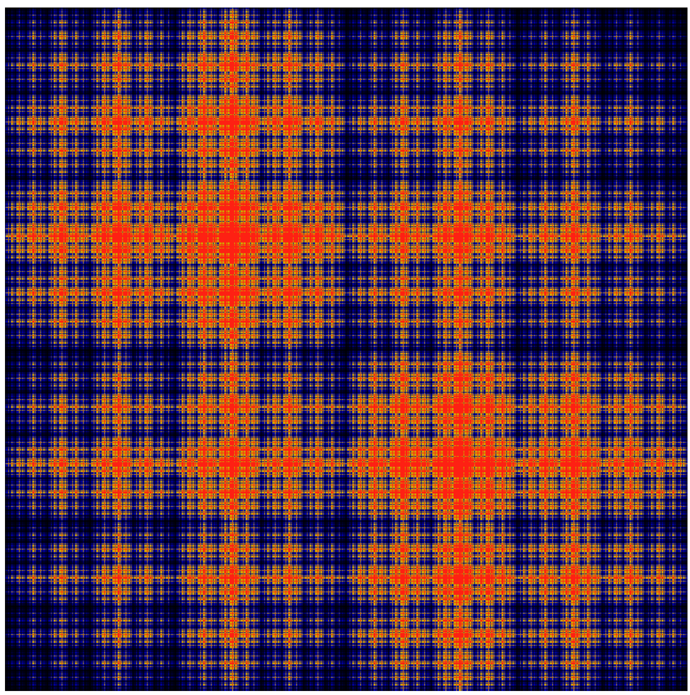
Applied Mathematics
and
Computing

Volume I



List of Contributors

J. Humpherys
Brigham Young University

J. Webb
Brigham Young University

R. Murray
Brigham Young University

J. West
University of Michigan

R. Grout
Brigham Young University

K. Finlinson
Brigham Young University

A. Zaitzeff
Brigham Young University

Preface

This lab manual is designed to accompany the textbook *Foundations of Applied Mathematics* by Dr. J. Humpherys.

Lab 11

Algorithms: QR Decomposition (Householder)

Lesson Objective: *Use orthogonal transformations to perform QR decomposition.*

Orthogonal transformations

Recall that a matrix Q is *unitary* if $Q^*Q = I$ or for real matrices, $Q^TQ = I$ (since the conjugate of a real number is itself). We like unitary transformations because they're very numerically stable. The number $\kappa(A) = \|A\| \left\|A^{-1}\right\|$ is called the *condition number* of A. We'll discuss condition number more in Lab **??**; for now, all you need to know is that if $\kappa(A)$ is small, then problems involving A are less susceptible to numerical errors. For induced matrix norms (which include most of the matrix norms we would ever care about), it holds that $\|Q\| = 1$ when Q is unitary. The inequality $\|AB\| \leq \|A\| \|B\|$ also holds for these norms. It follows that $\kappa(A) = \|A\| \left\|A^{-1}\right\| \geq \left\|AA^{-1}\right\| = \|I\| = 1$. Note that if Q is unitary, $Q^{-1} = Q^*$ and Q^* is also unitary, so $\kappa(Q) = \|Q\| \|Q^*\| = 1$. This means that orthogonal matrices have the smallest possible condition number, which is great!

Any unitary matrix Q can be described as a reflection, a rotation, or some combination of the two. If $det(Q) = 1$, then Q is a rotation; if $det(Q) = -1$, then Q is the composition of a reflection and a rotation. Let's explore these two types of unitary transformations and some of their applications. We will focus on the real case to simplify matters.

Householder reflections

A Householder reflection is a linear transformation $P : \mathbb{R}^n \to \mathbb{R}^n$ that reflects a vector x about a hyperplane. See figure 11.1. Recall that a hyperplane can be defined by a unit vector v which is orthogonal to the hyperplane. As shown in the figure, $x - \langle v, x\rangle v$ is the projection of x onto the hyperplane defined by v. (You should

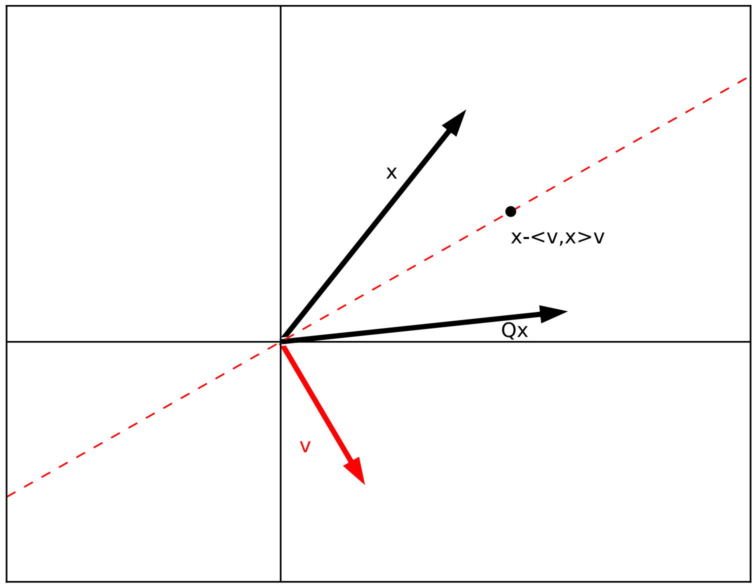


Figure 11.1: Householder reflector

verify this geometrically.) However, to reflect *across* the hyperplane, we must move twice as far; that is, $Px = x - 2\langle v, x\rangle v$. This can be written $Px = x - 2v(v^*x)$, so P has matrix representation $P = I - 2vv^*$. Note that $P^*P = I$; thus P is orthogonal.

