II.3 QR factorisation

Let $A \in \mathbb{C}^{m \times n}$ be a rectangular or square matrix such that $m \geq n$ (i.e. more rows then columns). In this chapter we consider two closely related factorisations:

1. The *QR factorisation*

$$A = QR = \underbrace{\left[\mathbf{q}_1 | \cdots | \mathbf{q}_m \right]}_{Q \in U(m)} \underbrace{\begin{bmatrix} \times & \cdots & \times \\ & \ddots & \vdots \\ & & \times \\ & & 0 \\ & & \vdots \\ & & 0 \end{bmatrix}}_{R \in \mathbb{C}^{m \times n}}$$

where Q is unitary (i.e., $Q \in U(m)$, satisfying $Q^\star Q = I$, with columns $\mathbf{q}_j \in \mathbb{C}^m$) and R is *right triangular*, which means it is only nonzero on or to the right of the diagonal ($r_{kj} = 0$ if $k > j$).

2. The *reduced QR factorisation*

$$A = \hat{Q}\hat{R} = \underbrace{\left[\mathbf{q}_1 | \cdots | \mathbf{q}_n \right]}_{\hat{Q} \in \mathbb{C}^{m \times n}} \underbrace{\begin{bmatrix} \times & \cdots & \times \\ & \ddots & \vdots \\ & & \times \end{bmatrix}}_{\hat{R} \in \mathbb{C}^{n \times n}}$$

where Q has orthogonal columns ($Q^\star Q = I$, $\mathbf{q}_j \in \mathbb{C}^m$) and $\hat{R}$ is upper triangular.

Note for a square matrix the reduced QR factorisation is equivalent to the QR factorisation, in which case R is *upper triangular*. The importance of these decomposition for square matrices is that their component pieces are easy to invert:

$$A = QR \qquad \Rightarrow \qquad A^{-1}\mathbf{b} = R^{-1}Q^\top \mathbf{b}$$

and we saw in the last two chapters that triangular and orthogonal matrices are easy to invert when applied to a vector $\mathbf{b}$, e.g., using forward/back-substitution.

For rectangular matrices we will see that they lead to efficient solutions to the *least squares problem*: find $\mathbf{x}$ that minimizes the 2-norm

$$\|A\mathbf{x} - \mathbf{b}\|.$$

Note in the rectangular case the QR decomposition contains within it the reduced QR decomposition:

$$A = QR = \left[\hat{Q} | \mathbf{q}_{n+1} | \cdots | \mathbf{q}_m \right] \begin{bmatrix} \hat{R} \\ \mathbf{0}_{m-n \times n} \end{bmatrix} = \hat{Q}\hat{R}.$$

In this lecture we discuss the followng:

1. QR and least squares: We discuss the QR decomposition and its usage in solving least squares problems.
2. Reduced QR and Gram–Schmidt: We discuss computation of the Reduced QR decomposition using Gram–Schmidt.
3. Householder reflections and QR: We discuss computing the QR decomposition using Householder reflections.

In [1]:
```
using LinearAlgebra, Plots, BenchmarkTools
```

1. QR and least squares

Here we consider rectangular matrices with more rows than columns. Given $A \in \mathbb{C}^{m \times n}$ and $\mathbf{b} \in \mathbb{C}^m$, least squares consists of finding a vector $\mathbf{x} \in \mathbb{C}^n$ that minimises the 2-norm: $\|A\mathbf{x} - \mathbf{b}\|$.

Theorem 1 (least squares via QR) Suppose $A \in \mathbb{C}^{m \times n}$ has full rank. Given a QR decomposition $A = QR$ then

$$\mathbf{x} = \hat{R}^{-1} \hat{Q}^\star \mathbf{b}$$

minimises $\|A\mathbf{x} - \mathbf{b}\|$.

