6. QR factorization

- triangular matrices
- QR factorization
- Gram–Schmidt algorithm
- Householder algorithm

Triangular matrix

a square matrix A is **lower triangular** if $A_{ij} = 0$ for $j > i$

$$A = \begin{bmatrix} A_{11} & 0 & \cdots & 0 & 0 \\ A_{21} & A_{22} & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & 0 & 0 \\ A_{n-1,1} & A_{n-1,2} & \cdots & A_{n-1,n-1} & 0 \\ A_{n1} & A_{n2} & \cdots & A_{n,n-1} & A_{nn} \end{bmatrix}$$

A is **upper triangular** if $A_{ij} = 0$ for $j < i$ (the transpose A^T is lower triangular)

a triangular matrix is **unit** upper/lower triangular if $A_{ii} = 1$ for all i

Forward substitution

solve $Ax = b$ when A is lower triangular with nonzero diagonal elements

Algorithm

$$
\begin{aligned}
x_1 &= b_1/A_{11} \\
x_2 &= (b_2 - A_{21}x_1)/A_{22} \\
x_3 &= (b_3 - A_{31}x_1 - A_{32}x_2)/A_{33} \\
&\vdots \\
x_n &= (b_n - A_{n1}x_1 - A_{n2}x_2 - \cdots - A_{n,n-1}x_{n-1})/A_{nn}
\end{aligned}
$$

Complexity: $1 + 3 + 5 + \cdots + (2n-1) = n^2$ flops

Back substitution

solve $Ax = b$ when A is upper triangular with nonzero diagonal elements

Algorithm

$$\begin{aligned}
x_n &= b_n / A_{nn} \\
x_{n-1} &= (b_{n-1} - A_{n-1,n} x_n) / A_{n-1,n-1} \\
x_{n-2} &= (b_{n-2} - A_{n-2,n-1} x_{n-1} - A_{n-2,n} x_n) / A_{n-2,n-2} \\
&\vdots \\
x_1 &= (b_1 - A_{12} x_2 - A_{13} x_3 - \cdots - A_{1n} x_n) / A_{11}
\end{aligned}$$

Complexity: n^2 flops

Inverse of a triangular matrix

a triangular matrix A with nonzero diagonal elements is nonsingular:

$$Ax = 0 \quad \Longrightarrow \quad x = 0$$

this follows from forward or back substitution applied to the equation $Ax = 0$

- inverse of A can be computed by solving $AX = I$ column by column

$$A \left[\begin{array}{cccc} x_1 & x_2 & \cdots & x_n \end{array} \right] = \left[\begin{array}{cccc} e_1 & e_2 & \cdots & e_n \end{array} \right] \qquad (x_i \text{ is column } i \text{ of } X)$$

- inverse of lower triangular matrix is lower triangular
- inverse of upper triangular matrix is upper triangular
- complexity of computing inverse of $n \times n$ triangular matrix is

$$n^2 + (n-1)^2 + \cdots + 1 \approx \frac{1}{3}n^3 \text{ flops}$$

Outline

- triangular matrices
- **QR factorization**
- Gram–Schmidt algorithm
- Householder algorithm

QR factorization

if $A \in \mathbf{R}^{m\times n}$ has linearly independent columns then it can be factored as

$$A = \begin{bmatrix} q_1 & q_2 & \cdots & q_n \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & \cdots & R_{1n} \\ 0 & R_{22} & \cdots & R_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & R_{nn} \end{bmatrix}$$

- vectors $q_1, \ldots, q_n$ are orthonormal m-vectors:

$$\|q_i\| = 1, \qquad q_i^T q_j = 0 \quad \text{if } i \neq j$$

- diagonal elements R_{ii} are nonzero
- if $R_{ii} < 0$, we can switch the signs of $R_{ii}, \ldots, R_{in}$, and the vector q_i
- most definitions require $R_{ii} > 0$; this makes Q and R unique

QR factorization in matrix notation

if $A \in \mathbf{R}^{m \times n}$ has linearly independent columns then it can be factored as

$$A = QR$$

Q-factor

- Q is $m \times n$ with orthonormal columns ($Q^T Q = I$)
- if A is square ($m = n$), then Q is orthogonal ($Q^T Q = QQ^T = I$)

R-factor

- R is $n \times n$, upper triangular, with nonzero diagonal elements
- R is nonsingular (diagonal elements are nonzero)

