STAT 30900: Homework 3

Caleb Derrickson

November 11, 2023

Collaborators: The TA's of the class, as well as Kevin Hefner, Alexander Cram, Zach Hempstead, and Daniel Chen.

Contents

Problem 1

Given a symmetric $A \in \mathbb{R}^{n\times n}$, $\mathbf{0} \neq \mathbf{x} \in \mathbb{R}^n$, and $\mathbf{b} \in \mathbb{R}^n$. Let

$$\mathbf{r} = \mathbf{b} - A\mathbf{x}$$

Consider the QR decomposition

$$[\mathbf{x}, \mathbf{r}] = QR$$

and observe that if $E\mathbf{x} = \mathbf{r}$, then

$$(Q^T EQ)(Q^T\mathbf{x}) = Q^T\mathbf{r}.$$

Show how to compute a symmetric $E \in \mathbb{R}^{n\times n}$ so that is attains

$$\min_{(A+E)\mathbf{x}=\mathbf{b}} \|E\|_F$$

where the minimum is taken over all symmetric E.

Solution:

We are given $[\mathbf{x}, \mathbf{r}] = QR$, which when multiplied on the left by Q^T implies $[Q^T\mathbf{x}, Q^T\mathbf{r}] = Q^TQR = R$. Note that R is upper triangular, so $Q^T\mathbf{x}$ and $Q^T\mathbf{r}$ will have one and at most two nonzero elements, respectively. We are given the relation between them in the form $(Q^TEQ)(Q^T\mathbf{x}) = Q^T\mathbf{r}$ if $E\mathbf{x} = \mathbf{r}$. Then, we can break this into cases based on the number of elements in $Q^T\mathbf{r}$.

1. Case 1: $Q^T\mathbf{r}$ has one nonzero element.

   Denote $\Gamma = Q^TEQ$. Then by the relation given,

$$\Gamma(Q^T\mathbf{x}) = \begin{bmatrix} \cdot & \cdots & \cdot \\ \vdots & \ddots & \vdots \\ \cdot & \cdots & \cdot \end{bmatrix} \begin{bmatrix} | \\ Q^T\mathbf{x} \\ | \end{bmatrix} = \begin{bmatrix} \alpha \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} \Gamma_{11}(Q^T\mathbf{x})_1 + \Gamma_{12}(Q^T\mathbf{x})_2 + \ldots \\ \Gamma_{21}(Q^T\mathbf{x})_1 + \Gamma_{22}(Q^T\mathbf{x})_2 + \ldots \\ \vdots \end{bmatrix}$$

   Note that the only nonzero element in $Q^T\mathbf{r}$ is the first element, so $\Gamma_{j1}(Q^T\mathbf{x})_1 + \Gamma_{j2}(Q^T\mathbf{x})_2 + \ldots = 0$ for $j \neq 1$. The only nonzero element in Q^Tx is the first, so any terms not involving $Q^T\mathbf{x})_1$ is zero. so $\Gamma_{j1}, \Gamma_{j2}, \ldots, \Gamma_{jn}$ are arbitrary for $j \neq 1$. Since γ is symmetric, then $\Gamma_{12}, \Gamma_{13}, \ldots, \Gamma_{1n}$ are also arbitrary. Then the only non-arbitrary element in Γ is Γ_{11}. Thus, $\min \|E\|_F = \min \left\|Q^TEQ\right\|_F = |\Gamma_{11}|$ (since all other elements are arbitrary, the minimum is attained when they are 0). Note the second equality holds since the Frobenius norm is unitary invariant.

2. Case 2: $Q^T\mathbf{r}$ has two nonzero elements.

   By the same logic in the previous case,

$$\Gamma(Q^T\mathbf{x}) = \begin{bmatrix} \cdot & \cdots & \cdot \\ \vdots & \ddots & \vdots \\ \cdot & \cdots & \cdot \end{bmatrix} \begin{bmatrix} | \\ Q^T\mathbf{x} \\ | \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} \Gamma_{11}(Q^T\mathbf{x})_1 + \Gamma_{12}(Q^T\mathbf{x})_2 + ... \\ \Gamma_{21}(Q^T\mathbf{x})_1 + \Gamma_{22}(Q^T\mathbf{x})_2 + ... \\ \vdots \\ \end{bmatrix}$$

   Then the only possible non arbitrary terms are $\Gamma_{11}(Q^T\mathbf{x})_1, \Gamma_{12}(Q^T\mathbf{x})_2, \Gamma_{21}(Q^T\mathbf{x})_1, \Gamma_{22}(Q^T\mathbf{x})_2$ by the same logic in the previous case. Note that $(Q^T\mathbf{x})_j = 0$ for $j \neq 1$. Then, $\Gamma_{22}(Q^T\mathbf{x})_2$ equals 0 for any value of Γ_{22}, making it arbitrary. Note that $(Q^T\mathbf{r})_2$ is nonzero, so this must mean $\Gamma_{21}(Q^T\mathbf{x})_1 \neq 0$, meaning $\Gamma_{21} \neq 0$. Since Γ is symmetric, this means $\Gamma_{12} \neq 0$. Thus the only non arbitrary elements of Γ are $\Gamma_{11}, \Gamma_{12}, \Gamma_{21}$. Therefore, $\min \|E\|_F^2 = \min \left\|Q^T E Q\right\|_F = |\Gamma_{11}| + |\Gamma_{12}| + |\Gamma_{21}|$.

Problem 2

Let $A \in \mathbb{R}^{m\times n}$ and suppose the complete orthogonal decomposition is given by

$$A = Q_1 \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} Q_2^T$$

where Q_1 and Q_2 are orthogonal, and L is a nonsingular lower triangular matrix. Recall that $X \in \mathbb{R}^{n\times m}$ is the unique pseudo-inverse of A is the following Moore-Penrose conditions hold:

(i) $AXA = A$

(ii) $XAX = X$

(iii) $(AX)^T = AX$

(iv) $(XA)^T = XA$

and in which case we write $A^\dagger = X$.

Problem 2, part a

Let

$$A^- = Q_2 \begin{bmatrix} L^{-1} & Y \\ 0 & 0 \end{bmatrix} Q_1^T, \qquad Y \neq 0.$$

Which of the four conditions (i) - (iv) are satisfied?

Solution:

We will go straight into calculations.

$$i)\ AXA = Q_1 \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} Q_2^T Q_2 \begin{bmatrix} L^{-1} & Y \\ 0 & 0 \end{bmatrix} Q_1^T Q_1 \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} Q_2^T \qquad \text{(Given.)}$$

$$= Q_1 \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} L^{-1} & Y \\ 0 & 0 \end{bmatrix} \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} Q_2^T \qquad \text{(Q_1, Q_2 are orthogonal.)}$$

$$= Q_1 \begin{bmatrix} \mathbb{I} & LY \\ 0 & 0 \end{bmatrix} \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} Q_2^T \qquad \text{(Matrix Multiplication.)}$$

$$= Q_1 \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} Q_2^T \qquad \text{(Matrix Multiplication.)}$$

$$= A \qquad \text{(By Definition.)}$$

$$ii)\ XAX = Q_2 \begin{bmatrix} L^{-1} & Y \\ 0 & 0 \end{bmatrix} Q_1^T Q_1 \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} Q_2^T Q_2 \begin{bmatrix} L^{-1} & Y \\ 0 & 0 \end{bmatrix} Q_1^T \qquad \text{(Given.)}$$

$$= Q_2 \begin{bmatrix} L^{-1} & Y \\ 0 & 0 \end{bmatrix} \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} L^{-1} & Y \\ 0 & 0 \end{bmatrix} Q_1^T \qquad \text{(}Q_1, Q_2\text{ are orthogonal.)}$$

$$= Q_2 \begin{bmatrix} \mathbb{I} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} L^{-1} & Y \\ 0 & 0 \end{bmatrix} Q_1^T \qquad \text{(Matrix Multiplication.)}$$

$$= Q_2 \begin{bmatrix} L^{-1} & Y \\ 0 & 0 \end{bmatrix} Q_1^T \qquad \text{(Matrix Multiplication.)}$$

$$= X \qquad \text{(By Definition.)}$$

$$iii)\ (AX)^T = \left(Q_1 \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} Q_2^T Q_2 \begin{bmatrix} L^{-1} & Y \\ 0 & 0 \end{bmatrix} Q_1^T \right)^T \qquad \text{(Given.)}$$

$$= \left(Q_1 \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} L^{-1} & Y \\ 0 & 0 \end{bmatrix} Q_1^T \right)^T \qquad \text{(}Q_2\text{ is orthogonal.)}$$

$$= \left(Q_1 \begin{bmatrix} \mathbb{I} & LY \\ 0 & 0 \end{bmatrix} Q_1^T \right)^T \qquad \text{(Matrix multiplication.)}$$

$$= Q_1 \begin{bmatrix} \mathbb{I} & 0 \\ LY & 0 \end{bmatrix} Q_1^T \qquad \text{(Transposition.)}$$

$$\neq AX \qquad \text{(As can be seen.)}$$

$$iv)\ (XA)^T = \left(Q_2 \begin{bmatrix} L^{-1} & Y \\ 0 & 0 \end{bmatrix} Q_1^T Q_1 \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} Q_2^T \right)^T \qquad \text{(Given.)}$$

$$= \left(Q_2 \begin{bmatrix} L^{-1} & Y \\ 0 & 0 \end{bmatrix} \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} Q_2^T \right)^T \qquad \text{(}Q_1\text{ is orthogonal.)}$$

$$= \left(Q_2 \begin{bmatrix} \mathbb{I} & 0 \\ 0 & 0 \end{bmatrix} Q_2^T \right)^T \qquad \text{(Matrix Multiplication.)}$$

$$= Q_2 \begin{bmatrix} \mathbb{I} & 0 \\ 0 & 0 \end{bmatrix} Q_2^T \qquad \text{(Transposition.)}$$

$$= XA \qquad \text{(As can be seen.)}$$

Thus, we see that (i), (ii), and (iv) hold, but not (iii).

Problem 2, part b

Prove that

$$A^\dagger = Q_2 \begin{bmatrix} L^{-1} & 0 \\ 0 & 0 \end{bmatrix} Q_1^T$$

by letting

$$A^\dagger = Q_2 \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} Q_1^T$$

and by completing the following steps

- Using (i), prove that $X_{11} = L^{-1}$.
- Using the symmetry conditions (iii) and (iv), prove that $X_{12} = X_{21} = 0$.
- Using (ii), prove that $X_{22} = 0$.

Solution:

Here we are letting the middle term take on any form (within reason), then arguing by the properties of the Moore-Penrose pseudo-inverse, that it must take this form. Then, let

$$A^\dagger = Q_2 \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} Q_1^T$$

We will then recover terms by the above properties in the suggested order.

