Lab 7

QR Decomposition

Lab Objective: *Use the Gram-Schmidt algorithm and orthonormal transformations to perform the QR decomposition.*

The QR decomposition of a matrix A is a factorization $A = QR$, where Q has orthonormal columns and R is upper triangular. This decomposition is useful for computing least squares and finding eigenvalues. As stated in the following theorem, the QR decomposition of A always exists when the rank of A equals the number of columns of A.

Theorem 7.1. *Let A be an $m \times n$ matrix of rank n. Then A can be factored into a product QR, where Q is an $m \times m$ matrix with orthonormal columns and R is a $m \times n$ upper triangular matrix of rank n.*

In this lab we will only discuss real matrices. All of these results can be extended to the complex numbers by replacing "orthogonal matrix" with "unitary matrix," "transpose" with "hermitian conjugate," and "symmetric matrix" with "hermitian matrix."

Modified Gram-Schmidt

Let A be an $m \times n$ matrix of rank n. There are many methods for computing the QR decomposition. First we will discuss the algorithm that computes Q by applying Gram-Schmidt to the columns of A.

Let $\{\mathbf{x}_i\}_{i=1}^n$ be the columns of A (the rank hypothesis implies that these are linearly independent vectors). Then the Gram-Schmidt algorithm computes an orthonormal basis $\{\mathbf{q}_i\}_{i=1}^n$ for the span of the $\mathbf{x}_i$. The Gram-Schmidt algorithm defines

$$\mathbf{q}_1 = \frac{\mathbf{x}_1}{\|\mathbf{x}_1\|}.$$

It recursively defines $\mathbf{q}_k$ for $k > 1$ by

$$\mathbf{q}_k = \frac{\mathbf{x}_k - \mathbf{p}_{k-1}}{\|\mathbf{x}_k - \mathbf{p}_{k-1}\|}, \quad k = 2, \dots, n,$$

where

$$\mathbf{p}_{k-1} = \sum_{i=1}^{k-1} \langle \mathbf{q}_i, \mathbf{x}_k \rangle \mathbf{q}_i, \quad k = 2, \ldots, n$$

and $\mathbf{p}_0 = 0$.

Let Q be the matrix with columns $\{\mathbf{q}_i\}_{i=1}^n$. Let R be the upper triangular matrix with entries r_{jk} where $r_{kk} = \|\mathbf{x}_k - \mathbf{p}_{k-1}\|$ and $r_{jk} = \langle \mathbf{q}_j, \mathbf{x}_k \rangle$ when $j < k$. Then $QR = A$ and R is nonsingular (see [ref:textbook] for a proof). Thus, if A is square, the matrices Q and R are its QR decomposition.

When implemented with a computer, the Gram-Schmidt algorithm may produce a matrix Q with columns that are not even close to orthonormal due to rounding errors. We now introduce the modified Gram-Schmidt algorithm, which consistently produces matrices Q whose columns are "very close" to orthonormal.

In the modified Gram-Schmidt algorithm, $\mathbf{q}_1$ is the normalization of $\mathbf{x}_1$ as before. We then make each of the vectors $\mathbf{x}_2, \ldots, \mathbf{x}_n$ orthogonal to $\mathbf{q}_1$ by defining

$$\mathbf{x}_k := \mathbf{x}_k - \langle \mathbf{q}_1, \mathbf{x}_k \rangle \mathbf{q}_1, \quad k = 2, \ldots, n.$$

(Compare this to the usual Gram-Schmidt algorithm, where we only made $\mathbf{x}_2$ orthogonal to $\mathbf{q}_1$.) Next we define $\mathbf{q}_2 = \frac{\mathbf{x}_2}{\|\mathbf{x}_2\|}$. Once again we make $\mathbf{x}_3, \ldots, \mathbf{x}_n$ orthogonal to $\mathbf{q}_2$ by defining

$$\mathbf{x}_k := \mathbf{x}_k - \langle \mathbf{q}_2, \mathbf{x}_k \rangle \mathbf{q}_2, \quad k = 3, \ldots, n.$$

(Each of these new vectors is a linear combination of vectors orthogonal to $\mathbf{q}_1$, and hence is orthogonal to $\mathbf{q}_1$ as well.) We continue this process until we have an orthonormal set $\mathbf{q}_1, \ldots, \mathbf{q}_k$. The entire modified Gram-Schmidt algorithm is described in Algorithm 7.1.