Example

$$
\begin{bmatrix} -1 & -1 & 1 \\ 1 & 3 & 3 \\ -1 & -1 & 5 \\ 1 & 3 & 7 \end{bmatrix}
= \begin{bmatrix} -1/2 & 1/2 & -1/2 \\ 1/2 & 1/2 & -1/2 \\ -1/2 & 1/2 & 1/2 \\ 1/2 & 1/2 & 1/2 \end{bmatrix}
\begin{bmatrix} 2 & 4 & 2 \\ 0 & 2 & 8 \\ 0 & 0 & 4 \end{bmatrix}
$$

$$
= \begin{bmatrix} q_1 & q_2 & q_3 \end{bmatrix}
\begin{bmatrix} R_{11} & R_{12} & R_{13} \\ 0 & R_{22} & R_{23} \\ 0 & 0 & R_{33} \end{bmatrix}
$$

$$
= QR
$$

Applications

in the following lectures, we will use the QR factorization to solve

- linear equations
- least squares problems
- constrained least squares problems

here, we show that it gives useful simple formulas for

- the pseudo-inverse of a matrix with linearly independent columns
- the inverse of a nonsingular matrix
- projection on the range of a matrix with linearly independent columns

QR factorization and (pseudo-)inverse

pseudo-inverse of a matrix A with linearly independent columns (page 4.23)

$$A^\dagger = (A^TA)^{-1}A^T$$

- pseudo-inverse in terms of QR factors of A:

$$\begin{aligned}
A^\dagger &= ((QR)^T(QR))^{-1}(QR)^T \\
&= (R^TQ^TQR)^{-1}R^TQ^T \\
&= (R^TR)^{-1}R^TQ^T \qquad\qquad (Q^TQ = I) \\
&= R^{-1}R^{-T}R^TQ^T \qquad\qquad (R \text{ is nonsingular}) \\
&= R^{-1}Q^T
\end{aligned}$$

- for square nonsingular A this is the inverse:

$$A^{-1} = (QR)^{-1} = R^{-1}Q^T$$

Range

recall definition of range of a matrix $A \in \mathbf{R}^{m\times n}$ (page 5.16):

$$\mathrm{range}(A) = \{Ax \mid x \in \mathbf{R}^n\}$$

suppose A has linearly independent columns with QR factors Q, R

- Q has the same range as A:

$$\begin{aligned} y \in \mathrm{range}(A) \quad &\iff \quad y = Ax \text{ for some } x \\ &\iff \quad y = QRx \text{ for some } x \\ &\iff \quad y = Qz \text{ for some } z \\ &\iff \quad y \in \mathrm{range}(Q) \end{aligned}$$