- By the first property, $AXA = A$ must be obeyed. Then,

$$\begin{aligned}
AA^\dagger A &= Q_1 \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} Q_2^T Q_2 \begin{bmatrix} X_{11} & x_{12} \\ X_{21} & X_{22} \end{bmatrix} Q_1^T Q_1 \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} && \text{(Given.)} \\
&= Q_1 \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} X_{11} & x_{12} \\ X_{21} & X_{22} \end{bmatrix} \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} && \text{(}Q_1, Q_2\text{ are orthogonal.)} \\
&= Q_1 \begin{bmatrix} LX_{11} & LX_{12} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} Q_2^T && \text{(Matrix multiplication.)} \\
&= Q_1 \begin{bmatrix} LX_{11}L & 0 \\ 0 & 0 \end{bmatrix} Q_2^T && \text{(Matrix multiplication.)} \\
\implies LX_{11}L &= L && (AA^\dagger A = A.) \\
\iff X_{11} &= L^{-1} && \text{(}L\text{ is nonsingular.)}
\end{aligned}$$

- Next, we will take the symmetric properties of the Moore-Penrose pseudo inverse.

$$AA^\dagger = \left(Q_1 \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} Q_2^T Q_2 \begin{bmatrix} L^{-1} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} Q_1^T\right)^T \qquad \text{(Given.)}$$

$$= \left(Q_1 \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} L^{-1} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} Q_1^T\right)^T \qquad \text{(Q_2 is orthogonal.)}$$

$$= \left(Q_1 \begin{bmatrix} \mathbb{I} & LX_{12} \\ 0 & 0 \end{bmatrix} Q_1^T\right)^T \qquad \text{(Matrix multiplication.)}$$

$$\Longrightarrow \; LX_{12} = L \qquad \text{(Transposition and $(AA^\dagger)^T = AA^\dagger$.)}$$

$$\Longrightarrow \quad X_{12} = 0 \qquad \text{(L is nonsingular, thus nonzero.)}$$

$$(A^\dagger A)^T = \left(Q_2 \begin{bmatrix} L^{-1} & 0 \\ X_{21} & X_{22} \end{bmatrix} Q_1^T Q_1 \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} Q_2^T\right)^T \qquad \text{(Given.)}$$

$$= \left(Q_2 \begin{bmatrix} L^{-1} & 0 \\ X_{21} & X_{22} \end{bmatrix} \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} Q_2^T\right)^T \qquad \text{(Q_1 is orthogonal.)}$$

$$= \left(Q_2 \begin{bmatrix} \mathbb{I} & 0 \\ X_{21}L & 0 \end{bmatrix} Q_2^T\right)^T \qquad \text{(Matrix multiplication.)}$$

$$\Longrightarrow \; X_{21}L = 0 \qquad \text{(Transposition and $(A^\dagger A)^T = A^\dagger A$.)}$$

$$\Longrightarrow \quad X_{21} = 0 \qquad \text{(L is invertible thus nonzero.)}$$

- Finally, we will show $X_{22} = 0$.

$$A^\dagger A A^\dagger = Q_2 \begin{bmatrix} L^{-1} & 0 \\ 0 & X_{22} \end{bmatrix} Q_1^T Q_1 \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} Q_2^T Q_2 \begin{bmatrix} L^{-1} & 0 \\ 0 & X_{22} \end{bmatrix} Q_1^T \qquad \text{(Given.)}$$

$$= Q_2 \begin{bmatrix} L^{-1} & 0 \\ 0 & X_{22} \end{bmatrix} \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} L^{-1} & 0 \\ 0 & X_{22} \end{bmatrix} Q_1^T \qquad \text{(Q_1, Q_2 are orthogonal.)}$$

$$= Q_2 \begin{bmatrix} \mathbb{I} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} L^{-1} & 0 \\ 0 & X_{22} \end{bmatrix} Q_1^T \qquad \text{(Matrix multiplication.)}$$

$$= Q_2 \begin{bmatrix} L^{-1} & 0 \\ 0 & 0 \end{bmatrix} Q_1^T \qquad \text{(Matrix multiplication.)}$$

$$\implies X_{22} = 0 \qquad (A^\dagger A A^\dagger = A^\dagger.)$$

Therefore, $A^\dagger$ is of the given form. Note that I passed over some steps, including steps where I equated entries only in the block matrix, ignoring the factors which would appear when multiplied by Q_i for its transpose. Since they are orthonormal however, these factors will cancel out, leaving us only with the entry in the block matrix.

Problem 3

Let $A \in \mathbb{R}^{m\times n}$, $\mathbf{b} \in \mathbb{R}^m$, and $\mathbf{c} \in \mathbb{R}^n$. We are interested in the least squares problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|A\mathbf{x} - \mathbf{b}\|_2^2 \tag{1}$$

Problem 3, part a

Show that $\mathbf{x}$ is a solution to (1) if and only if $\mathbf{x}$ is a solution to the *augmented system*

$$\begin{bmatrix} \mathbb{I} & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{r} \\ \mathbf{x} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ 0 \end{bmatrix} \tag{2}$$

Solution:

- $\underline{\implies)}$: Suppose (1) $\implies$ (2) were false, that is, $\mathbf{x}$ is not a solution to the augmented system. Note that since $\mathbf{x}$ is a solution to (1), then $A\mathbf{x} - \mathbf{b} = \mathbf{d}$, for some $\mathbf{d}$. Note that $\mathbf{d}$ is in general nonzero. For simplicity, we will break this proof down into cases.

  - <u>Case 1:</u> $\mathbf{b} \in \text{im}\,(A)$.

    Then the minimum of $\|A\mathbf{x} - \mathbf{b}\|$ would equal zero, since $\|A\mathbf{x}$ - $\mathbf{b}\| = \|A(\mathbf{x} - \mathbf{y})\| = 0$, which attains minimum when $\mathbf{x} = \mathbf{y}$, meaning $A\mathbf{x} = \mathbf{b}$. Then $\mathbf{d} = 0$. Then (2) simplifies to

    $$\begin{bmatrix} \mathbb{I} & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{0} \\ \mathbf{x} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ 0 \end{bmatrix}.$$

    This implies $A\mathbf{x} = b$ and $A^T\mathbf{0} = \mathbf{0}$. Since these are both true, then we run into a contradiction.

  - <u>Case 2:</u> $\mathbf{b} \notin \text{im}\,(A)$

    Then $\|A\mathbf{x} - b\| \neq 0$. This would mean $\mathbf{d} \neq 0$, and $\mathbf{d} \neq \text{im}(A)$ since if it were, then we can write $\mathbf{d} = A\mathbf{y}$, then $\|A(\mathbf{x} - \mathbf{y}) - b\| = 0$, which means $\mathbf{x}-\mathbf{y}$ is a solution to (1), meaning $\mathbf{x}$ is not a minimum, which is a contradiction. Since $\mathbf{d} \notin \text{im}\,(A)$, then $\mathbf{d} \in \ker(A^T)$ by the Fredholm Alternative. Then (2) is equivalent to

    $$\begin{bmatrix} \mathbb{I} & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{d} \\ \mathbf{x} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ 0 \end{bmatrix}.$$

    This implies $A\mathbf{x} + \mathbf{d} = \mathbf{b}$ and $A^T\mathbf{d} = 0$. This is is true, since we only know $\mathbf{d}$ up to sign. Thus (2) holds, giving us a contradiction.

- $\underline{\impliedby)}$:

  Note that (2) is equivalent to $A\mathbf{x} + \mathbf{b} = \mathbf{r}$ and $A^T\mathbf{r} = 0$. We will again break this down into cases.

- Case 1: $\mathbf{b} \in \operatorname{im}(A)$.

  Then $\mathbf{b} = A\mathbf{y}$ for some $\mathbf{y} \in \mathbb{R}^n$. Thus $\mathbf{r} = A\mathbf{x}+\mathbf{b} = A(\mathbf{x}-\mathbf{y})$. Thus when taking the minimum over all $\mathbf{x}$, we see that $\|A(\mathbf{x}-\mathbf{y})\| = 0$, exactly when $\mathbf{x} = \mathbf{y}$. Since $\|\cdot\|_2$ is positive, then $\mathbf{x} \in \operatorname{argmin}(\|A\mathbf{x}-\mathbf{b}\|)$, which means $\mathbf{x}$ is a solution to (1).

- Case 2: $\mathbf{b} \notin \operatorname{im}(A)$.

  Then, by the Fredholm Alternative, $\mathbf{b} \in \ker(A^T)$. Thus, the following can be shown:

$$\begin{aligned}
\|\mathbf{r}\|_2^2 = \|A\mathbf{x}-\mathbf{b}\|_2^2 &= (A\mathbf{x}-\mathbf{b})^T(A\mathbf{x}-\mathbf{b}) && \text{(2-norm, given.)} \\
&= \mathbf{x}^T A^T A\mathbf{x} - \mathbf{x}^T A^T \mathbf{b} - \mathbf{b}^T A\mathbf{x} + \mathbf{b}^T\mathbf{b} && \text{(Factoring out.)} \\
&= \mathbf{x}^T A^T A\mathbf{x} - \mathbf{b}^T A\mathbf{x} + \mathbf{b}^T\mathbf{b} && (\mathbf{b} \in \ker(A^T).) \\
&= \mathbf{x}^T A^T A\mathbf{x} - (A^T\mathbf{b})^T\mathbf{x} + \mathbf{b}^T\mathbf{b} && \text{(Associativity.)} \\
&= \mathbf{x}^T A^T A\mathbf{x} + \mathbf{b}^T\mathbf{b} && (\mathbf{b} \in \ker(A^T).) \\
&= \|A\mathbf{x}\|_2^2 + \|\mathbf{b}\|_2^2 && \text{(2-norm definition.)}
\end{aligned}$$

  This then means that $\mathbf{x}$ satisfies the Pythagorean Theorem, implying that $\mathbf{x}$ is a solution to (1).

Problem 3, part b

Show that the $(m+n) \times (m+n)$ matrix in (2) is nonsingular if and only if A has full column rank.

Solution:

Denote $\mathbb{A}$ to be the $(m+n) \times (m+n)$ matrix in (2). We will again break this down into cases.

- $\underline{\implies)}$:

  Suppose false, that is, $\mathbb{A}$ is singular, but A has full column rank. Note this means A has full rank. Since $\mathbb{A}$ is singular, then $\exists \mathbf{v} \in \mathbb{R}^{m+n} \setminus \{\mathbf{0}\}$ which is mapped to $\mathbf{0}$ under $\mathbb{A}$. Let $\mathbf{v} = [\mathbf{v_r}, \mathbf{v_x}]^T$. This then means, by (2), that $\mathbf{v_r} + A\mathbf{v_x} = 0$ and $A^T\mathbf{v_r} = 0$. We can substitute the first equation into the second to get $A^T A\mathbf{v_x} = 0$. Note that if A is full rank, then A^T is also full rank. This means that $A\mathbf{v_x} \in \ker(A^T)$. Since A^T has full rank, this means $\mathbf{v_x} \in \ker(A)$, which means $\mathbf{v_x} = 0$, since A is full rank. Then, since $A\mathbf{v_x} = -\mathbf{v_r}, \implies \mathbf{v_r} = 0$ as a result. Note we assumed that $\mathbf{v} \neq 0$, which is a contradiction, since $\mathbf{v}$ was found to only be zero.

- $\underline{\impliedby)}$:

  This is equivalent to showing $\det(\mathbb{A}) \neq 0$. Note that in general, a 2×2 block matrix has determinant

$$\det\left(\begin{bmatrix} A & B \\ C & D \end{bmatrix}\right) = \det(A)\det\left(D - CA^{-1}B\right)$$

  With $\mathbb{A}$, this then means $\det(\mathbb{A}) = \det(\mathbb{I})\det\left(0 - A^T(\mathbb{I})^{-1}A\right) = det(-A^T A) = (-1)^n \det\left(A^T A\right)$. Note that $A^T A$ has full rank when A has full rank. Furthermore, $A^T A$ is normal, thus its SVD decomposition coincides with an eigenvalue decomposition, with $\sigma^2 = \lambda$. since A has full rank, then all singular values of $A^T A$ are nonzero, thus all eigenvalues are nonzero, thus $\det\left(A^T A\right) \neq 0$. Therefore $\det(\mathbb{A}) \neq 0$ when A has full column rank.

