the typical procedure is to set the extra degrees of freedom to zero.*

*We haven't had time here to discuss the issues of singularity. If the square matrix* ***A*** *is actually singular, what happens? We may get a very small diagonal* ***R*** *value, or possibly a zero value. We haven't planned on handling such a case. Similarly, in the rectangular matrix cases, it's often desirable to do what amounts to pivoting, in order to process the "best" columns of data first.*

***Assignment***

*You will need to rewrite your h\_solve routine so that it can handle a rectangular system. Actually, the routine only has to solve an upper triangular system. Determine the size* ***[M,N]*** *of the matrix* ***R****, and then do an upper triangular solve as though the matrix was of size* ***min(M,N)****.*

*Roughly speaking, you should expect that you can get some solution* ***x*** *with zero residual, whenever* ***M*** *is no greater than* ***N****, and you will usually get an answer, with a nonzero residual, whenever* ***M*** *is strictly bigger than* ***N****.*

***Assignment****: Given the system matrix* ***A*** *and right hand side* ***b****, use your* ***QR*** *factor and solve routines to compute a solution* ***x****, and the residual* ***r=A\*x-b****.*

*System #1:*

*A = 1 2 3 b = 6  
 4 5 6 15*

*System #2:*

*A = hilbert(16) b = A \* ones(16,1)*

*System #3:*

*A = 1.00 1.0 b = 1.98  
 2.05 -1.0 0.95  
 3.06 1.0 3.98  
 -1.02 2.0 0.92  
 4.08 -1.0 2.90*

*System #4:*

*A = 1 3 b = 18  
 5 2 25  
 4 -1 7*

*Fill out the following table. For system #2 only, don't list all 16 elements of the solution, just the maximum solution error.*

*System Solution ||A\*x-b||inf   
#1 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
#2 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
#3 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
#4 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*and then mail the results to me.*

***Comments***

*Comments #1 - properties of an orthogonal matrix?*

*Properties of an orthogonal matrix:*

1. *YES; YES; YES; NO.*
2. *NO, an orthogonal matrix is not generally symmetric.*
3. *YES, essentially because*

***(Q1\*Q2)T=QT2\*QT1*** *.*

*with a similar statement for inverses. The important thing to notice is that the multiplication order gets reversed. Using these ideas, you can prove the statement.*

1. *YES, essentially because the inverse of the transpose is the transpose of the inverse.*
2. *YES, the* ***L2*** *vector norm of the first column of* ***Q*** *must be 1. In fact, the* ***L2*** *norm of every column and every row must be 1.*
3. *YES, the vectors formed by the first and last rows of* ***Q*** *must be perpendicular, that is, their "inner product" or "dot product" must be zero, and in fact any pair of distinct rows or columns must have a zero dot product.*
4. *NO, row 1 does not have to be orthogonal to column 2. There is no guaranteed relationship between a row and a column of the matrix.*
5. *YES, every entry of* ***Q*** *must be between -1 and 1 because the* ***L2*** *norm of every column must be 1.*
6. *NO, the determinant of* ***Q*** *is not always 1. It is always +1 or -1.*
7. *NO, the eigenvalues of* ***Q*** *are not always +1 or -1. All eigenvalues have magnitude 1.*
8. *NO, in fact the singular values of* ***Q*** *are always +1.*
9. *NO, the* ***L2*** *norm of* ***Q*** *is always 1, but we can't say about the others.*
10. *For any* ***x****,* ***||Q\*x||2=||x||2****.*
11. *For any matrix* ***A****,* ***||Q\*A||2=||A||2****.*
12. *The* ***L2*** *condition number of* ***Q*** *is 1.*
13. *YES, in any column of* ***Q****, at most 1 entry can be 1.0.*
14. *NO, in any column of* ***Q****, up to n-1 entries can be 0. After all, the identity matrix is orthogonal.*

*Comments #2 - can a rectangular matrix be orthogonal?*

*A matrix* ***Q*** *is orthogonal if it is square and it is the case that* ***QT\*Q=Q\*QT=I****. From this second condition, we can conclude that an orthogonal matrix has columns of unit* ***L2*** *norm which are pairwise orthogonal.*

*Conversely, if a matrix has columns of unit* ***L2*** *norm which are pairwise orthogonal...it may loosely be called an orthogonal matrix, but it isn't, unless the matrix is square. If the matrix is* ***M*** *by* ***N****, with* ***M>N*** *(the usual case), then you will get* ***QT\*Q=I*** *but not* ***Q\*QT=I****!*

*We will carelessly call a rectangular matrix orthogonal. This is much more convenient than calling it a matrix "with orthonormal columns that are pairwise orthogonal". However, we have to remember that this is incorrect terminology, and that certain things true about an orthogonal matrix will not be true for rectangular "orthogonal" matrices.*