- columns of Q are orthonormal and have the same span as columns of A

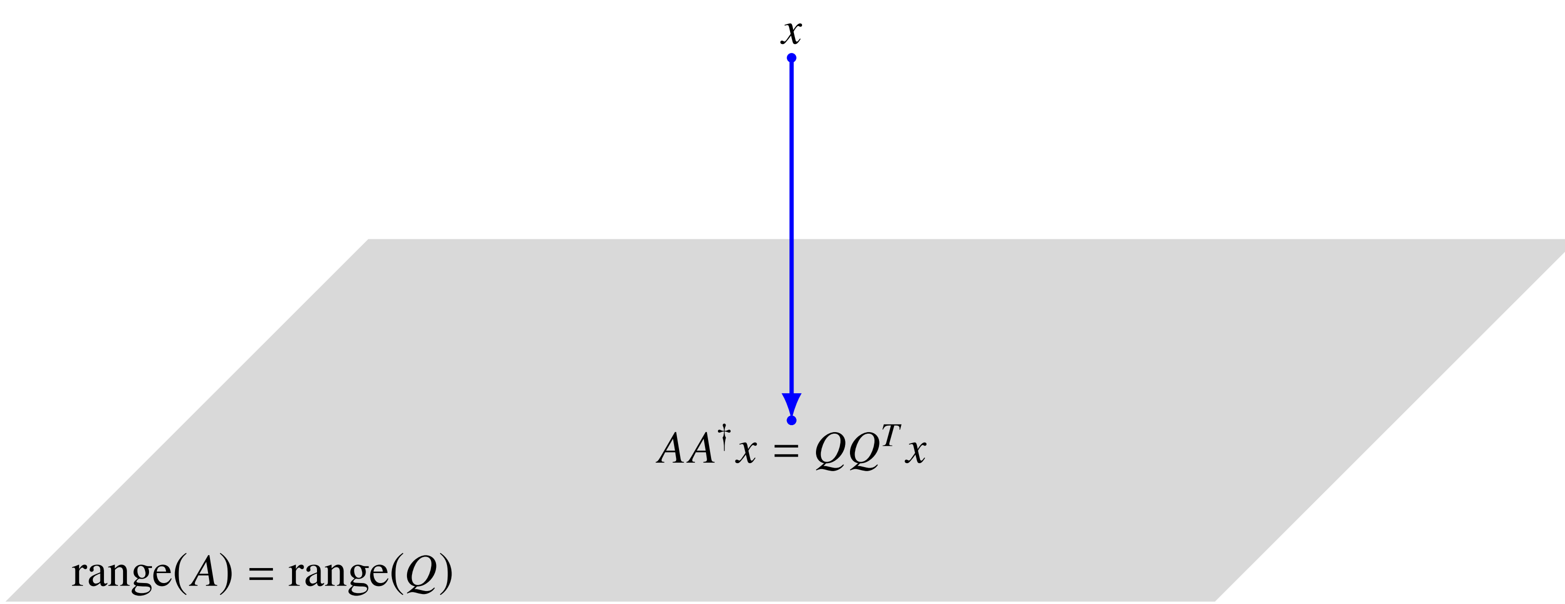
Projection on range

- combining $A = QR$ and $A^\dagger = R^{-1}Q^T$ (from page 6.10) gives

$$AA^\dagger = QRR^{-1}Q^T = QQ^T$$

note the order of the product in $AA^\dagger$ and the difference with $A^\dagger A = I$

- recall (from page 5.17) that $QQ^T x$ is the projection of x on the range of Q



QR factorization of complex matrices

if $A \in \mathbf{C}^{m \times n}$ has linearly independent columns then it can be factored as

$$A = QR$$

- $Q \in \mathbf{C}^{m \times n}$ has orthonormal columns ($Q^H Q = I$)
- $R \in \mathbf{C}^{n \times n}$ is upper triangular with real nonzero diagonal elements
- most definitions choose diagonal elements R_{ii} to be positive
- in the rest of the lecture we assume A is real

Algorithms for QR factorization

Gram–Schmidt algorithm (page 6.15)

- complexity is $2mn^2$ flops
- not recommended in practice (sensitive to rounding errors)

Modified Gram–Schmidt algorithm

- complexity is $2mn^2$ flops
- better numerical properties

Householder algorithm (page 6.25)

- complexity is $2mn^2 - (2/3)n^3$ flops
- represents Q as a product of elementary orthogonal matrices
- the most widely used algorithm (used by the function `qr` in MATLAB and Julia)

in the rest of the course we will take $2mn^2$ for the complexity of QR factorization

Outline

- triangular matrices
- QR factorization
- **Gram–Schmidt algorithm**
- Householder algorithm

Gram–Schmidt algorithm

Gram–Schmidt QR algorithm computes Q and R column by column

- after k steps we have a partial QR factorization

$$\left[\begin{array}{cccc} a_1 & a_2 & \cdots & a_k \end{array}\right] = \left[\begin{array}{cccc} q_1 & q_2 & \cdots & q_k \end{array}\right] \left[\begin{array}{cccc} R_{11} & R_{12} & \cdots & R_{1k} \\ 0 & R_{22} & \cdots & R_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & R_{kk} \end{array}\right]$$

- columns $q_1, \ldots, q_k$ are orthonormal

- diagonal elements $R_{11}, R_{22}, \ldots, R_{kk}$ are positive

- columns $q_1, \ldots, q_k$ have the same span as $a_1, \ldots, a_k$ (see page 6.11)

Computing column k

suppose we have completed the factorization for the first $k-1$ columns

- column k of the equation $A = QR$ reads

$$a_k = R_{1k}q_1 + R_{2k}q_2 + \cdots + R_{k-1,k}q_{k-1} + R_{kk}q_k$$

- regardless of how we choose $R_{1k}, \ldots, R_{k-1,k}$, the vector

$$\tilde{q}_k = a_k - R_{1k}q_1 - R_{2k}q_2 - \cdots - R_{k-1,k}q_{k-1}$$

  will be nonzero: $a_1, a_2, \ldots, a_k$ are linearly independent and therefore

$$a_k \notin \operatorname{span}\{a_1, \ldots, a_{k-1}\} = \operatorname{span}\{q_1, \ldots, q_{k-1}\}$$

- q_k is $\tilde{q}_k$ normalized: choose $R_{kk} = \|\tilde{q}_k\|$ and $q_k = (1/R_{kk})\tilde{q}_k$
- $\tilde{q}_k$ and q_k are orthogonal to $q_1, \ldots, q_{k-1}$ if we choose $R_{1k}, \ldots, R_{k-1,k}$ as

$$R_{1k} = q_1^T a_k, \qquad R_{2k} = q_2^T a_k, \qquad \ldots, \qquad R_{k-1,k} = q_{k-1}^T a_k$$