Problem 3, part c

Suppose A has full column rank and the QR decomposition of A is

$$A = Q\begin{bmatrix} R \\ 0 \end{bmatrix}.$$

Show that he solution to the augmented system

$$\begin{bmatrix} \mathbb{I} & A \\ A^T & 0 \end{bmatrix}\begin{bmatrix} \mathbf{y} \\ \mathbf{x} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{c} \end{bmatrix}$$

can be computed from

$$\mathbf{x} = (R^{-1})^T\mathbf{c}, \qquad \begin{bmatrix} \mathbf{d}_1 \\ \mathbf{d}_2 \end{bmatrix} = Q^T\mathbf{b},$$

and

$$\mathbf{x} = R^{-1}(\mathbf{d}_1 - \mathbf{z}), \qquad \mathbf{y} = Q\begin{bmatrix} \mathbf{z} \\ \mathbf{d}_2 \end{bmatrix}.$$

Solution:

This is just an exercise in reverse engineering, and computation. We can recover the first equation in the following steps:

$$\begin{aligned}
Q^T\mathbf{b} &= \begin{bmatrix} \mathbf{d}_1 \\ \mathbf{d}_2 \end{bmatrix} && \text{(Given.)} \\
&= \begin{bmatrix} \mathbf{d}_1 + \mathbf{z} - \mathbf{z} \\ \mathbf{d}_1 \end{bmatrix} && \text{(Adding a zero.)} \\
&= \begin{bmatrix} \mathbf{z} \\ \mathbf{d}_2 \end{bmatrix} + \begin{bmatrix} \mathbf{d}_1 - \mathbf{z} \end{bmatrix} && \text{(Separating.)} \\
&= \begin{bmatrix} \mathbf{z} \\ \mathbf{d}_2 \end{bmatrix} + \begin{bmatrix} R \\ 0 \end{bmatrix} R^{-1}(\mathbf{d}_1 - \mathbf{z}) && (R^{-1}R = \mathbb{I}.) \\
&= Q^T\left(Q\begin{bmatrix} \mathbf{z} \\ \mathbf{d}_2 \end{bmatrix}\right) + \begin{bmatrix} R \\ 0 \end{bmatrix} R^{-1}(\mathbf{d}_1 - \mathbf{z}) && (Q^TQ = \mathbb{I}.) \\
\implies Q^T\mathbf{b} &= Q^T\mathbf{y} + \begin{bmatrix} R \\ 0 \end{bmatrix}\mathbf{x} && \text{(Given definitions.)}
\end{aligned}$$

$$\implies \mathbf{b} = \mathbf{y} + Q\begin{bmatrix} R \\ 0 \end{bmatrix}\mathbf{x} \qquad \text{(Multiplying by } Q\text{.)}$$

$$\implies \mathbf{b} = \mathbf{y} + A\mathbf{x} \qquad (QR \text{ of } A\text{.)}$$

The second equation can also be found similarly.

$$\mathbf{z} = (R^{-1})^T\mathbf{c} \qquad \text{(Given.)}$$

$$\implies \mathbf{z}^T = \mathbf{c}^T R^{-1} \qquad \text{(Transposition.)}$$

$$\implies \mathbf{c}^T = \mathbf{z}^T R \qquad \text{(Multiplying by } R\text{.)}$$

$$\implies \mathbf{c} = R^T\mathbf{z} \qquad \text{(Transposition.)}$$

$$= \begin{bmatrix} R^T & 0 \end{bmatrix}\begin{bmatrix} \mathbf{z} \\ \mathbf{d}_2 \end{bmatrix} \qquad \text{(Block Equivalence, element of } \mathbf{d}_2 \text{ could be anything.)}$$

$$= \begin{bmatrix} R^T & 0 \end{bmatrix} Q^T Q\begin{bmatrix} \mathbf{z} \\ \mathbf{d}_2 \end{bmatrix} \qquad (Q^TQ = \mathbb{I}\text{.)}$$

$$= \left(Q\begin{bmatrix} R \\ 0 \end{bmatrix}\right)^T Q\begin{bmatrix} \mathbf{z} \\ \mathbf{d}_2 \end{bmatrix} \qquad \text{(Transposition.)}$$

$$\implies \mathbf{y} = A^T\mathbf{y} \qquad \text{(Given definitions.)}$$

Problem 3, part d

Hence deduce that if A has full column rank, then

$$A^\dagger = R^{-1}Q_1^T$$

where $Q = [Q_1, Q_2]$ with $Q_1 \in \mathbb{R}^{m\times n}$ and $Q_2 \in \mathbb{R}^{m\times(m-n)}$. Check this agrees with the general formula derived for a rank-retaining factorization $A = GH$ in the lectures.

Solution:

We first show that the given $A^\dagger$ satisfies the properties of the Moore-Penrose pseudo-inverse:

$$\begin{aligned}
i)\ AA^\dagger A &= (Q\begin{bmatrix} R \\ 0 \end{bmatrix} R^{-1}Q_1^T)A && \text{(Given.)} \\
&= \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix}\begin{bmatrix} R \\ 0 \end{bmatrix} R^{-1}Q_1^T)A && \text{(Definitions.)} \\
&= Q_1RR^{-1}Q_1^TA && \text{(Matrix multiplication.)} \\
&= (Q_1Q_1^T)A && (R^{-1}R = \mathbb{I}.) \\
&= (Q_1Q_1^T)\begin{bmatrix} Q_1 & Q_2 \end{bmatrix}\begin{bmatrix} R \\ 0 \end{bmatrix} && \text{(Writing out.)} \\
&= Q_1Q_1^TQ_1R && \text{(Matrix Multiplication.)} \\
&= Q_1R && \text{(Q_1 has orthonormal columns.)} \\
ii)\ A^\dagger AA^\dagger &= R^{-1}Q_1^T\begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix}\begin{bmatrix} R \\ 0 \end{bmatrix}A^\dagger && \text{(Given.)} \\
&= R^{-1}Q_1^TQ_1RA^\dagger && \text{(Matrix multiplication.)} \\
&= (\mathbb{I})A^\dagger && \text{(Q_1 has orthonormal columns, $R^{-1}R = \mathbb{I}$.)} \\
iii)\ (A^\dagger A)^T &= \left(R^{-1}Q_1^T\begin{bmatrix} Q_1 & Q_2 \end{bmatrix}\begin{bmatrix} R \\ 0 \end{bmatrix}\right)^T && \text{(By definition.)}
\end{aligned}$$

$$= (R^{-1}Q_1^TQ_1R)^T \qquad \text{(Matrix multiplication.)}$$

$$= (R^{-1}R)^T \qquad \text{(Q_1 has orthonormal columns.)}$$

$$= \mathbb{I} \qquad (R^{-1}R = \mathbb{I})$$

$$iv)\ (AA^\dagger)^T = \left(\begin{bmatrix} Q_1 & Q_2 \end{bmatrix}\begin{bmatrix} R \\ 0 \end{bmatrix} R^{-1}Q_1^T\right)^T \qquad \text{(Given.)}$$

$$= \left(Q_1RR^{-1}Q_1^T\right)^T \qquad \text{(Matrix multiplication.)}$$

$$= \left(Q_1Q_1^T\right)^T \qquad (R^{-1}R = \mathbb{I}.)$$

Note that I have to be careful in handling the Q subspaces. Since Q had orthonormal columns, $Q_1^TQ_1 = \mathbb{I}$, but $Q_1Q_1^T \neq \mathbb{I}$ in general. However, $(Q_1Q_1^T)^T = Q_1Q_1^T$, Thus (iv) will hold. This issue also pops ups in (i), but problems subside once we multiply on the right by A. Thus $(i) - (iv)$ hold. We need to show that this then agrees with the rank-retaining lectures as shown in class. That is,

$$A^\dagger = H^T(HH^T)^{-1}(G^TG)^{-1}G^T$$

We need to be careful in our choice of G and H, since if $H = \begin{bmatrix} R \\ 0 \end{bmatrix}$, then

$$HH^T = \begin{bmatrix} R \\ 0 \end{bmatrix}\begin{bmatrix} R^T & 0 \end{bmatrix} = \begin{bmatrix} RR^T & 0 \\ 0 & 0 \end{bmatrix},$$

which is not invertible. Thus taking this with respect to the *condensed QR* decomposition, then take $H = R, G = Q_1$. Then the following steps are justified:

$$A^\dagger = H^T(HH^T)^{-1}(G^TG)^{-1}G^T \qquad \text{(Given.)}$$

$$= R^T(RR^T)^{-1}(Q_1^TQ_1)^{-1}Q_1^T \qquad \text{(Plugging in.)}$$

$$= R^T(R^T)^{-1}R^{-1}(\mathbb{I})^{-1}Q_1^T \qquad ((AB)^{-1} = B^{-1}A^{-1} \text{ if defined.})$$

$$= R^{-1}Q_1^T \qquad\qquad (R^T(R^T)^{-1} = \mathbb{I}.)$$

Thus, we find our $A^\dagger$ to coincide with the one found using the general formula for a rank-retaining factorization.

Problem 4

Let $A \in \mathbb{R}^{m\times n}$. Suppose we apply QR factorization with column pivoting to obtain the decomposition

$$A = Q\begin{bmatrix} R & S \\ 0 & 0 \end{bmatrix}\Pi^T$$

where Q is orthogonal and R is upper triangular and invertible. Let $\mathbf{x}_B$ be the *basic solution*, i.e.,

$$\mathbf{x}_B = \Pi\begin{bmatrix} R^{-1} & 0 \\ 0 & 0 \end{bmatrix}Q^T\mathbf{b},$$

and let $\hat{\mathbf{x}} = A^{\dagger}\mathbf{b}$. Show that

$$\frac{\|\mathbf{x}_B - \hat{\mathbf{x}}\|_2}{\|\hat{\mathbf{x}}\|_2} \leq \left\|R^{-1}S\right\|_2.$$

Solution:

Note the form of $\|A\mathbf{x} - \mathbf{b}\|_2^2$ is equal to:

$$\begin{aligned}
\|A\mathbf{x} - \mathbf{b}\| &= \left\|Q\begin{bmatrix} R & S \\ 0 & 0 \end{bmatrix}\Pi^T\mathbf{x} - \mathbf{b}\right\|_2^2 && \text{(Form given.)} \\
&= \left\|\begin{bmatrix} R & S \\ 0 & 0 \end{bmatrix}\begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} - Q^T\mathbf{b}\right\|_2^2 && \text{(Definition of } \Pi^T\mathbf{x} \text{ and unitary invariance.)} \\
&= \left\|\begin{bmatrix} R\mathbf{u} + S\mathbf{v} \\ 0 \end{bmatrix} - \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix}\right\|_2^2 && \text{(Matrix multiplication.)} \\
&= \|R\mathbf{u} + S\mathbf{v} - \mathbf{c}\|_2^2 + \|\mathbf{d}\|_2^2 && \text{(Simplifying, 2-norm definition.)}
\end{aligned}$$

I claim that $\|A\mathbf{x} - \mathbf{b}\|_2^2$ is equal to $\|\mathbf{d}\|_2^2$ for $\mathbf{x}_B$. I will show this below:

$$\begin{aligned}
\|A\mathbf{x}_B - \mathbf{b}\|_2^2 &= \left\|Q\begin{bmatrix} R & S \\ 0 & 0 \end{bmatrix}\Pi^T\Pi\begin{bmatrix} R^{-1} & 0 \\ 0 & 0 \end{bmatrix}Q^T\mathbf{b} - \mathbf{b}\right\|_2^2 && \text{(Plugging in } \mathbf{x}_B.\text{)} \\
&= \left\|\begin{bmatrix} R & S \\ 0 & 0 \end{bmatrix}\begin{bmatrix} R^{-1} & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} \mathbf{c} \\ \mathbf{c} \end{bmatrix} - \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix}\right\| && \text{(Simplifying.)}
\end{aligned}$$

$$= \left\| \begin{bmatrix} \mathbb{I}_r & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} - \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} \right\|_2^2 \qquad \text{(Matrix multiplication.)}$$

$$= \left\| \begin{bmatrix} \mathbf{c} \\ 0 \end{bmatrix} - \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} \right\|_2^2 \qquad \text{(Multiplication.)}$$

$$= \|\mathbf{d}\|_2^2 \qquad \text{(Simplification.)}$$

Therefore, all minimizers have $R\mathbf{u} + S\mathbf{v} - c = 0$. Thus $\hat{\mathbf{x}}$ has this property, since it is the minimum least squares solution to $\|A\mathbf{x} - \mathbf{b}\|_2$. Also, $\|\mathbf{x}\|_2^2 = \|\mathbf{u}\|_2^2 + \|\mathbf{v}\|_2^2$ for any $\mathbf{x} = \Pi \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix}$. Thus

$$\hat{\mathbf{x}} = \operatorname{argmin} \left(\|\mathbf{u}\|_2^2 + \|\mathbf{v}\|_2^2 \right), \quad \text{For } R\mathbf{u} + S\mathbf{v} = \mathbf{c}.$$

Define $\mathbf{y} = \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix}$. Consider the Lagrangian:

$$L(\mathbf{y}, \lambda) = \|\mathbf{y}\|_2^2 + 2\left(\begin{bmatrix} R & S \end{bmatrix} \mathbf{y} - \mathbf{c} \right)^T \lambda$$

Note the transpose needs to be there in the constraint term due to λ being a vector, since the co-domain of the Lagrangian is the real numbers. The Lagrangian thus gives the solutions:

$$\begin{cases} \nabla_{\mathbf{y}} L(\mathbf{y}, \lambda) = 2\mathbf{y} + 2 \begin{bmatrix} R^T \\ S^T \end{bmatrix} \lambda = 0 \\ \nabla_{\lambda} L(\mathbf{y}, \lambda) = 2(\begin{bmatrix} R & S \end{bmatrix} \mathbf{y} - \mathbf{c}) = 0 \end{cases}$$

Or, written more concisely,

$$\begin{cases} \mathbf{y} + (\begin{bmatrix} R & S \end{bmatrix})^T \lambda = 0 \\ \begin{bmatrix} R & S \end{bmatrix} \mathbf{y} = \mathbf{c} \end{cases}$$

This then flows naturally into the augmented system, in the spirit of Problem 3.

