QR Factorization

Householder's Algorithm

Adam DeJans

Oakland University - Mathematics & Statistics

Table of contents

1. Introduction
2. Abstract Presentation
3. Python 2 Implementation
4. Testing
5. Runtime Analysis
6. Experiment and Report
7. Test Questions

Introduction

Introduction

Recall the QR decomposition of a matrix $A_{m \times n}$ is

$$A = QR$$

where Q is an orthogonal[1] matrix and R is upper trapezoidal.

[1] $QQ^T = I = Q^TQ$

Over-determined QR Example

An example of a QR factorization of an overdetermined 4×3 matrix.

$$
\begin{aligned}
A = \begin{bmatrix} -1 & -1 & 1 \\ 1 & 3 & 3 \\ -1 & -1 & 5 \\ 1 & 3 & 7 \end{bmatrix} &= \begin{bmatrix} q_1 & q_2 & q_3 & q_4 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ 0 & R_{22} & R_{23} \\ 0 & 0 & R_{33} \\ 0 & 0 & 0 \end{bmatrix} \\
&= \begin{bmatrix} -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 2 & 4 & 2 \\ 0 & -2 & -8 \\ 0 & 0 & -4 \\ 0 & 0 & 0 \end{bmatrix}
\end{aligned} \tag{1}
$$

Over-determined QR Example

We see that $Q = \begin{bmatrix} -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \end{bmatrix}$ and note that

$$Q^T Q = \begin{bmatrix} -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Under-determined QR Example

An example of a QR factorization of an underdetermined 3×4 matrix.

$$
\begin{aligned}
A &= \begin{bmatrix} -1 & 1 & -1 & 1 \\ -1 & 3 & -1 & 3 \\ 1 & 3 & 5 & 7 \end{bmatrix} \\
&= \begin{bmatrix} -0.577 & -0.154 & 0.801 \\ -0.577 & -0.617 & -0.534 \\ 0.577 & -0.771 & 0.267 \end{bmatrix} \begin{bmatrix} 1.732 & -0.577 & 4.041 & 1.732 \\ 0.000 & -4.320 & -3.086 & -7.406 \\ 0.000 & 0.000 & 1.069 & 1.069 \end{bmatrix} \\
&= QR
\end{aligned} \tag{2}
$$

Note the Orthongality of Q.

Notice that the Q has the property that $QQ^T = I = Q^TQ$. That is, it's inverse is it's transpose. We see that

$$\begin{bmatrix} -0.5774 & -0.1543 & 0.8018 \\ -0.5774 & -0.6172 & -0.5345 \\ 0.5774 & -0.7715 & 0.2673 \end{bmatrix} \begin{bmatrix} -0.5774 & -0.5774 & 0.5774 \\ -0.1543 & -0.6172 & -0.7715 \\ 0.8018 & -0.5345 & 0.2673 \end{bmatrix} = I$$

and

$$I = \begin{bmatrix} -0.5774 & -0.5774 & 0.5774 \\ -0.1543 & -0.6172 & -0.7715 \\ 0.8018 & -0.5345 & 0.2673 \end{bmatrix} \begin{bmatrix} -0.5774 & -0.1543 & 0.8018 \\ -0.5774 & -0.6172 & -0.5345 \\ 0.5774 & -0.7715 & 0.2673 \end{bmatrix}$$

Recall: The linear system of equations

$$Ax = b,$$

where A is $n \times n$ and $b \in \mathbb{R}^n$, can be solved using Gaussian elimination with partial pivoting (also called LU factorization). We can also solve for x by finding the inverse of A and noting that $x = A^{-1}b$.[2]

This however is not possible in the case where A is size $m \times n$ where $m > n$ or when A is singular.

[2] Note: Never, ever compute A^{-1} if you write software.