Householder triangularization

Consider the problem of computing the QR decomposition of a matrix A. You've already learned the Gram-Schmidt and the Modified Gram-Schmidt algorithms for this problem. The QR decomposition can also be computed using Householder triangularization. Gram-Schmidt and Modified Gram-Schmidt *orthogonalize* A by a series of *triangular* transformations. Conversely, the Householder method *triangularizes* A by a series of *orthogonal* transformations.

Let's demonstrate this method on a 4×3 matrix A. First we find a orthogonal transformation Q_1 that maps the first column of A into the range of e_1 (where e_1 is the vector where the first element is one and the remander of the elements are zeros).

$$\begin{pmatrix} * & * & * \\ * & * & * \\ * & * & * \\ * & * & * \end{pmatrix} \xrightarrow{Q_1} \begin{pmatrix} * & * & * \\ 0 & \multicolumn{2}{|c|}{\begin{matrix} * & * \\ * & * \\ * & * \end{matrix}} \\ 0 & & \\ 0 & & \end{pmatrix}$$

Let A_2 be the boxed submatrix of A. Now find an orthogonal transformation Q_2 that maps the first column of A_2 into the range of e_2.

$$\begin{pmatrix} * & * \\ * & * \\ * & * \end{pmatrix} \xrightarrow{Q_2} \begin{pmatrix} * & * \\ 0 & * \\ 0 & * \end{pmatrix}$$

Similarly, $\begin{pmatrix} * \\ * \end{pmatrix} \xrightarrow{Q_3} \begin{pmatrix} * \\ 0 \end{pmatrix}$. (Technically Q_2 and Q_3 act on the whole matrix and not just on the submatrices, so that $Q_i : \mathbb{R}^n \to \mathbb{R}^n$ for all i. Q_2 leaves the first row alone, and Q_3 leaves the first two rows alone.) Then $Q_3 Q_2 Q_1 A =$

$$Q_3 Q_2 Q_1 \begin{pmatrix} * & * & * \\ * & * & * \\ * & * & * \\ * & * & * \end{pmatrix} = Q_3 Q_2 \begin{pmatrix} * & * & * \\ 0 & * & * \\ 0 & * & * \\ 0 & * & * \end{pmatrix} = Q_3 \begin{pmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \\ 0 & 0 & * \end{pmatrix} = \begin{pmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \\ 0 & 0 & 0 \end{pmatrix}$$

We've accomplished our goal, which was to triangularize A using orthogonal transformations. But now, how do we find the Q_i that do what we want? Using Householder reflections. (Surprise!)

For example, to find Q_1, we choose the right hyperplane to reflect x into the range of e_1. It turns out there are two hyperplanes that will work, as shown in figure 11.2. (In the complex case, there are infinitely many such hyperplanes.) Between the two, the one that reflects x further will be more numerically stable. This is the hyperplane perpendicular to $v = sign(x_1)\,\|x\|_2\, e_1 + x$. The whole process is summarized in Algorithm 11.0.1.

Algorithm 11.0.1: HOUSEHOLDER TRIANGULARIZATION(A)

$m, n \leftarrow size(A)$
for $k \leftarrow 1$ **to** $n-1$
do $\begin{cases} x = A_{k:m,k} \\ v_k = sign(x_1)\,\|x\|_2\, e_1 + x \\ v_k = v_k / \|v_k\|_2 \\ P_k = eye(m,m) \\ P_k[k:m, k:m] = P_k[k:m, k:m] - 2 v_k v_k^T \\ A = sp.dot(P_k, A); \end{cases}$

This algorithm returns upper triangular R. You can find Q s.t. $QR = A$ by multiplying the P_k together appropriately.