Algorithm 7.1 The modified Gram-Schmidt. This algorithm returns orthogonal Q and upper triangular R such that $A = QR$.

```
1:  procedure MODIFIED GRAM-SCHMIDT(A)
2:      m, n ← shape(A)
3:      Q ← copy(A)
4:      R ← zeros((n, n))
5:      for 0 ≤ i < n do
6:          R_{i,i} ← ‖Q_{:,i}‖
7:          Q_{:,i} ← Q_{:,i} / R_{i,i}
8:          for i + 1 ≤ j < n do
9:              R_{i,j} ← Q_{:,j}^T Q_{:,i}
10:             Q_{:,j} ← Q_{:,j} − R_{i,j} Q_{:,i}
11:     return Q, R
```

QR decomposition in SciPy

The linear algebra library in SciPy calculates the QR decomposition using a software package called LAPACK (Linear Algebra PACKage), which is incredibly efficient. Here is an example of using SciPy to compute the QR decomposition of a matrix.

```
>>> import numpy as np
>>> from scipy import linalg as la

>>> A = np.random.rand(4,3)
>>> Q, R = la.qr(A)
>>> Q.dot(R) == A
array([[ True, False, False],
       [ True, False, False],
       [ True,  True, False],
       [ True,  True, False]], dtype=bool)
```

Note that `Q.dot(R)` does not equal `A` exactly because of rounding errors. However, we can check that the entries of `Q.dot(R)` are "close" to the entries of `A` with the NumPy method `np.allclose()`. This method checks that the elements of two arrays differ by less than a given tolerance, a tolerance specified by two keyword arguments `rtol` and `atol` that default to 10^{-5} and 10^{-8} respectively. You can read the documentation to learn how the tolerance is computed from these numbers.

```
>>> np.allclose(Q.dot(R), A)
True
```

We can use the same method to check that `Q` is "very close" to an orthogonal matrix.

```
>>> np.allclose(Q.T.dot(Q), np.eye(4))
True
```

Problem 1. Write a function that accepts as input a $m \times n$ matrix of rank n and computes its QR decomposition, returning the matrices Q and R. Your function should use Algorithm 7.1. Hint: Read about the function `np.inner()`. Another hint: Check that your function works by using `np.allclose()`.

Problem 2. Write a function that accepts a square matrix A of full rank and returns $|\det(A)|$. Use the QR decomposition of A to perform this calculation. Hint: What is the determinant of an orthonormal matrix?

Householder triangularization

Another way to compute the QR decomposition of a matrix is with a series of orthonormal transformations. Like the Modified Gram-Schmidt algorithm, orthonormal transformations are numerically stable, meaning that they are less susceptible to rounding errors.

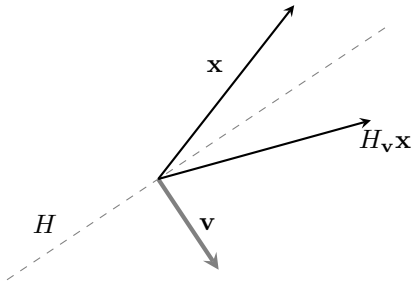

Figure 7.1: This is a picture of a Householder transformation. The normal vector $\mathbf{v}$ defines the hyperplane H. The Householder transformation $H_{\mathbf{v}}$ of $\mathbf{x}$ is just the reflection of $\mathbf{x}$ across H.

Householder transformations

The *hyperplane* in $\mathbb{R}^n$ with normal vector $\mathbf{v}$ is the set $\{\mathbf{x} \in \mathbb{R}^n \mid \langle \mathbf{x}, \mathbf{v} \rangle = 0\}$. Equivalently, the hyperplane defined by $\mathbf{v}$ is just the orthogonal complement $\mathbf{v}^\perp$. See Figure 7.1.

A *Householder transformation* of $\mathbb{R}^n$ is a linear transformation that reflects about a hyperplane. If a hyperplane H has normal vector $\mathbf{v}$, let $\mathbf{u} = \mathbf{v}/\|\mathbf{v}\|$. Then the Householder transformation that reflects about H corresponds to the matrix $H_{\mathbf{u}} = I - 2\mathbf{u}\mathbf{u}^T$. You can check that $(I - 2\mathbf{u}\mathbf{u}^T)^T(I - 2\mathbf{u}\mathbf{u}^T) = I$, so Householder transformations are orthogonal.