Linear Least Squares Problem

Given $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$, find vector $x \in \mathbb{R}^n$ that minimizes

$$\|Ax - b\|^2 = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} A_{ij}x_j - b_i \right)^2$$

The nice theoretical solution, although computationally unstable, is

$$(A^T A)^{-1} A^T b.$$

QR Factorization Applied to Least Squares Problem

Algorithm 1: QR Factorization Method Applied to Least Squares

Data: $A_{m \times n}, b$

Result: Solution to Least-Squares Problem

Compute QR factorization $A = QR$

Matrix-vector product $d = Q^T b$

Solve $R\hat{x} = d$ by back substitution

return $\hat{x}$

Example of QR Factorization Applied to Least Squares Problem

$$A = \begin{bmatrix} 3 & 5 & 2 \\ 1 & 2 & 4 \\ 0 & 1 & 2 \end{bmatrix}, b = \begin{bmatrix} 1 \\ 2 \\ 5 \end{bmatrix}$$

1. QR factorization $A = QR$ with

$$Q = \begin{bmatrix} -0.9487 & 0.0953 & 0.3015 \\ -0.3162 & -0.2860 & -0.9045 \\ 0.0000 & -0.9535 & 0.3015 \end{bmatrix},$$

$$R = \begin{bmatrix} -3.1623 & -5.3759 & -3.1623 \\ 0.0000 & -1.0488 & -2.8604 \\ 0.0000 & 0.0000 & -2.4121 \end{bmatrix}$$

Example of QR Factorization Applied to Least Squares Problem

2. Calculate $d = Q^T b = \begin{bmatrix} -10.1194 \\ -5.2442 \\ 0 \end{bmatrix}$

3. Solve $Rx = d$ by using back substitution

$$R = \begin{bmatrix} -3.1623 & -5.3759 & -3.1623 \\ 0.0000 & -1.0488 & -2.8604 \\ 0.0000 & 0.0000 & -2.4121 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -10.1194 \\ -5.2442 \\ 0 \end{bmatrix}$$

$x_3 = 0,$
$x_2 = \frac{-5.2442+2.8604(x_3)}{-1.0488},$
$x_1 = \frac{-10.1194+3.1623(x_3)+5.3759(x_2)}{-3.1623}$

QR Factorization Applied to Least Squares Problem

Notice that we rewrite the least squares solution using $A = QR$

$$\begin{aligned}
\hat{x} = (A^T A)^{-1} A^T b &= ((QR)^T (QR))^{-1} (QR)^T b \\
&= (R^T Q^T Q R)^{-1} R^T Q^T b \\
&= (R^T R)^{-1} R^T Q^T b \\
&= R^{-1} R^{-T} R^T Q^T b \\
&= R^{-1} Q^T b
\end{aligned} \tag{3}$$

The problem with computing the known solution occurs when forming Gram matrix $A^T A$. QR factorization method is more stable because it avoids forming $A^T A$.

Definitions

- Two vectors $u, v \in \mathbb{R}^n$ are orthogonal if $u \cdot v = 0$.

Recall that one way of expressing the Gaussian elimination algorithm is in terms of Gauss transformations that serve to introduce zeros into the lower triangle of a matrix.

Householder transformations are simple orthogonal transformations, corresponding to reflection through a plane, which can be used to similar effect. Reflection across the plane orthogonal to a unit normal vector v can be expressed in matrix form as

$$H = I - 2vv^T.$$

We see that the idea of QR factorization is similar to the Gauss elimination LU factorization when it comes to solving the least-squares problem.

Given an $m \times n$ matrix A, we bring it into an upper trapezoidal form R by multiplying it from the left by appropriately chosen Householder matrices. Since $A = QR$ and $Q^TQ = I$, when we multiply A by Q^T we get R:

$$Q^TA = Q^T(QR) = (Q^TQ)R = R.$$

The Householder algorithm returns the vectors that define Q so we don't need to explicitly find Q to solve the least-squares problem. When solving the least-squares minimization problem we have to compute Q^Tb but we see that $Q^Tb = H_n H_{n-1} \cdots H_1 b$; that is, we can apply the matrices $H_1, H_2, \ldots, H_n$ to b in the order they are generated to obtain the vector Q^Tb. In particular, the entries of the matrix Q do not have to be computed explicitly.