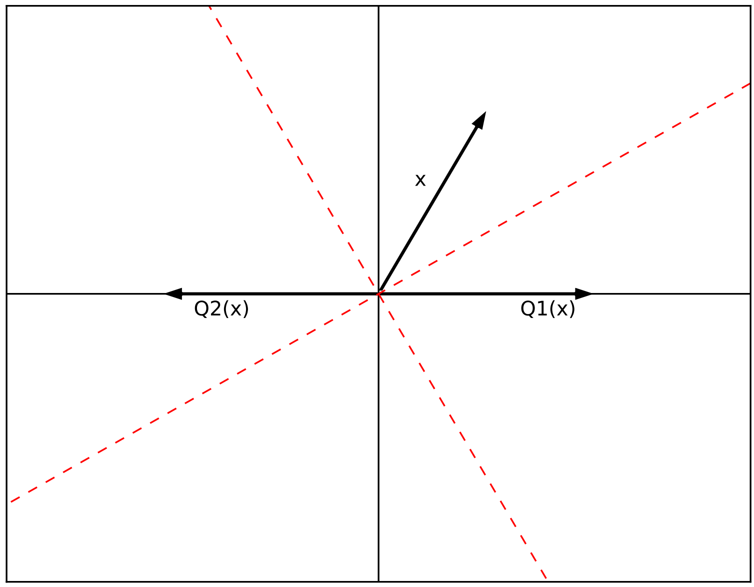


Figure 11.2: two reflectors

Problem 1 Write a script using Householder reflections to find the QR decomposition of a matrix A.

Stability of the Householder QR algorithm

Try the following in Python.

```
In [1]:  import scipy as sp
In [2]:  import numpy.linalg as la
In [3]:  import my_householder
In [4]:  Q,X = la.qr(sp.rand(50,50)) #create a random orthogonal
     matrix:
In [5]:  R = sp.triu(sp.rand(50,50)) # create a random upper
     triangular matrix
In [6]:   A = sp.dot(Q,R) #Q and R are the exact QR decomposition of A
# use your Householder QR script to estimate Q and R:
In [7]:   Q1,R1 = my_householder.qr(A)
#now check the relative errors of Q1 and R1
In [8]:  la.norm(Q1-Q)/la.norm(Q)
Out [8]:  0.282842955725
```

```
In [9]:  la.norm(R1-R)/la.norm(R)
Out[9]:  0.0428922016647
```

This is terrible! Python works in 16 decimal points of precision. But Q_1 and R_1 are only accurate to 0 and 1 decimal points, respectively. We've lost 16 decimal points of precision!

Don't lose hope. Check how close the product Q_1R_1 is to A.

```
In [10]:  A1 = sp.dot(Q1,R1)
In [11]:  la.norm(A1-A)/la.norm(A)
Out[11]:  9.73996046986e-16
```

We've now recovered 15 digits of accuracy. The errors in Q_1 and R_1 were somehow "correlated," so that they canceled out in the product. The errors in Q_1 and R_1 are called *forward errors*. The error in A_1 is the *backward error*. The Householder QR algorithm is a backward stable algorithm.

Householder QR factorization is more numerically stable than Gram-Schmidt or even Modified Gram-Schmidt (MGS). However, MGS is still useful for some types of iterative methods, because it finds the orthogonal basis one vector at a time instead of all at once (for example see Lab 15).

Upper Hessenberg Form

An upper Hessenberg matrix is a square matrix with zeros below the first subdiagonal. Every $n \times n$ matrix A can be written $A = Q^T HQ$ where Q is orthogonal and H is an upper Hessenberg matrix, called the Hessenberg form of A. Note the similarity of this decomposition to the Schur decomposition in Lab 35.

The Hessenberg decomposition can be computed using Householder reflections, in a process very similar to Householder triangularization. Let's demonstrate this process on a 5×5 matrix A. Note that $A = Q^T HQ$ is equivalent to $QAQ^T = H$; thus our strategy is to multiply A on the right and left by a series of orthogonal matrices until it is in Hessenberg form. If we try the same Q_1 as in the first step of the Householder algorithm, then with Q_1A we introduce zeros in the first column of A. However, since we now have to multiply Q_1A on the left by Q_1^T, all those zeros are destroyed, as demonstrated below. (Although this process may seem futile now, it actually does tend to decrease the size of the subdiagonal entries. If we repeat it over and over again, the subdiagonal entries will often converge to zero. That's the idea behind the QR algorithm in Lab 15.)

$$\underset{A}{\begin{pmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{pmatrix}} \xrightarrow{Q_1\cdot} \underset{Q_1A}{\begin{pmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{pmatrix}} \xrightarrow{\cdot Q_1^T} \underset{Q_1AQ_1^T}{\begin{pmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{pmatrix}}$$

Instead, let's try starting with a different Q_1 that leaves the *first* row alone and reflects the *rest* of the rows into the range of e_2. This means that Q_1^T leaves the first column alone.

$$\underset{A}{\begin{pmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{pmatrix}} \xrightarrow{Q_1\cdot} \underset{Q_1 A}{\begin{pmatrix} * & * & * & * & * \\ * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{pmatrix}} \xrightarrow{\cdot Q_1^T} \underset{Q_1 A Q_1^T}{\begin{pmatrix} * & * & * & * & * \\ * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{pmatrix}}$$

We now iterate through the matrix until we obtain

$$Q_3 Q_2 Q_1 A Q_1^T Q_2^T Q_3^T = \begin{pmatrix} * & * & * & * & * \\ * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * \end{pmatrix}$$

Problem 2 Write a script that transfers an input matrix to upper Hessenberg form. (Hint: You only need to modify your code code from problem 1 slightly.) We will use this technique in the eigenvalue lab later.