Gram–Schmidt algorithm

Given: $m \times n$ matrix A with linearly independent columns $a_1, \ldots, a_n$

Algorithm

for $k = 1$ to n

$$
\begin{aligned}
R_{1k} &= q_1^T a_k \\
R_{2k} &= q_2^T a_k \\
&\vdots \\
R_{k-1,k} &= q_{k-1}^T a_k \\
\tilde{q}_k &= a_k - (R_{1k} q_1 + R_{2k} q_2 + \cdots + R_{k-1,k} q_{k-1}) \\
R_{kk} &= \|\tilde{q}_k\| \\
q_k &= \frac{1}{R_{kk}} \tilde{q}_k
\end{aligned}
$$

Example

example on page 6.8:

$$\begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix} = \begin{bmatrix} -1 & -1 & 1 \\ 1 & 3 & 3 \\ -1 & -1 & 5 \\ 1 & 3 & 7 \end{bmatrix}$$

$$= \begin{bmatrix} q_1 & q_2 & q_3 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ 0 & R_{22} & R_{23} \\ 0 & 0 & R_{33} \end{bmatrix}$$

First column of Q and R

$$\tilde{q}_1 = a_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \\ 1 \end{bmatrix}, \qquad R_{11} = \|\tilde{q}_1\| = 2, \qquad q_1 = \frac{1}{R_{11}}\tilde{q}_1 = \begin{bmatrix} -1/2 \\ 1/2 \\ -1/2 \\ 1/2 \end{bmatrix}$$

Example

Second column of Q and R

- compute $R_{12} = q_1^T a_2 = 4$
- compute

$$\tilde{q}_2 = a_2 - R_{12} q_1 = \begin{bmatrix} -1 \\ 3 \\ -1 \\ 3 \end{bmatrix} - 4 \begin{bmatrix} -1/2 \\ 1/2 \\ -1/2 \\ 1/2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

- normalize to get

$$R_{22} = \|\tilde{q}_2\| = 2, \qquad q_2 = \frac{1}{R_{22}} \tilde{q}_2 = \begin{bmatrix} 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \end{bmatrix}$$

Example

Third column of Q and R

- compute $R_{13} = q_1^T a_3 = 2$ and $R_{23} = q_2^T a_3 = 8$
- compute

$$\tilde{q}_3 = a_3 - R_{13}q_1 - R_{23}q_2 = \begin{bmatrix} 1 \\ 3 \\ 5 \\ 7 \end{bmatrix} - 2 \begin{bmatrix} -1/2 \\ 1/2 \\ -1/2 \\ 1/2 \end{bmatrix} - 8 \begin{bmatrix} 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \end{bmatrix} = \begin{bmatrix} -2 \\ -2 \\ 2 \\ 2 \end{bmatrix}$$

- normalize to get

$$R_{33} = \|\tilde{q}_3\| = 4, \qquad q_3 = \frac{1}{R_{33}}\tilde{q}_3 = \begin{bmatrix} -1/2 \\ -1/2 \\ 1/2 \\ 1/2 \end{bmatrix}$$

Example

Final result

$$
\begin{bmatrix} -1 & -1 & 1 \\ 1 & 3 & 3 \\ -1 & -1 & 5 \\ 1 & 3 & 7 \end{bmatrix}
= \begin{bmatrix} q_1 & q_2 & q_3 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ 0 & R_{22} & R_{23} \\ 0 & 0 & R_{33} \end{bmatrix}
$$

$$
= \begin{bmatrix} -1/2 & 1/2 & -1/2 \\ 1/2 & 1/2 & -1/2 \\ -1/2 & 1/2 & 1/2 \\ 1/2 & 1/2 & 1/2 \end{bmatrix} \begin{bmatrix} 2 & 4 & 2 \\ 0 & 2 & 8 \\ 0 & 0 & 4 \end{bmatrix}
$$

Complexity

Complexity of cycle k (of algorithm on page 6.17)

- $k-1$ inner products with a_k: $(k-1)(2m-1)$ flops
- computation of $\tilde{q}_k$: $2(k-1)m$ flops
- computing R_{kk} and q_k: $3m$ flops

total for cycle k: $(4m-1)(k-1)+3m$ flops

Complexity for $m \times n$ factorization:

$$\begin{aligned} \sum_{k=1}^{n}((4m-1)(k-1)+3m) &= (4m-1)\frac{n(n-1)}{2}+3mn \\ &\approx 2mn^2 \text{ flops} \end{aligned}$$