Abstract Presentation

Geometrical Idea

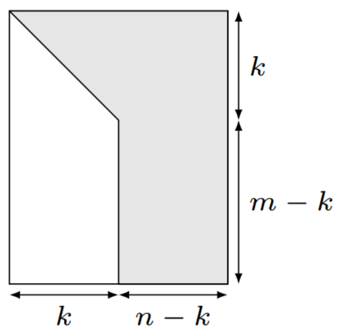
Start with an $m \times n$ matrix $A_{m\times n}$. We want to apply some orthogonal linear operator on A so that the first column becomes a multiple of vector e_1. At step k we apply the k^{th} orthogonal linear operator on A so that $A_{k:m,k}$ becomes a multiple of e_1.

Householder Triangularization

Computes reflectors $H_1, \ldots, H_n$ that reduce A to triangular form:

$$H_n H_{n-1} \cdots H_1 A = \begin{bmatrix} R & 0 \end{bmatrix}^T$$

After step k, the matrix $H_k H_{k-1} \cdots H_1 A$ has the following structure:



Note: elements in positions i, j for $i > j$ and $j \leq k$ are zero.

Geometry to Obtain H

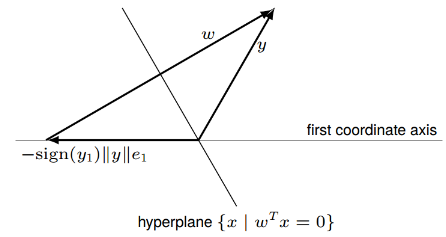


Figure 1: Geometrical image on obtaining matrix H. Note: $y = A_{k:m,k}$ Note 2: Orthogonal projection (Gram-Schmidt), could be used but is numerically unstable when vectors y and e_1 are close to orthogonal. Instead, Householder's reflects through the hyperplane.

Matrix H can be found via reflection. Our goal is to obtain

$$-sign(y_1)\|y\|e_1$$

by first finding the hyperplane of reflection. We see that

$$w = y - (-sign(y_1)\|y\|e_1) = y + sign(y_1)\|y\|e_1$$

is a vector *normal* to the hyperplane.

Geometry to Obtain H

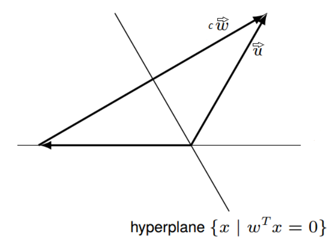


Figure 2: A geometrical image on how to obtain matrix H_u. $c\vec{w}$ is a scalar multiple of w.

Since w is an orthogonal vector of the hyperplane $\{x \mid w^T x = 0\}$ we see that the orthogonal linear operation $H_u : u \mapsto u - 2\frac{w}{\|w\|}\frac{w^T u}{\|w\|} = \left(I - 2\frac{ww^t}{\|w\|^2}\right) u$ reflects u over $\{x \mid w^T x = 0\}$.

Notice we can simplify this expression by letting $v = \frac{w}{\|w\|}$.

Algorithm

Algorithm 2: Householder Algorithm

Data: A

Result: Overwrites A with $\begin{bmatrix} R & 0 \end{bmatrix}^T$

```
for k=1 to n do
    y = A_{k:m,k}
    w = y + sign(y_1)||y||e_1
    v_k = (1/||w||) w
    A_{k:m,k:n} = (I − 2 v_k v_k^T) A_{k:m,k:n}
end
return A,v
```

$$y = A_{k:m,k}$$

$$w = y + sign(y_1)\|y\|e_1$$

$$v_k = \frac{1}{\|w\|}w$$

$$A_{k:m,k:n} = (I - 2v_k v_k^T)A_{k:m,k:n}$$

Householder Algorithm Example

Compute the QR decomposition of

$$A = \begin{bmatrix} 3 & -2 & 3 \\ 0 & 3 & 5 \\ 4 & 4 & 4 \end{bmatrix}.$$

Householder Algorithm Example, $k = 1$:

1. $w = a_1 - sign(a_{11})\|a_1\|e_1 = \begin{bmatrix} 3 \\ 0 \\ 4 \end{bmatrix} + sign(3)(5)\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 8 \\ 0 \\ 4 \end{bmatrix}$
2. $v_1 = \frac{w}{\|w\|_2} = \frac{1}{4\sqrt{5}}\begin{bmatrix} 8 \\ 0 \\ 4 \end{bmatrix} = \begin{bmatrix} \frac{2}{\sqrt{5}} \\ 0 \\ 4 \end{bmatrix}$
3. $A = A - 2v(v^T A) = \begin{bmatrix} -5 & -2 & -5 \\ 0 & 3 & 5 \\ 0 & 4 & 0 \end{bmatrix}$

Householder Algorithm Example, $k = 2$:

1. $w = A_{2:3,2} - sign(a_{22})\|A_{2:3,2}\|e_1 = \begin{bmatrix} 3 \\ 4 \end{bmatrix} + sign(3)(5)\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 8 \\ 4 \end{bmatrix}$
2. $v_2 = \frac{w}{\|w\|_2} = \frac{1}{4\sqrt{5}}\begin{bmatrix} 8 \\ 4 \end{bmatrix} = \begin{bmatrix} \frac{2}{\sqrt{5}} \\ 4 \end{bmatrix}$
3. $A_{2:3,2:3} = A_{2:3,2:3} - 2v(v^T A_{2:3,2:3}) = \begin{bmatrix} -5 & -3 \\ 0 & -4 \end{bmatrix}$
4. We see that only the lower, right 2×2 corner of A has changed, so we now have $A = \begin{bmatrix} -5 & -2 & -5 \\ 0 & -5 & -3 \\ 0 & 0 & -4 \end{bmatrix}$

Householder Algorithm Example, $k = 3$:

We see that the current structure of A is triangular, so no work is required for $k = 3$. We thus have

$$R = \begin{bmatrix} -5 & -2 & -5 \\ 0 & -5 & -3 \\ 0 & 0 & -4 \end{bmatrix}.$$

Notice that if we ever need Q or Q^T explicitly, we can form them from the compressed representation of v. We can also multiply Q and Q^T implicitly if needed (as in a least-squares problem). The vectors $v_1, \ldots, v_n$ define $H_1, \ldots, H_n$ and are thus an economical representation of Q.

Applying/Forming Q

Algorithm 3: Implicit Calculation of $Q^T b$

Data: v, b, m, n

Result: Calculates $Q^T b$

$Q^T b = b$

for k=1 *to* n **do**

$\quad Q^T b_{k:m} = Q^T b_{k:m} - 2 v_k (v_k Q^T b_{k:m})$

end

return $Q^T b$

Python 2 Implementation

Overview

- Imported helper packages
- Householder algorithm
- QR factorization algorithm

Import Helper Packages

Imported packages

- numpy is used for helper functions such as finding the norm and multiplication.
- numpy and scipy both have a QR factorization method which is used to compare with the written code.
- pandas is used as a way to keep track of runtimes.

```python
import pandas
import numpy as np
import time
import scipy
from scipy import linalg
```

Householder Algorithm

```
def householder(b):
    y = b.copy()
    y[0] = y[0] + np.sign(y[0])*np.linalg.norm(y)
    v = y / np.linalg.norm(y)
    return v
```

QR Factorization Algorithm

```
def qr(a):
    A = a.copy()
    m, n = A.shape # Dimenstions of matrix A
    v = [0]*n
    # Check for over/under determined matrix
    if m == n: # square matrix
        r = n−1 # we don’t need to manipulate the 1x1 matrix
    elif m > n: # overdetermined
        r = n
    else: # underdetermined; note this won’t create an upper triangle matrix
        r = m−1
    for i in range(r):
        v[i] = householder(A[i:, i])
        vk = np.array(v[i])
        A[i:,i:] −= 2*np.matmul(np.matmul(vk.reshape(m−i,1),
                                          vk.reshape(1,m−i)),A[i:,i:])
    return A, v
```

Python Code of Implicit Q^T Multiplication on b from $Ax = b$.