Householder triangularization

The QR decomposition of an $m \times n$ matrix A can also be computed with Householder transformations via the Householder triangularization. Whereas Gram-Schmidt makes A *orthonormal* using a series of transformations stored in an *upper triangular* matrix, Householder triangularization makes A *triangular* by a series of *orthonormal* transformations. More precisely, the Householder triangularization finds an $m \times m$ orthogonal matrix Q and and $m \times n$ upper triangular matrix R such that $QA = R$. Since Q is orthogonal, $Q^{-1} = Q^T$ so $A = Q^T R$, and we have discovered the QR decomposition.

Let's demonstrate the idea behind Householder triangularization on a 4×3 matrix A. Let $e_1, \ldots, e_4$ be the standard basis of $\mathbb{R}^4$. First we find an orthonormal transformation Q_1 that maps the first column of A into the span of e_1. This is diagrammed below, where $*$ represents an arbitrary entry.

$$\begin{pmatrix} * & * & * \\ * & * & * \\ * & * & * \\ * & * & * \end{pmatrix} \xrightarrow{Q_1} \begin{pmatrix} * & * & * \\ 0 & \boxed{*} & \boxed{*} \\ 0 & \boxed{*} & \boxed{*} \\ 0 & \boxed{*} & \boxed{*} \end{pmatrix}$$

Let $A_2 = Q_1 A$ be the matrix on the right above. Now find an orthonormal transformation Q_2 that fixes e_1 and maps the second column of A_2 into the span of e_1

and e_2. Notice that since Q_2 fixes e_1, the top row of A_2 will be fixed by Q_2, and only entries in the boxed submatrix will change.

Let $A_3 = Q_2Q_1A$ be the matrix on the right above. Finally, find an orthonormal transformation Q_3 that fixes e_1 and e_2 and maps the third column of A_3 into the span of e_1, e_2, and e_3. The diagram below summarizes this process, where boxed entries indicate those affected by the operation just performed.

$$Q_3Q_2Q_1\begin{pmatrix}*&*&*\\*&*&*\\*&*&*\\*&*&*\end{pmatrix} = Q_3Q_2\begin{pmatrix}\boxed{\begin{matrix}*&*&*\\0&*&*\\0&*&*\\0&*&*\end{matrix}}\end{pmatrix} = Q_3\begin{pmatrix}*&*&*\\0&\boxed{\begin{matrix}*&*\\0&*\\0&*\end{matrix}}\end{pmatrix} = \begin{pmatrix}*&*&*\\0&*&*\\0&0&\boxed{*}\\0&0&\boxed{0}\end{pmatrix}.$$

We've accomplished our goal, which was to triangularize A using orthonormal transformations. But how do we construct the matrices Q_k?

It turns out that we can choose each Q_k to be a Householder transformation. Suppose we have found Householder transformations $Q_1, \ldots, Q_{k-1}$ such that $Q_{k-1} \ldots Q_2Q_1 = A_k$ where

$$A_k = \begin{pmatrix} T & X' \\ 0 & X'' \end{pmatrix}.$$

Here, T is a $(k-1) \times (k-1)$ upper triangular matrix. Let $\mathbf{x}$ be the k^{th} column of A_k. Write $\mathbf{x} = \mathbf{x}' + \mathbf{x}''$ where $\mathbf{x}'$ is in the span of $e_1, \ldots, e_{k-1}$. So $\mathbf{x}''$ looks like $k-1$ zeros followed by the first column of X''. The idea is to reflect $\mathbf{x}''$ about a hyperplane into the span of e_k. It turns out that there are two choices that will work (see Figure 7.2). These hyperplanes have normal vectors $\mathbf{x}'' + \|\mathbf{x}''\|e_k$ and $\mathbf{x}'' - \|\mathbf{x}''\|e_k$. In fact, the more numerically stable transformation is to reflect about the hyperplane with normal vector $\mathbf{v}_k = \mathbf{x}'' + \text{sign}(x_k)\|\mathbf{x}''\|e_k$, where x_k is the k^{th} entry of $\mathbf{x}''$ (or the top left entry of X''). (You can check that $\mathbf{v}_k$ is orthonormal to $e_1, \ldots, e_{k-1}$, so the plane orthonormal to $\mathbf{v}_k$ contains $e_1, \ldots, e_{k-1}$, and reflecting about it fixes $e_1, \ldots, e_{k-1}$.) Thus, Q_k is the Householder transformation $H_{\mathbf{v}_k}$.