Proof

The norm-preserving property (see PS4 Q3.1) of unitary matrices tells us

$$\|A\mathbf{x} - \mathbf{b}\| = \|QR\mathbf{x} - \mathbf{b}\| = \|Q(R\mathbf{x} - Q^\star\mathbf{b})\| = \|R\mathbf{x} - Q^\star\mathbf{b}\| = \left\| \begin{bmatrix} \hat{R} \\ \mathbf{0}_{m-n \times n} \end{bmatrix} \mathbf{x} - \begin{bmatrix} (\\ \mathbf{q} \\ \\ (\end{bmatrix} \right.$$

Now note that the rows $k > n$ are independent of $\mathbf{x}$ and are a fixed contribution. Thus to minimise this norm it suffices to drop them and minimise:

$$\|\hat{R}\mathbf{x} - \hat{Q}^\star \mathbf{b}\|$$

This norm is minimised if it is zero. Provided the column rank of A is full, $\hat{R}$ will be invertible (Exercise: why is this?).

■

Example 1 (quadratic fit) Suppose we want to fit noisy data by a quadratic

$$p(x) = p_0 + p_1 x + p_2 x^2$$

That is, we want to choose p_0, p_1, p_2 at data samples $x_1, \ldots, x_m$ so that the following is true:

$$p_0 + p_1 x_k + p_2 x_k^2 \approx f_k$$

where f_k are given by data. We can reinterpret this as a least squares problem: minimise the norm

$$\left\| \begin{bmatrix} 1 & x_1 & x_1^2 \\ \vdots & \vdots & \vdots \\ 1 & x_m & x_m^2 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \end{bmatrix} - \begin{bmatrix} f_1 \\ \vdots \\ f_m \end{bmatrix} \right\|$$

We can solve this using the QR decomposition:

In [2]:
```
m,n = 100,3

x = range(0,1; length=m) # 100 points
f = 2 .+ x .+ 2x.^2 .+ 0.1 .* randn.() # Noisy quadratic

A = x .^ (0:2)'  # 100 x 3 matrix, equivalent to [ones(m) x x.^2]
Q,R̂ = qr(A)
Q̂ = Q[:,1:n] # Q represents full orthogonal matrix so we take first 3 column
 
p₀,p₁,p₂ = R̂ \ Q̂'f
```

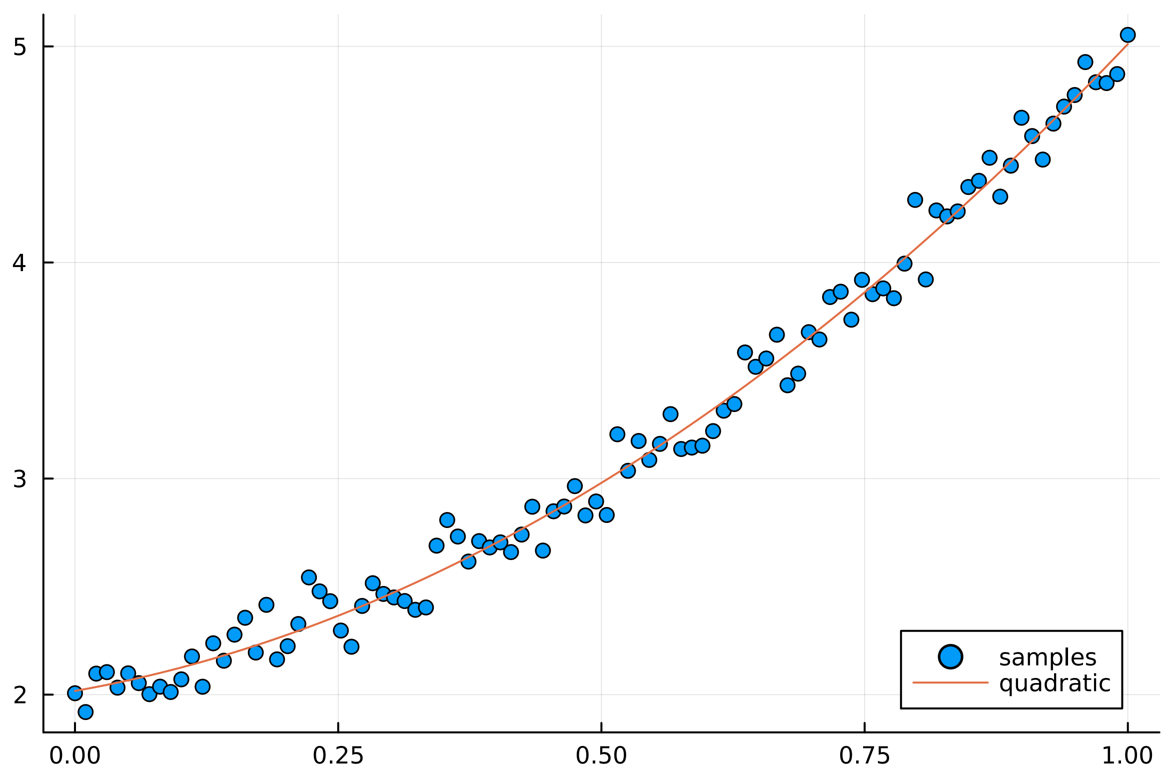
Out[2]:
```
3-element Vector{Float64}:
 2.0172265583556173
 0.857813720251557
 2.1358409286607394
```