Numerical experiment

- we use the following MATLAB code

```
[m, n] = size(A);
Q = zeros(m,n);
R = zeros(n,n);
for k = 1:n
    R(1:k-1,k) = Q(:,1:k-1)' * A(:,k);
    v = A(:,k) - Q(:,1:k-1) * R(1:k-1,k);
    R(k,k) = norm(v);
    Q(:,k) = v / R(k,k);
end;
```

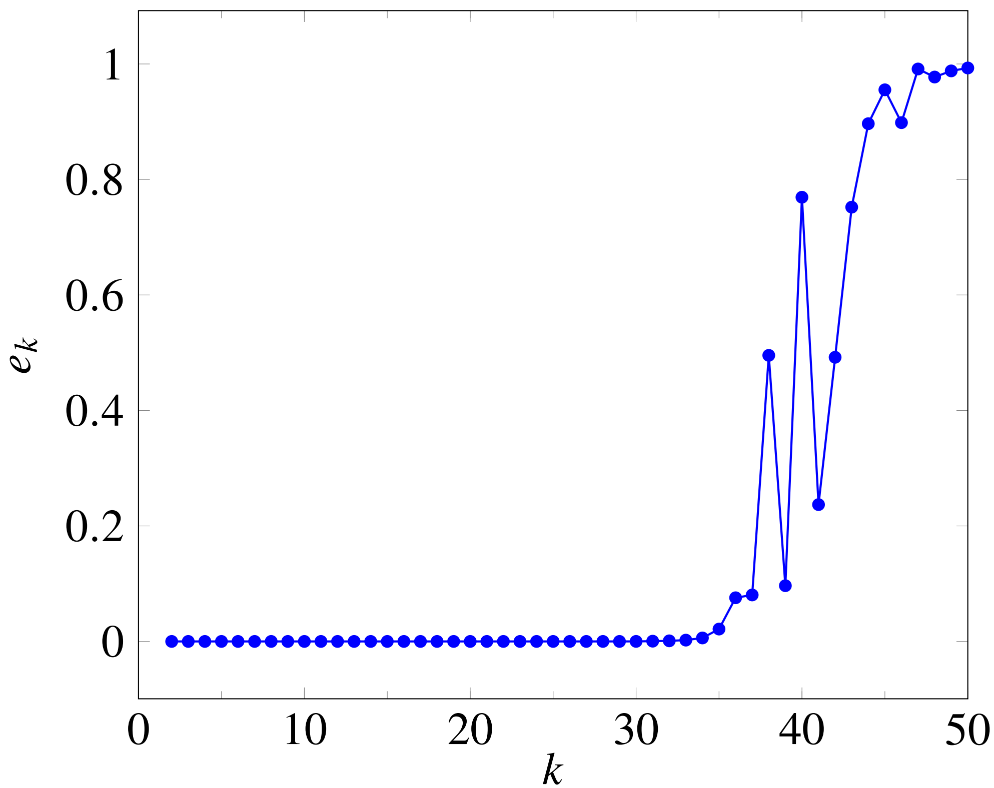
- we apply this to a square matrix A of size $m = n = 50$
- A is constructed as $A = USV$ with U, V orthogonal, S diagonal with

$$S_{ii} = 10^{-10(i-1)/(n-1)}, \quad i = 1, \ldots, n$$

Numerical experiment

plot shows deviation from orthogonality between q_k and previous columns

$$e_k = \max_{1 \leq i < k} |q_i^T q_k|, \quad k = 2, \ldots, n$$



loss of orthogonality is due to rounding error

Outline

- triangular matrices
- QR factorization
- Gram–Schmidt algorithm
- **Householder algorithm**

Householder algorithm

- the most widely used algorithm for QR factorization (`qr` in MATLAB and Julia)
- less sensitive to rounding error than Gram–Schmidt algorithm
- computes a “full” QR factorization

$$A = \begin{bmatrix} Q & \tilde{Q} \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix}, \qquad \begin{bmatrix} Q & \tilde{Q} \end{bmatrix} \text{ orthogonal}$$

- the full Q-factor is constructed as a product of orthogonal matrices

$$\begin{bmatrix} Q & \tilde{Q} \end{bmatrix} = H_1 H_2 \cdots H_n$$

each H_i is an $m \times m$ symmetric, orthogonal “reflector” (page 5.10)