$$\begin{bmatrix} \mathbb{I} & \begin{bmatrix} R & S \end{bmatrix}^T \\ \begin{bmatrix} R & S \end{bmatrix} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{y} \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{c} \end{bmatrix}$$

Here, $A = \begin{bmatrix} R & S \end{bmatrix}^T$. We can then define the QR decomposition of A as

$$\begin{bmatrix} R^T \\ S^T \end{bmatrix} = Q_C \begin{bmatrix} R_c \\ 0 \end{bmatrix} = Q_{1c} R_c$$

Note that $\begin{bmatrix} R & S \end{bmatrix}^T$ will have a full-rank QR decomposition since R^T is invertible, thus has full column rank. Then by Problem 3c,

$$\Pi^T \hat{\mathbf{x}} = \mathbf{y} = Q_c \begin{bmatrix} R_c^{-T}\mathbf{c} \\ 0 \end{bmatrix} = Q_{1c} R_c^{-T}\mathbf{c},$$

Where it is understood that Q_{1c} has the first r columns of Q_c. Note that $Q_{1c} = \begin{bmatrix} R & S \end{bmatrix}^T R_c^{-1}$, thus this can be placed into our definition for $\hat{\mathbf{x}}$ to get:

$$\Pi^T \hat{\mathbf{x}} = \begin{bmatrix} R^T \\ S^T \end{bmatrix} R_c^{-1} R_c^{-T}\mathbf{c}.$$

Since $\Pi\hat{\mathbf{x}} = \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix}$, we can rewrite this as:

$$\begin{aligned}
\Pi\hat{\mathbf{x}} &= \begin{bmatrix} R^T \\ S^T \end{bmatrix} R_c^{-1} R_c^{-T}\mathbf{c} && \text{(Given.)} \\
\implies \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} &= \begin{bmatrix} R^T \\ S^T \end{bmatrix} R_c^{-1} R_c^{-T} && \text{(Definition of } \Pi\hat{\mathbf{x}}.) \\
\implies \mathbf{u} &= R^T R_c^{-1} R_c^{-T}\mathbf{c}, \\
\mathbf{v} &= S^T R_c^{-1} R_c^{-T}\mathbf{c} && \text{(Writing what we have.)} \\
\implies R^{-T}\mathbf{u} &= R_c^{-1} R_c^{-T}\mathbf{c} \\
\iff \hat{\mathbf{x}} &= \begin{bmatrix} \mathbf{u} \\ S^T R^{-T}\mathbf{u} \end{bmatrix} && \text{(Substitution of } R^{-T}\mathbf{u}.)
\end{aligned}$$

$\mathbf{x}_B$ can also be written in a similar form, that is,

$$\begin{aligned}
\mathbf{x}_B &= \Pi \begin{bmatrix} R^{-1} & 0 \\ 0 & 0 \end{bmatrix} Q^T \mathbf{b} && \text{(Given.)} \\
&= \Pi \begin{bmatrix} R^{-1} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} Q_1^T \\ Q_2^T \end{bmatrix} \mathbf{b} && \text{(Transpose of } Q \text{ into first } r \text{ columns.)} \\
&= \Pi \begin{bmatrix} R^{-1} Q_1^T \mathbf{b} \\ 0 \end{bmatrix} && \text{(Matrix Multiplication.)} \\
&= \Pi \begin{bmatrix} R^{-1} \mathbf{c} \\ 0 \end{bmatrix} && \text{(Definition of } \mathbf{c}\text{.)}
\end{aligned}$$

We can then write the difference between $\mathbf{x}_B$ and $\hat{\mathbf{x}}$ in the following manner:

$$\begin{aligned}
\Pi^T(\mathbf{x}_B - \hat{\mathbf{x}}) &= \begin{bmatrix} R^{-1}\mathbf{c} \\ 0 \end{bmatrix} - \begin{bmatrix} \mathbf{u} \\ (R^{-1}S)^T \mathbf{u} \end{bmatrix} && \text{(Both found above.)} \\
&= \begin{bmatrix} \mathbf{u} + R^{-1} S \mathbf{v} \\ 0 \end{bmatrix} - \begin{bmatrix} \mathbf{u} \\ (R^{-1}S)^T \mathbf{u} \end{bmatrix} && \text{(From constraint, } R\mathbf{u} + S\mathbf{v} = \mathbf{c}\text{.)} \\
&= \begin{bmatrix} R^{-1} S \mathbf{v} \\ (R^{-1}S)^T \mathbf{u} \end{bmatrix} && \text{(Simplifying.)} \\
\implies \|\mathbf{x}_B - \hat{\mathbf{x}}\|_2^2 &= \left\|R^{-1} S \mathbf{v}\right\|_2^2 + \left\|(R^{-1}S)^T \mathbf{u}\right\|_2^2 && \text{(Taking norm, unitary invariance.)} \\
&\le \left\|R^{-1} S\right\|_2^2 \left(\|\mathbf{u}\|_2^2 + \|\mathbf{v}\|_2^2\right) && \text{(2-norm consistency.)} \\
&= \left\|R^{-1} S\right\|_2^2 \|\Pi \hat{\mathbf{x}}\|_2^2 && \text{(Norm equivalence of } \Pi\hat{\mathbf{x}}\text{.)} \\
\iff \frac{\|\mathbf{x}_B - \hat{\mathbf{x}}\|_2^2}{\|\Pi \hat{\mathbf{x}}\|_2^2} &\le \left\|R^{-1} S\right\|_2^2 && \text{(Rearranging.)}
\end{aligned}$$

Thus, by square-rooting both sides, we get:

$$\frac{\|\mathbf{x}_B - \hat{\mathbf{x}}\|_2}{\|\hat{\mathbf{x}}\|_2} \leq \left\|R^{-1}S\right\|_2$$

Which is what we wanted to show. Sorry for the messy formatting; this was a hairy proof, and I'm not the best at formatting in the first place.

Problem 5

Let $\mathbf{u} \in \mathbb{R}^n, \mathbf{u} \neq \mathbf{0}$. A *Householder* matrix $H_{\mathbf{u}} \in \mathbb{R}^{n \times n}$ is defined by

$$H_{\mathbf{u}} = \mathbb{I} - \frac{2\mathbf{u}\mathbf{u}^T}{\|\mathbf{u}\|_2^2}$$

Problem 5, part a

Show that $H_{\mathbf{u}}$ is both symmetric and orthogonal.

Solution:

- Symmetry:

$$\begin{aligned}
(H_{\mathbf{u}})^T &= (\mathbb{I} - \frac{2}{\|\mathbf{u}\|_2^2}\mathbf{u}\mathbf{u}^T)^T && \text{(Given.)} \\
&= \mathbb{I}^T - \frac{2}{\|\mathbf{u}\|_2^2}(\mathbf{u}\mathbf{u}^T)^T && \text{(Transpose is linear.)} \\
&= \mathbb{I} - \frac{2}{\|\mathbf{u}\|_2^2}\mathbf{u}\mathbf{u}^T && \text{(}\mathbf{u}\mathbf{u}^T \text{ and } \mathbb{I} \text{ are symmetric.)} \\
&= H_{\mathbf{u}}
\end{aligned}$$

- Orthogonality:

  Note $H_{\mathbf{u}}^T = H_{\mathbf{u}}$, so we just need to show $H_{\mathbf{u}}^2 = \mathbb{I}$.

$$\begin{aligned}
H_{\mathbf{u}}^2 &= (\mathbb{I} - \frac{2}{\|\mathbf{u}\|_2^2}\mathbf{u}\mathbf{u}^T)(\mathbb{I} - \frac{2}{\|\mathbf{u}\|_2^2}\mathbf{u}\mathbf{u}^T) && \text{(Given.)} \\
&= \mathbb{I}^2 + \frac{4}{\|\mathbf{u}\|_2^4}\mathbf{u}\mathbf{u}^T\mathbf{u}\mathbf{u}^T - \frac{4}{\|\mathbf{u}\|_2^2}\mathbf{u}\mathbf{u}^T && \text{(Expanding.)} \\
&= \mathbb{I} + \frac{4}{\|\mathbf{u}\|_2^4}\mathbf{u}\|\mathbf{u}\|_2^2\mathbf{u}^T - \frac{4}{\|\mathbf{u}\|_2^2}\mathbf{u}\mathbf{u}^T && \text{(Definition of 2-norm.)} \\
&= \mathbb{I} + \frac{4}{\|\mathbf{u}\|_2^2}\mathbf{u}\mathbf{u}^T - \frac{4}{\|\mathbf{u}\|_2^2}\mathbf{u}\mathbf{u}^T && \text{(Simplifying.)} \\
&= \mathbb{I} && \text{(Simplifying.)}
\end{aligned}$$

Problem 5, part b

Show that for any $\alpha \in \mathbb{R}, \alpha \neq 0$,

$$H_{\alpha\mathbf{u}} = H_{\mathbf{u}}$$

In other words, $H_{\mathbf{u}}$ only depends on the "direction" of **u** and not on its "magnitude".

Solution:

We will go straight into calculations:

$$\begin{aligned}
H_{\alpha\mathbf{u}} &= \mathbb{I} - \frac{2}{\|\alpha\mathbf{u}\|_2^2}(\alpha\mathbf{u})(\alpha\mathbf{u})^T && \text{(Given.)} \\
&= \mathbb{I} - \frac{2}{|\alpha|^2\|\mathbf{u}\|_2^2}\alpha^2(\mathbf{u})(\mathbf{u})^T && \text{(Transpose is linear and norm definitions.)} \\
&= \mathbb{I} - \frac{2}{\|\mathbf{u}\|_2^2}(\mathbf{u})(\mathbf{u})^T && (\alpha^2 = |\alpha|^2.) \\
&= H_{\mathbf{u}} && \text{(Definition.)}
\end{aligned}$$

Problem 5, part c

In general, given a matrix $M \in \mathbb{R}^{n\times n}$ and a vector $\mathbf{x} \in \mathbb{R}^n$, computing the matrix-vector product $M\mathbf{x}$ requires n inner products - one for each row of M with $\mathbf{x}$. Show that $H_{\mathbf{u}}\mathbf{x}$ can be computed using only two inner products.

Solution:

We will go straight into calculations:

$$\begin{aligned}
H_{\mathbf{u}}\mathbf{x} &= (\mathbb{I} - \frac{2}{\|\mathbf{u}\|_2^2}\mathbf{u}\mathbf{u}^T)\mathbf{x} && \text{(Given.)} \\
&= \mathbb{I}\mathbf{x} - \frac{2}{\langle \mathbf{u}|\mathbf{u}\rangle}\mathbf{u}\mathbf{u}^T\mathbf{x} && \text{(2-norm definition and distribution.)} \\
&= \mathbf{x} - \frac{2}{\langle \mathbf{u}|\mathbf{u}\rangle}\mathbf{u}\,\langle \mathbf{u}|\mathbf{x}\rangle && \text{(Inner product definition.)}
\end{aligned}$$

As we can see we need to just calculate two inner products when doing the matrix-vector product $H_{\mathbf{u}}\mathbf{x}$, appearing in the 2-norm of $\mathbf{u}$ and the multiplication of the rank-1 matrix $\mathbf{u}\mathbf{u}^T$.