We can visualise the fit:

In [3]:
```
p = x -> p₀ + p₁*x + p₂*x^2

scatter(x, f; label="samples", legend=:bottomright)
plot!(x, p.(x); label="quadratic")
```

Out[3]:



Note that `\` with a rectangular system does least squares by default:

In [4]:
```
A \ f
```

Out[4]:
```
3-element Vector{Float64}:
 2.0172265583556177
 0.8578137202515559
 2.13584092866074
```

2. Reduced QR and Gram–Schmidt

How do we compute the QR decomposition? We begin with a method you may have seen before in another guise. Write

$$A = [\,\mathbf{a}_1 | \cdots | \mathbf{a}_n\,]$$

where $\mathbf{a}_k \in \mathbb{C}^m$ and assume they are linearly independent (A has full column rank).

Proposition 1 (Column spaces match) Suppose $A = \hat{Q}\hat{R}$ where $\hat{Q} = [\mathbf{q}_1|\dots|\mathbf{q}_n]$ has orthogonal columns and $\hat{R}$ is upper-triangular, and A has full rank. Then the first j columns of $\hat{Q}$ span the same space as the first j columns of A:

$$\mathrm{span}(\mathbf{a}_1, \dots, \mathbf{a}_j) = \mathrm{span}(\mathbf{q}_1, \dots, \mathbf{q}_j).$$

Proof

Because A has full rank we know $\hat{R}$ is invertible, i.e. its diagonal entries do not vanish: $r_{jj} \neq 0$. If $\mathbf{v} \in \operatorname{span}(\mathbf{a}_1, \ldots, \mathbf{a}_j)$ we have for $\mathbf{c} \in \mathbb{C}^j$

$$\mathbf{v} = \left[\, \mathbf{a}_1 | \cdots | \mathbf{a}_j \,\right] \mathbf{c} = \left[\, \mathbf{q}_1 | \cdots | \mathbf{q}_j \,\right] \hat{R}[1:j, 1:j] \mathbf{c} \in \operatorname{span}(\mathbf{q}_1, \ldots, \mathbf{q}_j)$$

while if $\mathbf{w} \in \operatorname{span}(\mathbf{q}_1, \ldots, \mathbf{q}_j)$ we have for $\mathbf{d} \in \mathbb{R}^j$

$$\mathbf{w} = \left[\, \mathbf{q}_1 | \cdots | \mathbf{q}_j \,\right] \mathbf{d} = \left[\, \mathbf{a}_1 | \cdots | \mathbf{a}_j \,\right] \hat{R}[1:j, 1:j]^{-1} \mathbf{d} \in \operatorname{span}(\mathbf{a}_1, \ldots, \mathbf{a}_j).$$