Reflector

$$H = I - 2vv^T \qquad \text{with } \|v\| = 1$$

- Hx is reflection of x through hyperplane $\{z \mid v^T z = 0\}$ (see page 5.10)
- H is symmetric
- H is orthogonal
- matrix-vector product Hx can be computed efficiently as

$$Hx = x - 2(v^T x)v$$

complexity is $4p$ flops if v and x have length p

Reflection to multiple of unit vector

given nonzero p-vector $y = (y_1, y_2, \ldots, y_p)$, define

$$w = \begin{bmatrix} y_1 + \operatorname{sign}(y_1)\|y\| \\ y_2 \\ \vdots \\ y_p \end{bmatrix}, \qquad v = \frac{1}{\|w\|}w$$

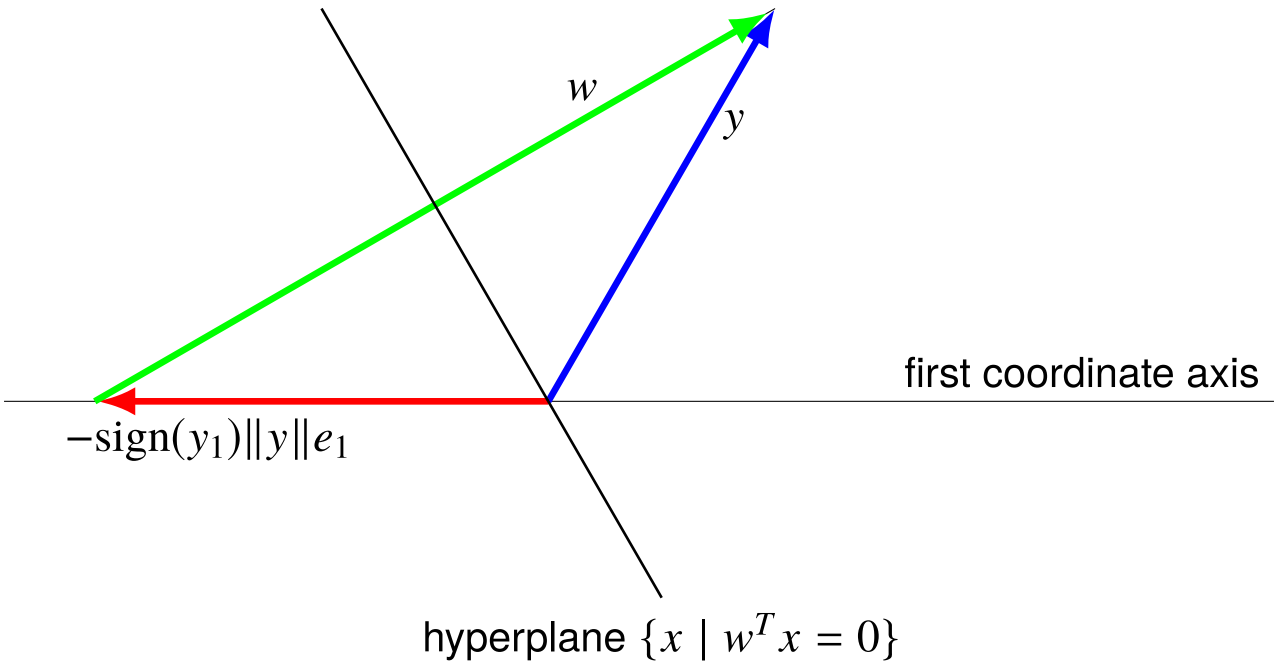
- we define $\operatorname{sign}(0) = 1$
- vector w satisfies

$$\|w\|^2 = 2\,(w^T y) = 2\|y\|\,(\|y\| + |y_1|)$$

- reflector $H = I - 2vv^T$ maps y to multiple of $e_1 = (1, 0, \ldots, 0)$:

$$Hy = y - \frac{2(w^T y)}{\|w\|^2}w = y - w = -\operatorname{sign}(y_1)\|y\|e_1$$