Problem 5, part d

Given $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$ where $\mathbf{a} \neq \mathbf{b}$ and $||\mathbf{a}||_2 = ||\mathbf{b}||_2$. Find $\mathbf{u} \in \mathbb{R}^n, \mathbf{u} \neq 0$ such that

$$H_{\mathbf{u}}\mathbf{a} = \mathbf{b}.$$

Solution:

After some guessing and checking, I found $\mathbf{u} = \mathbf{b} - \mathbf{a}$ to work. Here is the proof:

$$H_{\mathbf{b}-\mathbf{a}}\mathbf{a} = \left(\mathbb{I} - \frac{2}{||\mathbf{a}-\mathbf{b}||_2^2}(\mathbf{b}-\mathbf{a})(\mathbf{b}-\mathbf{a})^T\right) \qquad \text{(Given.)}$$

$$= \mathbf{a} - \frac{2}{||\mathbf{b}-\mathbf{a}||_2^2}(\mathbf{b}\mathbf{b}^T\mathbf{a} - \mathbf{b}\mathbf{a}^T\mathbf{a} - \mathbf{a}\mathbf{b}^T\mathbf{a} + \mathbf{a}\mathbf{a}^T\mathbf{a}) \qquad \text{(Factoring out.)}$$

$$= \mathbf{a} - \frac{2}{||\mathbf{b}-\mathbf{a}||_2^2}(\mathbf{b}\langle\mathbf{b}|\mathbf{a}\rangle - \mathbf{b}||\mathbf{a}||_2^2 - \mathbf{a}\langle\mathbf{b}|\mathbf{a}\rangle + \mathbf{a}||\mathbf{a}||_2^2) \qquad \text{(Inner product.)}$$

$$= \mathbf{a} - \frac{2}{||\mathbf{b}-\mathbf{a}||_2^2}(||\mathbf{a}||_2^2(\mathbf{a}-\mathbf{b}) + \langle\mathbf{b}|\mathbf{a}\rangle(\mathbf{b}-\mathbf{a})) \qquad \text{(Grouping.)}$$

$$= \mathbf{a} - \frac{2(\mathbf{a}-\mathbf{b})}{||\mathbf{b}-\mathbf{a}||_2^2}(||\mathbf{a}||_2^2 - ||\mathbf{a}||_2^2\cos(\theta_{ab})) \qquad \text{(}\langle\mathbf{a}|\mathbf{b}\rangle = ||\mathbf{a}||\,||\mathbf{b}||\cos(\theta_{ab})\text{.)}$$

$$= \mathbf{a} - \frac{2||\mathbf{a}||_2^2(\mathbf{a}-\mathbf{b})}{||\mathbf{b}-\mathbf{a}||_2^2}(1 - \cos(\theta_{ab})) \qquad \text{(Grouping.)}$$

$$= \mathbf{a} - \frac{2||\mathbf{a}||_2^2(1-\cos(\theta_{ab}))(\mathbf{a}-\mathbf{b})}{||\mathbf{b}||_2^2 + ||\mathbf{a}||_2^2 - 2\langle\mathbf{a}|\mathbf{b}\rangle} \qquad \text{(2-norm definition.)}$$

$$= \mathbf{a} - \frac{2||\mathbf{a}||_2^2(1-\cos(\theta_{ab}))(\mathbf{a}-\mathbf{b})}{||\mathbf{b}||_2^2 + ||\mathbf{a}||_2^2 - 2||\mathbf{a}||_2^2\cos(\theta_{ab})} \qquad \text{(}\langle\mathbf{a}|\mathbf{b}\rangle = ||\mathbf{a}||\,||\mathbf{b}||\cos(\theta_{ab})\text{.)}$$

$$= \mathbf{a} - \frac{2||\mathbf{a}||_2^2(1-\cos(\theta_{ab}))(\mathbf{a}-\mathbf{b})}{2||\mathbf{a}||_2^2 - 2||\mathbf{a}||_2^2\cos(\theta_{ab})} \qquad \text{(}\langle\mathbf{a}|\mathbf{b}\rangle = ||\mathbf{a}||_2 = ||\mathbf{b}||_2\text{.)}$$

$$= \mathbf{a} - \frac{2||\mathbf{a}||_2^2(1-\cos(\theta_{ab}))(\mathbf{a}-\mathbf{b})}{2||\mathbf{a}||_2^2(1-\cos(\theta_{ab}))} \qquad \text{(Grouping.)}$$

$$= \mathbf{a} - \frac{(1-\cos(\theta_{ab}))(\mathbf{a}-\mathbf{b})}{1-\cos(\theta_{ab})} \qquad \text{(Simplifying.)}$$

$$= \mathbf{a} - (\mathbf{a}-\mathbf{b}) \qquad \text{(Simplifying.)}$$

$$= \mathbf{b} \qquad \text{(Simplifying.)}$$

Problem 5, part e

Show that $\mathbf{u}$ is an eigenvector of $H_{\mathbf{u}}$. What is the corresponding eigenvalue?

Solution:

We will go straight into calculations:

$$
\begin{aligned}
H_{\mathbf{u}}\mathbf{u} &= \left(\mathbb{I} - \frac{2}{\|\mathbf{u}\|_2^2}\mathbf{u}\mathbf{u}^T\right)\mathbf{u} && \text{(Given.)}\\
&= \mathbf{u} - \frac{2}{\|\mathbf{u}\|_2^2}\mathbf{u}\mathbf{u}^T\mathbf{u} && \text{(Distribution.)}\\
&= \mathbf{u} - \frac{2}{\|\mathbf{u}\|_2^2}\mathbf{u}\|\mathbf{u}\|_2^2 && \text{(2-norm definition.)}\\
&= \mathbf{u} - 2\mathbf{u} && \text{(Simplifying.)}\\
&= -\mathbf{u} && \text{(Simplifying.)}
\end{aligned}
$$

So $\mathbf{u}$ is an eigenvector of $H_{\mathbf{u}}$ with eigenvalue -1.

Problem 5, part f

Show that every $\mathbf{v} \in \text{span}\{\mathbf{u}\}^{\perp}$ is an eigenvector of $H_{\mathbf{u}}$. What are the corresponding eigenvalues? What is $\dim(\text{span}\{\mathbf{u}\}^{\perp})$?

Solution:

We will go straight into calculations:

$$\begin{aligned}
H_{\mathbf{u}}\mathbf{v} &= \left(\mathbb{I} - \frac{2}{\|\mathbf{u}\|_2^2}\mathbf{u}\mathbf{u}^T\right)\mathbf{v} && \text{(Given.)} \\
&= \mathbf{v} - \frac{2}{\|\mathbf{u}\|_2^2}\mathbf{u}\mathbf{u}^T\mathbf{v} && \text{(Distribution.)} \\
&= \mathbf{v} - \frac{2}{\|\mathbf{u}\|_2^2}\mathbf{u}(0) && (\mathbf{v} \in \text{span}(\{\mathbf{u}\}^{\perp}).) \\
&= \mathbf{v} && \text{(Simplifying.)}
\end{aligned}$$

So $\mathbf{v}$ is an eigenvector of $H_{\mathbf{u}}$ with eigenvalue 1 for all $\mathbf{v} \in \text{span}\{\mathbf{u}\}^{\perp}$. Note since $\mathbb{R}^n = \text{span}\{\mathbf{u}\}^{\perp} \oplus \text{span}\{\mathbf{u}\}$, then $\dim(\mathbb{R}^n) = \dim(\text{span}\{\mathbf{u}\}^{\perp}) + \text{span}\{\mathbf{u}\}$. Then $\dim(\text{span}\{\mathbf{u}\}^{\perp}) = n - 1$.

Problem 5, part g

Find the eigenvalue decomposition of $H_{\mathbf{u}}$, i.e., find an orthogonal matrix Q and a diagonal matrix Λ such that

$$H_{\mathbf{u}} = Q\Lambda Q^T$$

Solution:

Note that we found all eigenvalues and eigenvectors in the previous two parts. Thus Q can be formed by the found eigenvectors of $H_{\mathbf{u}}$. Therefore, I claim that Q is of the form

$$Q = \begin{bmatrix} | & | & & | \\ \mathbf{u} & \mathbf{v}_2 & \cdots & \mathbf{v}_n \\ | & | & & | \end{bmatrix}$$

We just need to show that Q is orthogonal, that is, Q is composed of orthogonal columns. Note since we showed that every vector in $\text{span}\{\mathbf{u}\}^{\perp}$ is an eigenvector of $H_{\mathbf{u}}$, we are free to chose any orthogonal basis of $\mathbb{R}^{n-1}$ (the last dimension is reserved for $\mathbf{u}$). Without loss of generality, choose $\mathbf{e}_2, ..., \mathbf{e}_n$, the set of canonical unit vectors of $\mathbb{R}^{n-1}$ to be $\mathbf{v}_2, ..., \mathbf{v}_n$. Note these are all orthogonal to each other, and since they are chosen from $\text{span}\{\mathbf{u}\}^{\perp}$, they are all orthogonal to $\mathbf{u}$. Thus, we can choose

$$Q = \begin{bmatrix} | & | & & | \\ \mathbf{u} & \mathbf{e}_2 & \cdots & \mathbf{e}_n \\ | & | & & | \end{bmatrix}$$

which will be orthogonal. Finally, we choose Λ to be the diagonal matrix whose entries coincide with their corresponding eigenvector in Q, so $\Lambda = \text{diag}(-1, 1, ..., 1) \in \mathbb{R}^{n \times n}$.

Problem 6

In this exercise, we will implement and compare Gram–Schmidt and Householder QR. Your implementation should be tailored to the program you are using for efficiency (e.g. vectorize your code in Matlab/Octave/Scilab). Assume in the following that the input is a matrix $A \in \mathbb{R}^{m \times n}$ with $\text{rank}(A) = n \leq m$ and we want to find its full QR decomposition, $A = QR$, where $Q \in O(m)$ and $R \in \mathbb{R}^{m \times n}$ is upper triangular.

Problem 6, part a

Implement the (classical) Gram-Schmidt algorithm to obtain Q and R.

Solution:

Here is my implementation of Gram-Schmidt in python. I tried to vectorize my code as much as possible. Note that even though A is not "passed by reference" in python, I avoided doing so much in my code.

```python
def my_qr(A: np.ndarray):
       """
       Returns the QR factorization of a nonsingular array
       via (classical) Gram-Schmidt.

       Parameters
       ----------
       A: np.ndarray
          The matrix with which we find the QR factorization.
          Must be nonsingular, sized n x n

       Returns
       -------
       Q: np.ndarray
          A set of orthonormal column vectors of size n x n

       R: np.ndarray
          An upper triangular matrix
       """
       m, n = A.shape
       Q = np.zeros((m, m))
       R = np.zeros((m, n))

       for i in range(0, m): #Row Iter
          prev = 0 # used to catch r_{jk}q_j in sum
          for j in range(0, i+1): # maintain upper triangularity
             if i != j:
                R[j, i] = np.dot(Q[:, j], A[:, i])
                prev += R[j, i] * Q[:, j]
             else: #Diagonal term, take prev
                R[i, i] = np.linalg.norm(A[:, i] - prev, ord = 2)
                assert R[i, i] != 0, "Diagonal is zero, function cannot continue"
                Q[:, j] = (1/R[i, i]) * (A[:, i] - prev)
       return (Q, R)
```

Problem 6, part b

Implement the Householder QR algorithm to obtain Q and R. You should

1. Store Q implicitly, taking advantage of the fact that it can be uniquely specified by a sequence of vectors of decreasing dimensions;

2. Choose α in your Householder matrices to have the opposite sign of x_1 to avoid cancellation in v_1 (cf. notations in lecture notes.)

Solution:

Here is my code. I believe everything here is vectorized, so it should be suitably fast. I made a simple helper function to calculate the Hausholder matrix, since it seems we'll need to compute it a few times in the next parts.

```python
def haus(a, j, m):
    """
    Calculates the HausHolder Iteration from given vector, index,
    and size of original matrix
    """
    v = a / (a[0] + np.copysign(np.linalg.norm(a), a[0]))
    v[0] = 1

    H = np.identity(m)
    H[j:, j:] -= (2 / np.linalg.norm(v, ord=2)**2) * np.outer(v, v)
    return v, H

def qr_decomposition_haus(A: np.ndarray) -> np.array:
    """
    Calcualtes the QR decomposition of the given matrix and
    returns the HausHolder vectors and R inside the given matrix.
    """
    m,n = A.shape

    R = A.copy()
    Q = np.identity(m)
    A = np.vstack((A, np.zeros(A.shape[1])))

    for j in range(n):
        # Apply Householder transformation.
        v, H = haus(R[j:, j, np.newaxis], j, m)
        R = np.matmul(H, R)
        Q = np.matmul(H, Q)

        A[j+1:, j] = v.T

    #Replace the Upper triangle of A with R
    indices = np.triu_indices_from(A, k=0)
    A[indices] = R[indices]

    return A
```

Problem 6, part c

Implement an algorithm for forming the product $Q\mathbf{x}$ and another for forming the product $Q^T\mathbf{y}$ when Q is stored implicitly as in (b).