∎

It is possible to find $\hat{Q}$ and $\hat{R}$ the using the *Gram–Schmidt algorithm*. We construct it column-by-column:

Algorithm 1 (Gram–Schmidt) For $j = 1, 2, \ldots, n$ define

$$\begin{aligned} \mathbf{v}_j &:= \mathbf{a}_j - \sum_{k=1}^{j-1} \underbrace{\mathbf{q}_k^\star \mathbf{a}_j}_{r_{kj}} \mathbf{q}_k \\ r_{jj} &:= \|\mathbf{v}_j\| \\ \mathbf{q}_j &:= \frac{\mathbf{v}_j}{r_{jj}} \end{aligned}$$

Theorem 2 (Gram–Schmidt and reduced QR) Define $\mathbf{q}_j$ and r_{kj} as in Algorithm 1 (with $r_{kj} = 0$ if $k > j$). Then a reduced QR decomposition is given by:

$$A = \underbrace{\left[\, \mathbf{q}_1 | \cdots | \mathbf{q}_n \,\right]}_{\hat{Q} \in \mathbb{C}^{m \times n}} \underbrace{\begin{bmatrix} r_{11} & \cdots & r_{1n} \\ & \ddots & \vdots \\ & & r_{nn} \end{bmatrix}}_{\hat{R} \in \mathbb{C}^{n \times n}}$$

Proof

We first show that $\hat{Q}$ has orthogonal columns. Assume that $\mathbf{q}_\ell^\star \mathbf{q}_k = \delta_{\ell k}$ for $k, \ell < j$. For $\ell < j$ we then have

$$\mathbf{q}_\ell^\star \mathbf{v}_j = \mathbf{q}_\ell^\star \mathbf{a}_j - \sum_{k=1}^{j-1} \mathbf{q}_\ell^\star \mathbf{q}_k \mathbf{q}_k^\star \mathbf{a}_j = 0$$

hence $\mathbf{q}_\ell^\star \mathbf{q}_j = 0$ and indeed $\hat{Q}$ has orthogonal columns. Further: from the definition of $\mathbf{v}_j$ we find

$$\mathbf{a}_j = \mathbf{v}_j + \sum_{k=1}^{j-1} r_{kj} \mathbf{q}_k = \sum_{k=1}^{j} r_{kj} \mathbf{q}_k = \hat{Q} \hat{R} \mathbf{e}_j$$

∎

Gram–Schmidt in action

We are going to compute the reduced QR of a random matrix

In [5]:
```
m,n = 5,4
A = randn(m,n)
Q,R̂ = qr(A)
Q̂ = Q[:,1:n]
```

Out[5]:
```
5×4 Matrix{Float64}:
 -0.70943    -0.454043   0.41333    0.344949
 -0.0439827  -0.275512  -0.320392  -0.138942
 -0.360728    0.548527  -0.493729   0.56686
 -0.419119    0.592301   0.369346  -0.541286
 -0.434729   -0.257365  -0.588492  -0.497378
```

The first column of `\hat Q` is indeed a normalised first column of `A`:

In [6]:
```
R = zeros(n,n)
Q = zeros(m,n)
R[1,1] = norm(A[:,1])
Q[:,1] = A[:,1]/R[1,1]
```

Out[6]:
```
5-element Vector{Float64}:
 0.7094298199902744
 0.043982737231058715
 0.36072796028194515
 0.4191187281458683
 0.4347294327768107
```

We now determine the next entries as

In [7]:
```
R[1,2] = Q[:,1]'A[:,2]
v = A[:,2] - Q[:,1]*R[1,2]
R[2,2] = norm(v)
Q[:,2] = v/R[2,2]
```

Out[7]:
```
5-element Vector{Float64}:
 -0.4540426563561427
 -0.27551156613455136
  0.5485266393297886
  0.5923009397834829
 -0.2573650438818667
```

And the third column is then:

In [8]:
```
R[1,3] = Q[:,1]'A[:,3]
R[2,3] = Q[:,2]'A[:,3]
v = A[:,3] - Q[:,1:2]*R[1:2,3]
R[3,3] = norm(v)
Q[:,3] = v/R[3,3]
```

Out[8]:
```
5-element Vector{Float64}:
 -0.41332990163830907
  0.3203921940446482
  0.49372888242968505
 -0.3693457774967121
  0.5884919045382065
```

(Note the signs may not necessarily match.)