Here we can see how the vector v returned from the QR factorization function can be used to implicitly calculate $Q^T b$, a step in solving the least-squares problem.

```
def QTb(v,b,m,n,A):
    QTb = b.copy()
    for i in range(n):
    # b[k:m] = b[k:m] − 2*vk(vk*b[k:m])
    QTb[i:] = np.subtract(QTb[i:],2.0*np.matmul(np.array(v[i]).reshape(m−i,1),
        np.matmul(np.array(v[i].T).reshape(1,m−i),QTb[i:])))
return QTb
```

Testing

Testing Correctness

To test the correctness of the implemented algorithm we compare our results with the python library *numpy* by generating random matrices of all possible sizes of $m \times n$ where $2 \leq m, n \leq 100$ and checking to see that the norm of the difference of the two matrices is zero.

We notice the following:

- All Q matrices are accurate up to 10^{-13}
- All R matrices are accurate up to 4×10^{-4}

Runtime Analysis

Theoretical Runtime Analysis

At step k of the algorithm we call the Householder function which has negligible complexity and we focus on the computation of

$$A_{k:m,k:n} = A_{k:m,k:n} - 2v_k(v_k^T A_{k:m,k:n}).$$

Notice that this is derived from applying the k^{th} Householder matrix H_k to A:

$$\begin{aligned}
H_k A &= \left(I - 2\begin{bmatrix} 0 & v_k \end{bmatrix}^T \begin{bmatrix} 0 & v_k \end{bmatrix}\right) A \\
&= A - 2\begin{bmatrix} 0 & v_k \end{bmatrix}^T \begin{bmatrix} 0 & v_k \end{bmatrix} A \\
&= A - 2\begin{bmatrix} 0 & 0 \\ 0 & v_k v_k^T \end{bmatrix} \begin{bmatrix} A_{1:k-1,1:k-1} & A_{1:k-1,k:n} \\ A_{k:m,1:k-1} & A_{k:m,k:n} \end{bmatrix} \\
&= A - 2\begin{bmatrix} 0 & 0 \\ 0 & v_k v_k^T A_{k:m,k:n} \end{bmatrix}
\end{aligned} \tag{4}$$

Theoretical Runtime Analysis

Examining $A_{k:m,k:n} = A_{k:m,k:n} - 2v_k(v_k^T A_{k:m,k:n})$ we make the following observations on the computation:

- Inside parenthesis multiplication: $v_k^T A_{k:m,k:n}$
  $(m-k+1)(n-k+1) + (m-k)(n-k+1) = (2m-2k+1)(n-k+1)$ arithmetic operations
- Outside parenthesis multiplication: $v_k(v_k^T A_{k:m,k:n})$
  $(m-k+1)(n-k+1)$ arithmetic operations
- Subtractions $A_{k:m,k:n} - 2v_k(v_k^T A_{k:m,k:n})$
  $(m-k+1)(n-k+1)$ arithmetic operations

Theoretical Runtime Analysis

Setting r as

$$r = \begin{cases} n-1 & \text{if } m = n, \\ n & \text{if } m > n, \\ m & \text{if } m < n. \end{cases}$$

the number of approximate arithmetic operations is:

$$\begin{aligned} \sum_{k=1}^{r} 4(m-k+1)(n-k+1) &\approx \int_0^r 4(m-t)(n-t)\,dt \\ &= \frac{2r}{3}(6mn - 3mr - 3nr + 2r^2) \end{aligned} \tag{5}$$

Theoretical Runtime Analysis

A QR factorization of an overdetermined matrix $A_{m \times n} {}_{(m \geq n)}$ is

$$\mathcal{O}\left(mn^2\right)$$

Experiment and Report

The runtime experiment is set as follows:

- Time starts the line before QR function is called.
- Time ends the line after QR function is called.
- Random matrices $A_{m \times n}$ were generated with $m \geq n$, from $1000 \leq m \leq 3400$ and $500 \leq n \leq 2400$.
- The range of mn^2 was between $250, 000, 000$ and $16, 456, 000, 000$, taking from 6.36 seconds to 816.51 seconds to run QR.