Solution:

Here is my code. I tested it to make sure the returned matrix is indeed orthonomal. All you need to do to get Q is just call **calculate_Q**.

```
def calculate_Hausholder(v: np.array):
    """
    Helper function to calculate the Hausholder matrix from v.

    Parameters
    ----------
    v : np.array
        The given vector. Needs to be normalized prior to passing.

    Returns
    -------
    H : np.ndarray
        The Hausholder matrix of v.
    """
    H = np.identity(max(v.shape)) - 2*np.outer(v, v)
    return H

def recover_Qis(A: np.ndarray):
    """
    Calculates all Q_is from a Hausholder applied matrix A.

    Parameters
    ----------

    A : np.ndarray
        The matrix on which Hausholder QR has been performed

    Returns
    -------
    Qis : list(np.ndarray)
        A list containing all Q's
    """
    Qis = []
    m, n = A.shape

    for i in range(n):
        v = A[i+1:, i].copy()
        Q = np.identity(m-1)
        Q[i:, i:] = calculate_Hausholder(v)
        Qis.append(Q)

    return Qis

def calculate_Q(A: np.ndarray):
    """
    Returns the orthonormal matrix Q from the matrix A after Hausholder QR
    is applied.

    Parameters
    ----------

    A : np.ndarray
        The matrix on which Hausholder QR has been performed

    Returns
    -------

    Q: np.ndarray
        The orthonormal matrix involved in QR
    """

    Qis = recover_Qis(A.copy())
    Q = np.identity(Qis[0].shape[0])
    for Qi in Qis:
        Q = np.matmul(Q, Qi.T)

    return Q
```

Problem 6, part d

For increasing values of n, generate an upper triangular matrix $R \in \mathbb{R}^{n\times n}$ and a $B \in \mathbb{R}^{n\times n}$, both with random standard normal entries. Use your program's built-in function for QR factorization to obtain a random $Q \in O(n)$ from the QR factorization of B. Now form $A = QR$ and apply your algorithms in (a) and (b) to find the QR factors of A - let these be your $\hat{Q}$ and $\hat{R}$. Tabulate (using graphs with appropriate scales) the relative errors

$$\frac{\|R - \hat{R}\|_F}{\|R\|_F}, \quad \frac{\|Q - \hat{Q}\|_F}{\|Q\|_F}, \quad \frac{\|A - \hat{Q}\hat{R}\|_F}{\|A\|_F}, \quad \frac{\|\mathbb{I} - \hat{Q}^T\hat{Q}\|_F}{\|\mathbb{I}\|_F}$$

for various values of n and for each method. Scale $Q, R, \hat{Q}, \hat{R}$ appropriately so that R and $\hat{R}$ have positive diagonal elements. Note that the denominators $\|Q\|_F = \|\mathbb{I}\|_F = \sqrt{n}$ cannot be omitted since they depend on n.

1. Comment on the relative errors in $\hat{Q}$ and $\hat{R}$ (these are called forward errors) versus the relative error in $\hat{Q}\hat{R}$ and $\hat{Q}^T\hat{Q}$ (these are called backward error; note that the latter measures a loss of orthogonality).
2. Comment on the relative error in $\hat{Q}\hat{R}$ and $\hat{Q}^T\hat{Q}$ computed with Gram–Schmidt versus that computed with Householder QR.

Solution:

I will provide my plots below, followed by my code. Note the size of the matrices I observed went up to 1000, with stepsize 10. It seems as though the Hausholder QR proved far more stable in all areas besides Forward Error QR. I would expect to see the Forward error for Gram Schmidt be more unstable, considering the the two Backward errors for GS are themselves unstable. There seems to be just one error in these plots: my backward Error in Q is, although stable, should be closer to zero. I suspect this to just be an error in my calculation of the HausHolder Q. Upon inspection of my Q, it seems to be both orthonormal and give the correct form of QR, so I suspect this to be a sign issue.

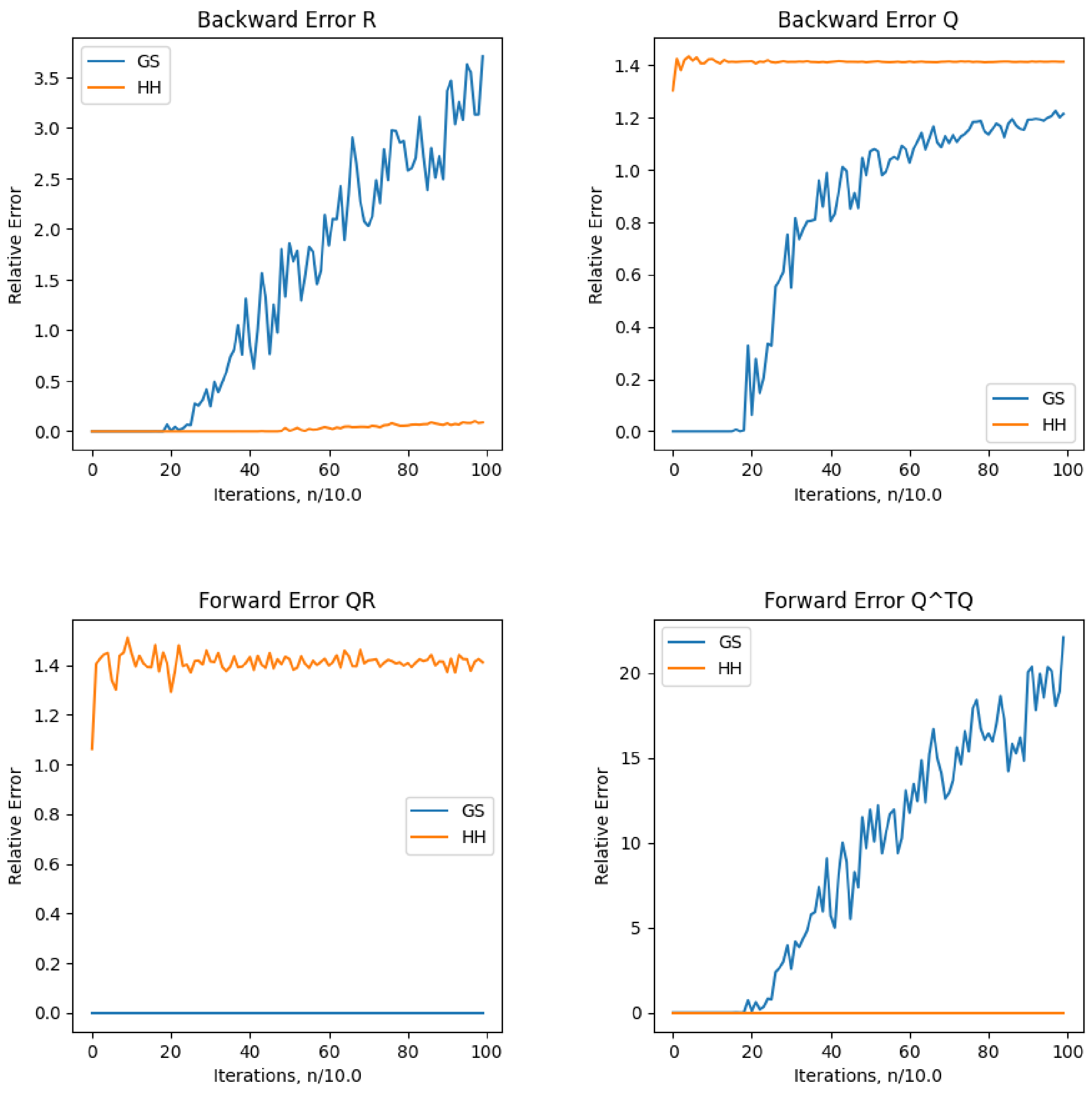

Figure 1: Relative Errors for Gram Schmidt (GS) and Hausholder (HH) QR.

```python
import numpy as np
from IPython.display import display, clear_output
import matplotlib.pyplot as plt
def calculate_Hausholder(v: np.array):
    """
    Helper function to calculate the Hausholder matrix from v.

    Parameters
    ----------
    v : np.array
        The given vector. Needs to be normalized prior to passing.

    Returns
    -------
    H : np.ndarray
        The Hausholder matrix of v.
    """

    H = np.identity(max(v.shape)) - 2*np.outer(v, v)
    return H


def my_qr(A: np.ndarray):
        """
        Returns the QR factorization of a nonsingular array
        via (classical) Gram-Schmidt.

        Parameters
        ----------
        A: np.ndarray
            The matrix with which we find the QR factorization.
            Must be nonsingular, sized n x n

        Returns
        -------
        Q: np.ndarray
            A set of orthonormal column vectors of size n x n

        R: np.ndarray
            An upper triangular matrix
        """
        m, n = A.shape
        Q = np.zeros((m, m))
        R = np.zeros((m, n))

        for i in range(0, m): #Row Iter
            prev = 0 # used to catch r_{jk}q_j in sum
            for j in range(0, i+1): # maintain upper triangularity
                if i != j:
                    R[j, i] = np.dot(Q[:, j], A[:, i])
                    prev += R[j, i] * Q[:, j]
                else: #Diagonal term, take prev
                    R[i, i] = np.linalg.norm(A[:, i] - prev, ord = 2)
                    assert R[i, i] != 0, "Diagonal is zero, function cannot continue"
                    Q[:, j] = (1/R[i, i]) * (A[:, i] - prev)

        return Q, R

def hausholder_qr(A: np.ndarray):
    """
        Returns the QR factorization obtained through
        the Hausholder Reflection method.

        Elements belonging to and above diagonal are R,

        Below the diagonal are the column vectors v_1, ..., v_n-1
        which are used to get Q_1, ..., Q_n-1.

        Last element of v_i is not stored. Can be recovered since
        v is a unit vector.
    """
    m, n = A.shape
    v_mat = []
    for i in range(m-1):

        # Obtaining v_i
        u = A[i:, i].copy()
        u[0] -= np.linalg.norm(u, ord = 2)
        v = u/np.linalg.norm(u, ord = 2)

        v_mat.append(v)

        #Calculating Q only to get next iteration
        Q = np.identity(m)
        Q[i:, i:] = calculate_Hausholder(v)

        A = np.matmul(Q, A)

    #Storing v_i's
    for i in range(m-1):
        A[i+1:, i] = np.array(v_mat[i])[:-1]

    return A

def recover_Qis(A: np.ndarray):
    """
    Calculates all Q_is from a Hausholder applied matrix A.

    Parameters
    ----------

    A : np.ndarray
        The matrix on which Hausholder QR has been performed

    Returns
    -------
    Qis : list(np.ndarray)
        A list containing all Q's
    """
    Qis = []

    m, n = A.shape

    for i in range(m - 1):
        v = np.zeros(m - i)
        v[1:] = A[i+1:, i].copy()
        v[0] = np.sqrt(1 - np.linalg.norm(v, ord = 2)**2)
        Q = np.identity(m)
        Q[i:, i:] = calculate_Hausholder(v)
        Qis.append(Q)

    return Qis

def calculate_Q(A: np.ndarray):
    """
    Returns the orthonormal matrix Q from the matrix A after Hausholder QR
    is applied.

    Parameters
    ----------

    A : np.ndarray
        The matrix on which Hausholder QR has been performed

    Returns
    -------

    Q: np.ndarray
        The orthonormal matrix involved in QR
    """

    Qis = recover_Qis(A.copy())
    Q = np.identity(Qis[0].shape[0])
    for Qi in Qis:
        Q = np.matmul(Q, Qi.T)

    return Q

def Rel_error(A : np.ndarray, B : np.ndarray) -> float:
    """
    Helper function to calculate the relative error of two matrices.
    Relative error is taken with respect to A, and with the
    Frobenius norm
    """

    num = np.linalg.norm(A - B, ord='fro')
    den = np.linalg.norm(A, ord = 'fro')
    return num * (1/den)
```

```
maxsize = 1000
iterations = 100

i = 0

#Result array.
#Size Quantity x iterations x number of functions
res = np.ndarray((4, iterations, 2))
```

```
for n in range(int(maxsize/iterations), maxsize +1 , int(maxsize/iterations)):


    #Temporary array to store quantities.
    #Only holds one set of Quants at a time.
    res_arr = np.zeros(4)

    #Get a random upper triangular matrix. Size will increase
    R = np.triu(np.random.default_rng().normal(10, 2.5, size=(n, n)))

    # Get a random matrix.
    B = np.random.default_rng().normal(10, 2.5, size=(n, n))
    Q = np.linalg.qr(B)[0]

    #Form A = QR
    A = np.matmul(Q, R)

    #Identity matrix
    I = np.identity(n)

    #QR from the Gram schmidt QR
    GS_Q, GS_R = my_qr(A)

    #Forward Errors for GS
    GS_A = np.matmul(GS_Q, GS_R)
    GS_I = np.matmul(GS_Q.T, GS_Q)

    #QR from HausHolder QR
    HH = hausholder_qr(A)
    HH_Q = calculate_Q(HH)
    HH_R = np.triu(HH)

    #Forward errors for HH
    HH_A = np.matmul(HH_Q, HH_R)
    HH_I = np.matmul(HH_Q.T, HH_Q)

    #Calcualting relative error for GS
    res_arr = np.array([Rel_error(R, GS_R),
                        Rel_error(Q, GS_Q),
                        Rel_error(A, GS_A),
                        Rel_error(I, GS_I)
```

```
                    ])

    #Storing GS results for iteration
    res[:, i, 0] = res_arr

    #Calculating relative error for HH
    res_arr = np.array([Rel_error(R, HH_R),
                    Rel_error(Q, HH_Q),
                    Rel_error(A, HH_A),
                    Rel_error(I, HH_I)])

    #Storing HH results for iteration
    res[:, i, 1] = res_arr

    #Incrementing for next iteration
    i+=1

    #Clearing output
    clear_output(wait=True)

    print("Percentage done: {:.2f}%".format(100*n/maxsize))
```

```
Titles = ["Backward Error R", "Backward Error Q", "Forward Error QR", "Forward Error Q^TQ"]
method = ["GS", "HH"]
fig, ax = plt.subplots(2, 2, figsize=(10, 10))
for i in range(2):
    for j in range(2):
        index = i * 2 + j
        ax[i, j].set_title(Titles[index])

        for k in range(2):
            ax[i, j].plot(res[index, :, k], label=method[k])
            ax[i, j].legend()

        ax[i, j].set_xlabel(f'Iterations, n/{maxsize / iterations}')
        ax[i, j].set_ylabel("Relative Error")

plt.tight_layout(pad=5.0)
plt.show()
```