We can clean this up as a simple algorithm:

In [9]:
```
function gramschmidt(A)
    m,n = size(A)
    m ≥ n || error("Not supported")
    R = zeros(n,n)
    Q = zeros(m,n)
    for j = 1:n
        for k = 1:j-1
            R[k,j] = Q[:,k]'*A[:,j]
        end
        v = A[:,j] - Q[:,1:j-1]*R[1:j-1,j]
        R[j,j] = norm(v)
        Q[:,j] = v/R[j,j]
    end
    Q,R
end

Q,R = gramschmidt(A)
norm(A - Q*R)
```

Out[9]:
```
7.850462293418876e-17
```

Complexity and stability

We see within the `for j = 1:n` loop that we have $O(mj)$ operations. Thus the total complexity is $O(mn^2)$ operations.

Unfortunately, the Gram–Schmidt algorithm is *unstable*: the rounding errors when implemented in floating point accumulate in a way that we lose orthogonality:

In [10]:
```
A = randn(300,300)
Q,R = gramschmidt(A)
norm(Q'Q-I)
```

Out[10]:
```
1.6958205615505476e-12
```

3. Householder reflections and QR

As an alternative, we will consider using Householder reflections to introduce zeros below the diagonal. Thus, if Gram–Schmidt is a process of *triangular orthogonalisation* (using triangular matrices to orthogonalise), Householder reflections is a process of *orthogonal triangularisation* (using orthogonal matrices to triangularise).

Consider multiplication by the Householder reflection corresponding to the first column, that is, for

$$Q_1 := Q_{\mathbf{a}_1}^{\mathrm{H}},$$

consider

$$Q_1 A = \begin{bmatrix} \times & \times & \cdots & \times \\ & \times & \cdots & \times \\ & \vdots & \ddots & \vdots \\ & \times & \cdots & \times \end{bmatrix} = \begin{bmatrix} \alpha & \mathbf{w}^\top \\ & A_2 \end{bmatrix}$$

where

$$\alpha := -\mathrm{csign}(a_{11})\|\mathbf{a}_1\|, \mathbf{w} = (Q_1 A)[1, 2:n] \qquad \text{and} \qquad A_2 = (Q_1 A)[2:m, 2:n],$$

$\mathrm{csign}(z) := \mathrm{e}^{\mathrm{i}\arg z}$. That is, we have made the first column triangular. In terms of an algorithm, we then introduce zeros into the first column of A_2, leaving an A_3, and so-on. But we can wrap this iterative algorithm into a simple proof by induction:

Theorem 3 (QR) Every matrix $A \in \mathbb{C}^{m\times n}$ has a QR factorisation:

$$A = QR$$

where $Q \in U(m)$ and $R \in \mathbb{C}^{m\times n}$ is right triangular.

Proof

Assume $m \geq n$. If $A = [\mathbf{a}_1] \in \mathbb{C}^{m\times 1}$ then we have for the Householder reflection $Q_1 = Q_{\mathbf{a}_1}^{\mathrm{H}}$

$$Q_1 A = [\alpha \mathbf{e}_1]$$

which is right triangular, where $\alpha = -\mathrm{sign}(a_{11})\|\mathbf{a}_1\|$. In other words

$$A = \underbrace{Q_1}_{Q} \underbrace{[\alpha \mathbf{e}_1]}_{R}.$$

For $n > 1$, assume every matrix with less columns than n has a QR factorisation. For $A = [\mathbf{a}_1|\ldots|\mathbf{a}_n] \in \mathbb{C}^{m\times n}$, let $Q_1 = Q_{\mathbf{a}_1}^{\mathrm{H}}$ so that