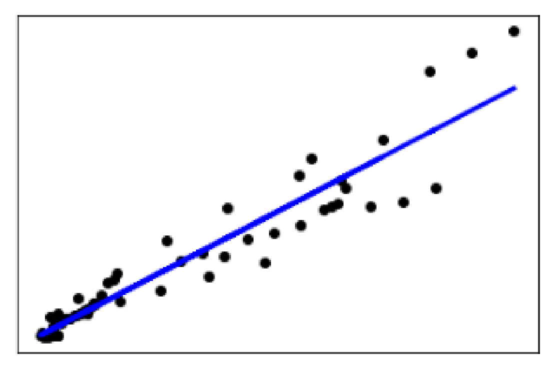


Figure 3: mn^2 vs. time plot in seconds. The blue line represents an average.

We note that the data is relatively linear and thus is correlating with our theoretical run-time.

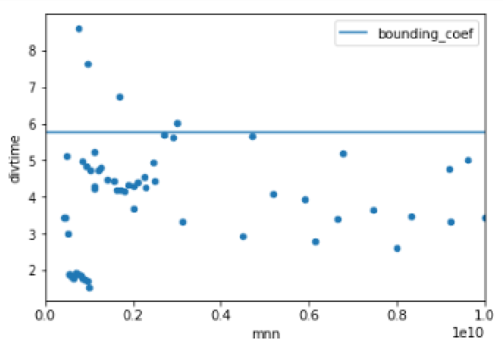


Figure 4: mn^2 vs. $(time/mn^2) \times 10^8$

Here we can note that as we approach *very large sizes* of mn^2, all data is bounded by the blue line. This represents the our bounding coefficient. This number is 5.75×10^{-8}.

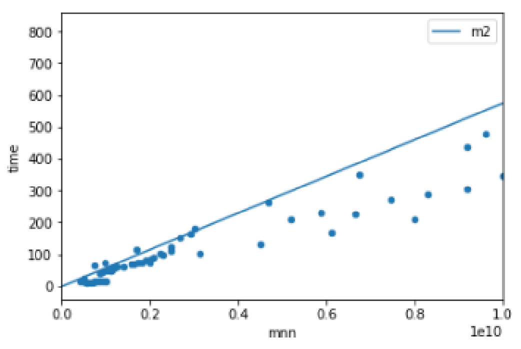


Figure 5: mn^2 vs. time plot in seconds. The blue line $m2$ denotes the line $y = 5.75 \times 10^{-8}x$ of which the data is bounded beneath.

Thus, we can conclude that the constant hidden in our Big-Oh notation is 5.75×10^{-8}.

Conclusion

Some final remarks:

- NEVER compute A^{-1} when writing software!
- QR decomposition via Householder transformations is the simplest of all numerically stable QR factorization algorithms since reflections are used mechanically for producing zeroes in the upper-trapezoidal matrix R.
- However, the disadvantage is that decomposition via Householder transformations is not easily parallelisable. (Can you see why?)

Test Questions

Test Questions

1. Why do we use $-sign(y_1)\|y\|e_1$ instead of $sign(y_1)\|y\|e_1$ in the geometrical derivation?
2. Is the factorization produced by Householder's algorithm unique? Discuss.
3. Why is the use of Householder transformations not [easily] parallelisable?

References

- QR decomposition, Wikipedia
- Bindel, Matrix Computations (CS 6210) (Fall 2009) Lecture Notes
- L. Vandenberghe EE133A (Spring 2017) Lecture Notes

Questions?