Problem 6, part e

Create a *Vandermonde matrix* and a vector,

$$A = \begin{bmatrix} 1 & \alpha_0 & \alpha_0^2 & \dots & \alpha_0^{n-1} \\ 1 & \alpha_1 & \alpha_1^2 & \dots & \alpha_1^{n-1} \\ 1 & \alpha_2 & \alpha_2^2 & \dots & \alpha_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_{m-1} & \alpha_{m-1}^2 & \dots & \alpha_{m-1}^{n-1} \end{bmatrix} \in \mathbb{R}^{m\times n}, \quad \mathbf{b} = \begin{bmatrix} \exp(\sin 4\alpha_0) \\ \exp(\sin 4\alpha_1) \\ \exp(\sin 4\alpha_2) \\ \vdots \\ \exp(\sin 4\alpha_{m-1}) \end{bmatrix} \in \mathbb{R}^m,$$

where $\alpha_i = i/(m-1), i = 0, ..., m-1$. This arises when we try to do polynomial fitting

$$e^{\sin(4x)} \approx c_0 + c_1 x + x_2 x^2 + \cdots + c_{n-1}x^{n-1}$$

over the interval $[0, 1]$ at discrete points $x = 0, \frac{1}{m-1}, \dots, \frac{m-2}{m-1}, 1$. For $n = 15$ and $m = 100$, solve the least squares problem $\min \|A\mathbf{x} - \mathbf{b}\|_2$ and state your value of c_{14} using each of the following methods.

(i) Applying QR Factorization to A.

(ii) Applying QR Factorization to the augmented matrix $[A, \mathbf{b}] \in \mathbb{R}^{m\times(n+1)}$.

(iii) Solving the normal equations $A^T A\mathbf{x} = A^T\mathbf{b}$.

For (i) and (ii), your code should show how the respective QR factors are used in obtaining a solution of the least squares problem. Your are free to use your program's built-in functions for solving linear systems but for other things, use that you have implemented in (a), (b), (c). The true value of c_{14} is 2006.787453080206....Comment on the accuracy of each method and algorithm.

Solution:

In investigation of my code, it seems that there is something happening going test cases to the Vandermonde matrix itself. Before talking about this, I will provide my modified code so you can see that I actually tried to perform this calculation, but was getting wild results. Here I will report the result of my code as "Reported solution to c_14: -11722843.91526459", which is completely wrong. I will provide figures that show the discrepancies between my QR decompositions and those found in numpy (Shown in Figures 2, 3, and 4). I will continue this and finish this problem with a QR decompositions that work, **but are not mine**. I will disclose where I got them in their relative blocks.

```python
import numpy as np
import math
import sys
import matplotlib.pyplot as plt
def calculate_Hausholder(v: np.array):
    """
    Helper function to calculate the Hausholder matrix from v.

    Parameters
    ----------
    v : np.array
        The given vector. Needs to be normalized prior to passing.

    Returns
    -------
    H : np.ndarray
        The Hausholder matrix of v.
    """
    H = np.identity(max(v.shape)) - 2*np.outer(v, v)
    return H

def recover_Qis(A: np.ndarray):
    """
    Calculates all Q_is from a Hausholder applied matrix A.

    Parameters
    ----------

    A : np.ndarray
        The matrix on which Hausholder QR has been performed

    Returns
    -------
    Qis : list(np.ndarray)
        A list containing all Q's
    """
    Qis = []

    m, n = A.shape

    for i in range(n):
        v = A[i+1:, i].copy()
        Q = np.identity(m-1)
        Q[i:, i:] = calculate_Hausholder(v)
        Qis.append(Q)

    return Qis

def calculate_Q(A: np.ndarray):
    """
    Returns the orthonormal matrix Q from the matrix A after Hausholder QR
    is applied.

    Parameters
    ----------

    A : np.ndarray
        The matrix on which Hausholder QR has been performed

    Returns
    -------

    Q: np.ndarray
        The orthonormal matrix involved in QR
    """

    Qis = recover_Qis(A.copy())
    Q = np.identity(Qis[0].shape[0])
    for Qi in Qis:
        Q = np.matmul(Q, Qi.T)

    return Q


def haus(a, j, m):
    """
    Calculates the HausHolder Iteration from given vector, index,
    and size of original matrix
    """
    v = a / (a[0] + np.copysign(np.linalg.norm(a), a[0]))
    v[0] = 1

    H = np.identity(m)
    H[j:, j:] -= (2 / np.linalg.norm(v, ord=2)**2) * np.outer(v, v)
    return v, H

def qr_decomposition_haus(A: np.ndarray) -> np.array:
    """
    Calcualtes the QR decomposition of the given matrix and
    returns the HausHolder vectors and R inside the given matrix.
    """
    m,n = A.shape

    R = A.copy()
    Q = np.identity(m)
    A = np.vstack((A, np.zeros(A.shape[1])))

    for j in range(n):
        # Apply Householder transformation.
        v, H = haus(R[j:, j, np.newaxis], j, m)

        R = np.matmul(H, R)
        Q = np.matmul(H, Q)

        A[j+1:, j] = v.T

    #Replace the Upper triangle of A with R
    indices = np.triu_indices_from(A, k=0)
    A[indices] = R[indices]

    return A

def my_qr(A: np.ndarray):
        """
        Returns the QR factorization of a nonsingular array
        via (classical) Gram-Schmidt.

        Parameters
        ----------
        A: np.ndarray
            The matrix with which we find the QR factorization.
            Must be nonsingular, sized n x n

        Returns
        -------
        Q: np.ndarray
            A set of orthonormal column vectors of size n x n

        R: np.ndarray
            An upper triangular matrix
        """
        m, n = A.shape
        Q = np.zeros((m, m))
        R = np.zeros((m, n))

        for i in range(0, n): #Row Iter
            prev = 0 # used to catch r_{jk}q_j in sum
            for j in range(0, i+1): # maintain upper triangularity
                if i != j:
                    R[j, i] = np.dot(Q[:, j], A[:, i])
                    prev += R[j, i] * Q[:, j]
                else: #Diagonal term, take prev
                    R[i, i] = np.linalg.norm(A[:, i] - prev, ord = 2)
                    assert R[i, i] != 0, "Diagonal is zero, function cannot continue"
                    Q[:, j] = (1/R[i, i]) * (A[:, i] - prev)

        return (Q, R)
def get_Van_b():
    """
    Gets the Vandermonde matrix and b we are considering.

    """
    n = 15
    m = 100

    alphas = np.array([i/(m - 1) for i in range(m)])

    Van = np.ndarray((m, n))
    for i in range(m):
        for j in range(n):
            Van[i, j] = alphas[i]**j
    b = np.zeros(m)
    for i in range(m):
        b[i] = math.exp(math.sin(4 * alphas[i]))

    return Van, b
Van, b = get_Van_b()
```

```python
def back_substitution(A: np.ndarray, b: np.ndarray) -> np.ndarray:
    n = b.size
    x = np.zeros_like(b)

    if A[n-1, n-1] == 0:
        raise ValueError


    x[n-1] = b[n-1]/A[n-1, n-1]
    C = np.zeros((n,n))
    for i in range(n-2, -1, -1):
        bb = 0
        for j in range (i+1, n):
            bb += A[i, j]*x[j]

        C[i, i] = b[i] - bb
        x[i] = C[i, i]/A[i, i]

    return x
n = 15
m = 100

HH = qr_decomposition_haus(Van)
#HH = np.linalg.qr(Van)

R = np.triu(HH)

#Take nonzero rows of R. Since Van is full rank, will be n rows
R = np.array([x for x in R if x.any() != 0])
#Check if I did it right
assert R.shape == (n, n), "Oops"

Q = calculate_Q(HH)

#Q = HH[0]
c = np.matmul(Q[:, :n].T, b)
d = np.matmul(Q[:, n:].T, b) #Don't think I need to actually compute this

#Solution to part i
x_soli = back_substitution(R, c)
print("Reported solution to c_14: {}".format(x_soli[-1]))
```

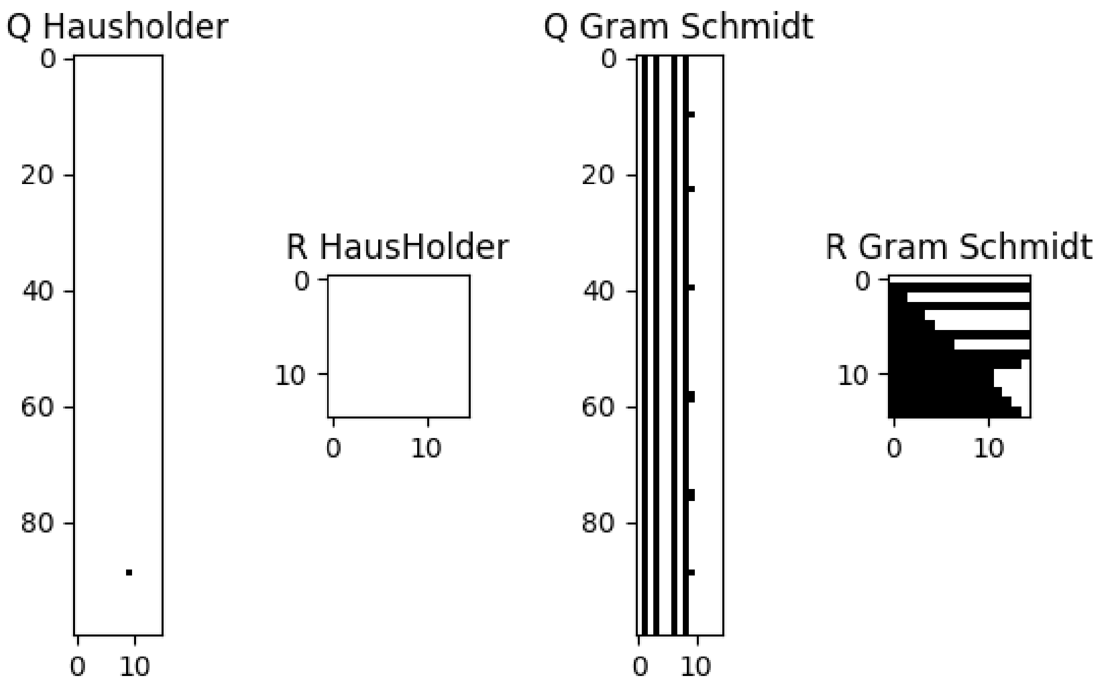

Figure 2: The Absolute difference entry-wise between my HausHolder and Gram Schmidt QR decompositions compared to numpy's QR decomposition. The tolerance that I take each entry to be equal is 0.0001. Here I report an element to match if it is colored black, and white if it is not within floating point tolerance.