Geometry



the reflection through the hyperplane $\{x \mid w^T x = 0\}$ with normal vector

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

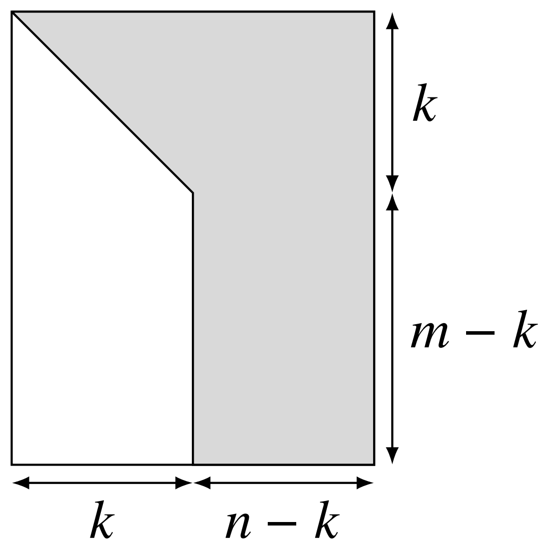
maps y to the vector $-\mathrm{sign}(y_1)\|y\|e_1$

Householder triangularization

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

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

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



(elements in positions i, j for $i > j$ and $j \leq k$ are zero)

Householder algorithm

the following algorithm overwrites A with $\begin{bmatrix} R \\ 0 \end{bmatrix}$

Algorithm: for $k = 1$ to n,

1. define $y = A_{k:m,k}$ and compute $(m - k + 1)$-vector v_k:

$$w = y + \mathrm{sign}(y_1)\|y\|e_1, \qquad v_k = \frac{1}{\|w\|}w$$

2. multiply $A_{k:m,k:n}$ with reflector $I - 2v_kv_k^T$:

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

(see page 109 in textbook for “slice” notation for submatrices)

Comments

- in step 2 we multiply $A_{k:m,k:n}$ with the reflector $I - 2v_k v_k^T$:

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

- this is equivalent to multiplying A with $m \times m$ reflector

$$H_k = \begin{bmatrix} I & 0 \\ 0 & I - 2v_k v_k^T \end{bmatrix} = I - 2 \begin{bmatrix} 0 \\ v_k \end{bmatrix} \begin{bmatrix} 0 \\ v_k \end{bmatrix}^T$$

- algorithm overwrites A with

$$\begin{bmatrix} R \\ 0 \end{bmatrix}$$

and returns the vectors $v_1, \ldots, v_n$, with v_k of length $m - k + 1$

Example

example on page 6.8:

$$A = \begin{bmatrix} -1 & -1 & 1 \\ 1 & 3 & 3 \\ -1 & -1 & 5 \\ 1 & 3 & 7 \end{bmatrix} = H_1 H_2 H_3 \begin{bmatrix} R \\ 0 \end{bmatrix}$$

we compute reflectors H_1, H_2, H_3 that triangularize A:

$$H_3 H_2 H_1 A = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ 0 & R_{22} & R_{23} \\ 0 & 0 & R_{33} \\ 0 & 0 & 0 \end{bmatrix}$$

Example

First column of R

- compute reflector that maps first column of A to multiple of e_1:

$$y = \begin{bmatrix} -1 \\ 1 \\ -1 \\ 1 \end{bmatrix}, \quad w = y - \|y\|e_1 = \begin{bmatrix} -3 \\ 1 \\ -1 \\ 1 \end{bmatrix}, \quad v_1 = \frac{1}{\|w\|}w = \frac{1}{2\sqrt{3}} \begin{bmatrix} -3 \\ 1 \\ -1 \\ 1 \end{bmatrix}$$

- overwrite A with product of $I - 2v_1v_1^T$ and A

$$A := (I - 2v_1v_1^T)A = \begin{bmatrix} 2 & 4 & 2 \\ 0 & 4/3 & 8/3 \\ 0 & 2/3 & 16/3 \\ 0 & 4/3 & 20/3 \end{bmatrix}$$

Example

Second column of R

- compute reflector that maps $A_{2:4,2}$ to multiple of e_1:

$$y = \begin{bmatrix} 4/3 \\ 2/3 \\ 4/3 \end{bmatrix}, \quad w = y + \|y\|e_1 = \begin{bmatrix} 10/3 \\ 2/3 \\ 4/3 \end{bmatrix}, \quad v_2 = \frac{1}{\|w\|}w = \frac{1}{\sqrt{30}} \begin{bmatrix} 5 \\ 1 \\ 2 \end{bmatrix}$$

- overwrite $A_{2:4,2:3}$ with product of $I - 2v_2v_2^T$ and $A_{2:4,2:3}$:

$$A := \begin{bmatrix} 1 & 0 \\ 0 & I - 2v_2v_2^T \end{bmatrix} A = \begin{bmatrix} 2 & 4 & 2 \\ 0 & -2 & -8 \\ 0 & 0 & 16/5 \\ 0 & 0 & 12/5 \end{bmatrix}$$