The final question is to find an efficient algorithm for computing $Q = Q_nQ_{n-1} \ldots Q_1$ and $R = Q_nQ_{n-1} \ldots Q_1A$. The idea is to start with $R = A$ and $Q = I$. Then we compute Q_1 and modify R to be Q_1A and Q to be Q_1. Next we compute Q_2 and modify R to be Q_2Q_1A and Q to be Q_2Q_1, and so forth. As we have already discussed, Q_k fixes the first $k-1$ rows and columns of any matrix it acts on. In fact, if $\mathbf{x}'' = (0, \ldots, 0, x_k, x_{k+1}, \ldots, x_n)$ as above, then $\mathbf{v}_k = (0, \ldots, 0, v_{k_0}, x_{k+1}, \ldots, x_n)$ where $v_{k_0} = x_k + \text{sign}(x_k)\|\mathbf{x}''\|$. If $\mathbf{u}_k$ is the normalization of $(v_{k_0}, x_{k+1}, \ldots, x_n) \in \mathbb{R}^{n-(k-1)}$, then

$$Q_k = I - \frac{2\mathbf{x}''(\mathbf{x}'')^T}{\|\mathbf{x}''\|^2} = \begin{pmatrix} I & 0 \\ 0 & I - 2\mathbf{u}_k\mathbf{u}_k^T \end{pmatrix}.$$

This means that, using block multiplication,

$$Q_kA_k = \begin{pmatrix} I & 0 \\ 0 & I - 2\mathbf{u}_k\mathbf{u}_k^T \end{pmatrix}\begin{pmatrix} T & X' \\ 0 & X'' \end{pmatrix} = \begin{pmatrix} T & X' \\ 0 & (I - 2\mathbf{u}_k\mathbf{u}_k^T)X'' \end{pmatrix}.$$

So at each stage of the algorithm, we only need to update the entries in the bottom right submatrix of A_k, and these change via matrix multiplication by $I - 2\mathbf{u}_k\mathbf{u}_k^T$.

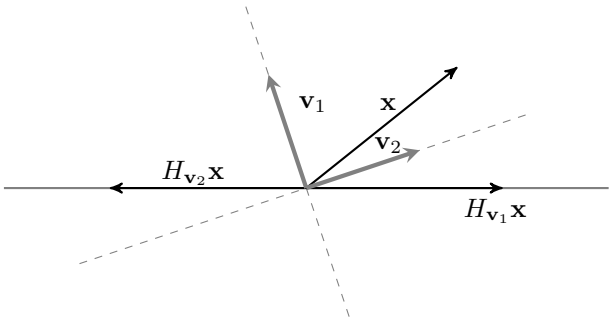

Figure 7.2: If we want to reflect $\mathbf{x}$ about a hyperplane into the span of e_1, there are two hyperplanes that will work. The two choices are defined by the normal vectors $\mathbf{v}_1$ and $\mathbf{v}_2$. Reflecting about the hyperplane defined by $\mathbf{v}_i$ produces $H_{\mathbf{v}_i}\mathbf{x}$.

Similarly,

$$Q_k Q_{k-1} \dots Q_1 = Q_k \begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} I & 0 \\ 0 & I - 2\mathbf{u}_k \mathbf{u}_k^T \end{pmatrix} \begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} A \\ (I - 2\mathbf{u}_k \mathbf{u}_k^T) B \end{pmatrix},$$

so to update $\prod Q_i$, we need only modify the bottom rows. These also change via matrix multiplication by $I - 2\mathbf{u}_k \mathbf{u}_k^T$.

These arguments produce Algorithm 7.2.