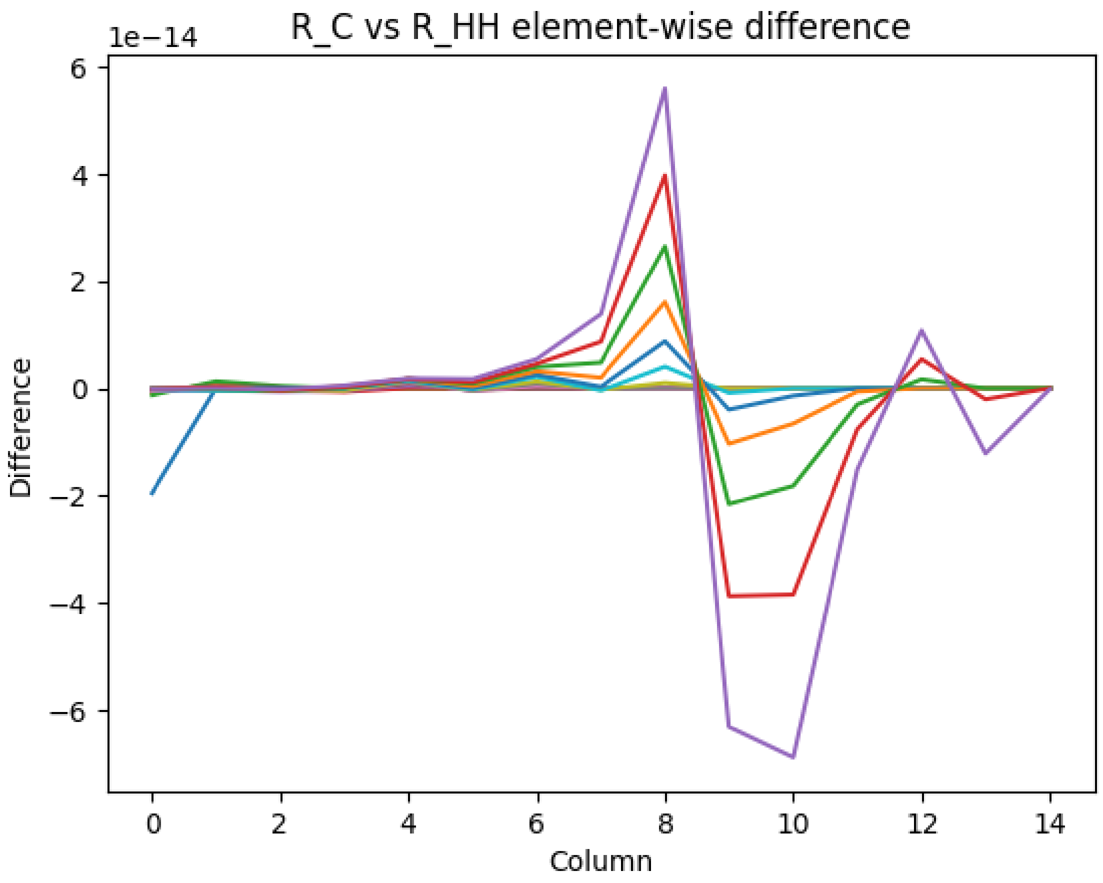

Figure 3: The element-wise difference between my HausHolder QR versus the QR decomposition found in numpy. Note the lines are the rows in R, where they span 15 elements each. It seems that the difference is within floating point precision.

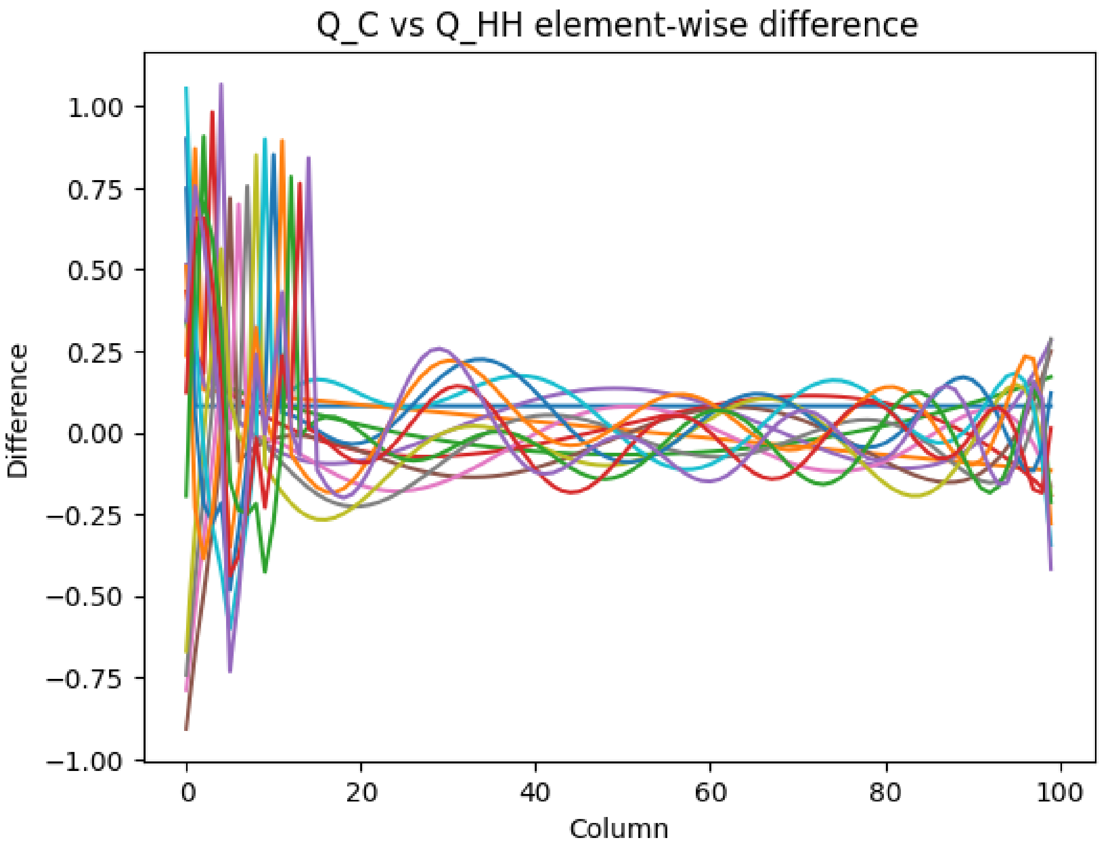

Figure 4: The element-wise difference between my HausHolder QR versus the QR decomposition found in numpy. Note the lines are the rows in Q, where they span 15 elements each. It seems that these results are completely off, but I am not entirely sure why.

```
"""
THIS IS NOT MY CODE. THIS IS TAKEN FROM A STACK-OVERFLOW POST ON HAUSHOLDER QR
   FACTORIZATION.
THE LINK IS
https://stackoverflow.com/questions/53489237/
how-can-you-implement-householder-based-qr-decomposition-in-python
"""
def householder_vectorized(a):
    """Use this version of householder to reproduce the output of np.linalg.qr
    exactly (specifically, to match the sign convention it uses)

    based on https://rosettacode.org/wiki/QR_decomposition#Python
    """
    v = a / (a[0] + np.copysign(np.linalg.norm(a), a[0]))
    v[0] = 1
    tau = 2 / (v.T @ v)

    return v,tau

def qr_decomposition(A: np.ndarray) -> Union[np.ndarray, np.ndarray]:
    m,n = A.shape
    R = A.copy()
    Q = np.identity(m)

    for j in range(0, n):
        # Apply Householder transformation.
        v, tau = householder_vectorized(R[j:, j, np.newaxis])

        H = np.identity(m)
        H[j:, j:] -= tau * (v @ v.T)
        R = H @ R
        Q = H @ Q

    return Q[:n].T, np.triu(R[:n])
```

Continue Problem 6, part e

Here I will continue problem 6, part e with the HausHolder QR decomposition shown in the code block directly above this section. I will use the Gram Schmidt QR that I wrote - I wanted to get accurate results with at least one of them.

1. since the Vandermonde matrix given is full rank (i.e. is full column rank), we can apply Full QR decomposition to get
$$A = Q\begin{bmatrix} R \\ 0 \end{bmatrix}.$$
We can split Q up into the first 15 columns as Q_1 and the rest as Q_2. When we plug this into the least squares problem, we see that
$$\min \|A\mathbf{x} - \mathbf{b}\|_2^2 = \min(\|R\mathbf{x} - \mathbf{c}\|_2^2 + \|\mathbf{d}\|_2^2),$$
where $\mathbf{d} = Q_2^T\mathbf{b}$ and $\mathbf{c} = Q_1^T\mathbf{b}$. Since R is invertible, we get the solution to the least squares problem will be $\mathbf{x} = R^{-1}\mathbf{c}$, which will have the minimum $\|\mathbf{d}\|_2$. with this approach, we report that the HausHolder QR factorization gives a value for $c_{14} \approx 2006.7870665592704$. This is within 3 decimal places of the true value given. This is in contrast to my Gram Schmidt QR factorization, where we report the value to be $c_{14} \approx 1.099680$, which is completely off. I will provide my code below.

```
n = 15
m = 100

#Calcualting QR
Q_HH, R_HH = qr_decomposition(Van)
Q_GS, R_GS = my_qr(Van)

#Take nonzero rows of R. Since Van is full rank, will be n rows
R_HH = np.array([x for x in R_HH if x.any() != 0])

#Check if I did it right
assert R_HH.shape == (n, n), "Oops"

c = np.matmul(Q_HH[:, :n].T, b)

#Solution to part i
x_soli = back_substitution(R_HH, c)
print("HausHolder Approximate solution to c_14: {} ".format(x_soli[-1]))
#Take nonzero rows of R. Since Van is full rank, will be n rows
R_GS = np.array([x for x in R_GS if x.any() != 0])

#Check if I did it right
```

```
assert R_GS.shape == (n, n), "Oops"

c = np.matmul(Q_GS[:, :n].T, b)
x_soli = back_substitution(R_GS, c)
print("Gram Schmidt Approximate solution to c_14: {}".format(x_soli[-1]))
```

2. When we apply QR factorization to the augmented matrix, we will get back a 100-by-16 size Q and 16-by-16 size R. What this will do, since the first 15 columns of R give us back A, is compute the residuals of **b** has with A. That is, the minimum value any **x** can attain will be given as the norm of the last column. We can then take R_{aug} to be the first 15 columns of R, and **c** to be the last column and first 15 rows R, since the last entry will be our minimum value. With this, we report the HausHolder QR Factorization to give a c_{14} value of 2006.7870664971297, which is correct up to three decimal places of accuracy. The Gram Scmidt approximation gives $c_14 \approx 1.0996800275618466$, which is completely incorrect. I will provide my code below.

```
n = 15
m = 100
aug_A = np.append(Van,b.reshape((b.shape[0],1)) ,1)

#Calcualting QR
Q_HH, R_HH = qr_decomposition(aug_A)
Q_GS, R_GS = my_qr(aug_A)

#Retrieving c and R_aug
c = R_HH[:-1, -1]
R_aug = R_HH[:-1, :-1]

x_solii = back_substitution(R_aug, c)
print("HausHolder Approximate solution to c_14: {} ".format(x_solii[-1]))

#Redo for GS
c = R_GS[:n, -1]
R_aug = R_GS[:n, :-1]

x_solii = back_substitution(R_aug, c)
print("Gram Schmidt Approximate solution to c_14: {}".format(x_solii[-1]))
```

3. The solution to the least squares normal equation is just $\mathbf{x} = A^{\dagger}A^{T}\mathbf{b}$, where $A^{\dagger} = R^{-1}Q_1^T$, given in Problem 3. I understand that I am computing the inverse of a matrix which means that I instantly fail the class. However, I will never actually do this in a non-academic setting, and I understand this is a horrible way to get the solution, namely since the normal equation's condition number squares with the condition number of A. I report the result of HausHolder QR gives a value of $c_{14} \approx 5004.537912368087$, which is understandably incorrect, since the condition number of A^TA is of order 10^{17}. The Gram schmidt QR reports a value of $c_{14} \approx -73024.29641227845$, which is again wildly incorrect. Here is my code:

```python
A_n = Van.T @ Van
b_n = Van.T @b

Q_nHH, R_nHH = qr_decomposition(A_n)
Q_nGS, R_nGS = my_qr(A_n)

A_p = np.linalg.inv(R_nHH) @ Q_nHH[:n, :].T

x_soliii = np.linalg.solve(A_p, b_n)
print("HausHolder Approximate solution to c_14: {} ".format(x_soliii[-1]))
A_p = np.linalg.inv(R_nGS) @ Q_nGS[:n, :].T

x_soliii = np.linalg.solve(A_p, b_n)
print("Gram Schmidt Approximate solution to c_14: {}".format(x_soliii[-1]))
```