Example

Third column of R

- compute reflector that maps $A_{3:4,3}$ to multiple of e_1:

$$y = \begin{bmatrix} 16/5 \\ 12/5 \end{bmatrix}, \quad w = y + \|y\| e_1 = \begin{bmatrix} 36/5 \\ 12/5 \end{bmatrix}, \quad v_3 = \frac{1}{\|w\|} w = \frac{1}{\sqrt{10}} \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

- overwrite $A_{3:4,3}$ with product of $I - 2v_3v_3^T$ and $A_{3:4,3}$:

$$A := \begin{bmatrix} I & 0 \\ 0 & I - 2v_3v_3^T \end{bmatrix} A = \begin{bmatrix} 2 & 4 & 2 \\ 0 & -2 & -8 \\ 0 & 0 & -4 \\ 0 & 0 & 0 \end{bmatrix}$$

Example

Final result

$$
\begin{aligned}
H_3H_2H_1A &= \begin{bmatrix} I & 0 \\ 0 & I - 2v_3v_3^T \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & I - 2v_2v_2^T \end{bmatrix} (I - 2v_1v_1^T)A \\
&= \begin{bmatrix} I & 0 \\ 0 & I - 2v_3v_3^T \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & I - 2v_2v_2^T \end{bmatrix} \begin{bmatrix} 2 & 4 & 2 \\ 0 & 4/3 & 8/3 \\ 0 & 2/3 & 16/3 \\ 0 & 4/3 & 20/3 \end{bmatrix} \\
&= \begin{bmatrix} I & 0 \\ 0 & I - 2v_3v_3^T \end{bmatrix} \begin{bmatrix} 2 & 4 & 2 \\ 0 & -2 & -8 \\ 0 & 0 & 16/5 \\ 0 & 0 & 12/5 \end{bmatrix} \\
&= \begin{bmatrix} 2 & 4 & 2 \\ 0 & -2 & -8 \\ 0 & 0 & -4 \\ 0 & 0 & 0 \end{bmatrix}
\end{aligned}
$$

Complexity

Complexity in cycle k (of algorithm on page 6.30): the dominant terms are

- $(2(m-k+1)-1)(n-k+1)$ flops for product $v_k^T(A_{k:m,k:n})$
- $(m-k+1)(n-k+1)$ flops for outer product with v_k
- $(m-k+1)(n-k+1)$ flops for subtraction from $A_{k:m,k:n}$

sum is roughly $4(m-k+1)(n-k+1)$ flops

Total for computing R and vectors $v_1, \ldots, v_n$:

$$\begin{aligned} \sum_{k=1}^{n} 4(m-k+1)(n-k+1) &\approx \int_0^n 4(m-t)(n-t)dt \\ &= 2mn^2 - \frac{2}{3}n^3 \quad \text{flops} \end{aligned}$$

Q-factor

the Householder algorithm returns the vectors $v_1, \ldots, v_n$ that define

$$\begin{bmatrix} Q & \tilde{Q} \end{bmatrix} = H_1 H_2 \cdots H_n$$

- usually there is no need to compute the matrix $\begin{bmatrix} Q & \tilde{Q} \end{bmatrix}$ explicitly
- the vectors $v_1, \ldots, v_n$ are an economical representation of $\begin{bmatrix} Q & \tilde{Q} \end{bmatrix}$
- products with $\begin{bmatrix} Q & \tilde{Q} \end{bmatrix}$ or its transpose can be computed as

$$\begin{bmatrix} Q & \tilde{Q} \end{bmatrix} x = H_1 H_2 \cdots H_n x$$

$$\begin{bmatrix} Q & \tilde{Q} \end{bmatrix}^T y = H_n H_{n-1} \cdots H_1 y$$

Multiplication with Q-factor

- the matrix-vector product $H_k x$ is defined as

$$H_k x = \begin{bmatrix} I & 0 \\ 0 & I - 2v_k v_k^T \end{bmatrix} \begin{bmatrix} x_{1:k-1} \\ x_{k:m} \end{bmatrix} = \begin{bmatrix} x_{1:k-1} \\ x_{k:m} - 2(v_k^T x_{k:m})v_k \end{bmatrix}$$

- complexity of multiplication $H_k x$ is $4(m - k + 1)$ flops:

- complexity of multiplication with $H_1 H_2 \cdots H_n$ or its transpose is

$$\sum_{k=1}^{n} 4(m - k + 1) \approx 4mn - 2n^2 \text{ flops}$$

- roughly equal to matrix-vector product with $m \times n$ matrix ($2mn$ flops)