Algorithm 7.2 Householder triangularization. This algorithm returns orthonormal Q and upper triangular R satisfying $A = QR$.

procedure HOUSEHOLDER(A)
  $m, n \leftarrow \text{shape}(A)$
  $R \leftarrow \text{copy}(A)$
  $Q \leftarrow I_m$
  for $0 \leq k < n - 1$ **do**
    $u_k \leftarrow \text{copy}(R_{k:,k})$
    $u_{k_0} \leftarrow u_{k_0} + \text{sign}(u_{k_0}) \|u_k\|$
    $u_k \leftarrow u_k / \|u_k\|$
    $R_{k:,k:} \leftarrow R_{k:,k:} - 2u_k \left(u_k^\mathsf{T} R_{k:,k:}\right)$
    $Q_{k:} \leftarrow Q_{k:} - 2u_k \left(u_k^\mathsf{T} Q_{k:}\right)$
  return Q^H, R

Problem 3. Write a function that accepts as input a $m \times n$ matrix of rank n and computes its QR decomposition, returning the matrices Q and R. Your function should use Algorithm 7.2. Hint: Read about the function `np.outer()`.

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 $A = Q^T H Q$ where Q is orthonormal and H is an upper Hessenberg matrix, called the Hessenberg form of A.

A fast algorithm for computing the QR decomposition of a Hessenberg matrix will be taught in Lab 8. This algorithm in turn leads to a fast algorithm for finding eigenvalues of a matrix, which will be discussed in Lab 10.

For now, we will outline an algorithm for computing the upper Hessenberg form of any matrix. Like Householder triangularization, this algorithm uses Householder transformations. To find orthogonal Q and upper Hessenberg H such that $A = Q^T H Q$, it suffices to find such matrices that satisfy $Q^T A Q = H$. Thus, our strategy is to multiply A on the right and left by a series of orthonormal matrices until it is in Hessenberg form. If we use the same Q_1 as in the first step of the Householder algorithm, then with $Q_1 A$ we introduce zeros in the first column of A. However, since we now have to multiply $Q_1 A$ on the left by Q_1^T, all those zeros are destroyed.

Instead, let's try choosing a Q_1 that fixes e_1 and reflects the first column of A into the span of e_1 and e_2. Because Q_1 fixes e_1, the product $Q_1 A$ leaves the first row of A alone, and $(Q_1 A)Q_1^T$ leaves the first column of $(Q_1 A)$ alone. If A is a 5×5 matrix, this looks like

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

We now iterate through the matrix until we obtain

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

The pseudocode for computing the Hessenberg form of a matrix with Householder transformations is shown in Algorithm 7.3. Although the Hessenberg form exists for any square matrix, this algorithm only works for full-rank square matrices. Notice that this algorithm is very similar to Algorithm 7.2.

When A is symmetric, its upper Hessenberg form is a tridiagonal matrix. This is because the Q_i's zero out everything below the first subdiagonal of A and the Q_i^T's zero out everything above the first superdiagonal. Thus, the Hessenberg form of a symmetric matrix is especially useful, since as we saw in Lab 4, tridiagonal matrices make computations fast.

Algorithm 7.3 Reduction to Hessenberg form for a nonsingular matrix. This algorithm returns orthogonal Q and upper Hessenberg H such that $A = Q^T H Q$.

procedure HESSENBERG(A)
  $m, n \leftarrow \text{shape}(A)$
  $H \leftarrow \text{copy}(A)$
  $Q \leftarrow I_m$
  for $0 \leq k < n - 2$ **do**
    $u_k \leftarrow \text{copy}\,(H_{k+1:,k})$
    $u_{k_0} \leftarrow u_{k_0} + \text{sign}(u_{k_0})\,\|u_k\|$
    $u_k \leftarrow u_k / \|u_k\|$
    $H_{k+1:,k:} \leftarrow H_{k+1:,k:} - 2u_k(u_k^\mathsf{T} H_{k+1:,k:})$
    $H_{:,k+1:} \leftarrow H_{:,k+1:} - 2(H_{:,k+1:}u_k)u_k^\mathsf{T}$
    $Q_{k+1:} \leftarrow Q_{k+1:} - 2u_k(u_k^\mathsf{T} Q_{k+1:})$
  return Q, H

Problem 4. Write a function that accepts as input a nonsingular square matrix A and computes its Hessenberg form, returning orthogonal Q and upper Hessenberg H satisfying $A = Q^T H Q$. Your function should use Algorithm **??**. What happens when you compute the Hessenberg factorization of a symmetric matrix?