$$Q_1 A = \begin{bmatrix} \alpha & \mathbf{w}^\top \\ & A_2 \end{bmatrix}$$

where $A_2 = (Q_1 A)[2:m, 2:n]$ and $\mathbf{w} = (Q_1 A)[1, 2:n]$. By assumption $A_2 = \tilde{Q}\tilde{R}$. Thus we have

$$
\begin{aligned}
A &= Q_1 \begin{bmatrix} \alpha & \mathbf{w}^\top \\ & \tilde{Q}\tilde{R} \end{bmatrix} \\
&= \underbrace{Q_1 \begin{bmatrix} 1 & \\ & \tilde{Q} \end{bmatrix}}_{Q} \underbrace{\begin{bmatrix} \alpha & \mathbf{w}^\top \\ & \tilde{R} \end{bmatrix}}_{R}.
\end{aligned}
$$

∎

This proof by induction leads naturally to an iterative algorithm. Note that $\tilde{Q}$ is a product of all Householder reflections that come afterwards, that is, we can think of Q as:

$$
Q = Q_1 \tilde{Q}_2 \tilde{Q}_3 \cdots \tilde{Q}_n \qquad \text{for} \qquad \tilde{Q}_j = \begin{bmatrix} I_{j-1} & \\ & Q_j \end{bmatrix}
$$

where Q_j is a single Householder reflection corresponding to the first column of A_j. This is stated cleanly in Julia code:

Algorithm 2 (QR via Householder) For $A \in \mathbb{C}^{m \times n}$ with $m \geq n$, the QR factorisation can be implemented as follows:

In [11]:

```julia
function householderreflection(x)
    y = copy(x)
    if x[1] == 0
        y[1] += norm(x)
    else # note sign(z) = exp(im*angle(z)) where `angle` is the argument of
        y[1] += sign(x[1])*norm(x)
    end
    w = y/norm(y)
    I - 2*w*w'
end
function householderqr(A)
    T = eltype(A)
    m,n = size(A)
    if n > m
        error("More columns than rows is not supported")
    end

    R = zeros(T, m, n)
    Q = Matrix(one(T)*I, m, m)
    Aⱼ = copy(A)

    for j = 1:n
        𝐚₁ = Aⱼ[:,1] # first columns of Aⱼ
        Q₁ = householderreflection(𝐚₁)
        Q₁Aⱼ = Q₁*Aⱼ
        α,𝐰 = Q₁Aⱼ[1,1],Q₁Aⱼ[1,2:end]
        Aⱼ₊₁ = Q₁Aⱼ[2:end,2:end]

        # populate returned data
        R[j,j] = α
        R[j,j+1:end] = 𝐰
```

```
        # following is equivalent to Q = Q*[I 0 ; 0 Qⱼ]
        Q[:,j:end] = Q[:,j:end]*Q₁

        Aⱼ = Aⱼ₊₁ # this is the "induction"
    end
    Q,R
end

m,n = 100,50
A = randn(m,n)
Q,R = householderqr(A)
@test Q'Q ≈ I
@test Q*R ≈ A
```

Out[11]: **Test Passed**

Note because we are forming a full matrix representation of each Householder reflection this is a slow algorithm, taking $O(n^4)$ operations. The problem sheet will consider a better implementation that takes $O(n^3)$ operations.

Example 2 We will now do an example by hand. Consider the 4×3 matrix

$$A = \begin{bmatrix} 4 & 2 & -1 \\ 0 & 15 & 18 \\ -2 & -4 & -4 \\ -2 & -4 & -10 \end{bmatrix}$$

For the first column we have

$$Q_1 = I - \frac{1}{12} \begin{bmatrix} 4 \\ 0 \\ -2 \\ -2 \end{bmatrix} [4 \quad 0 \quad -2 \quad -2] = \frac{1}{3} \begin{bmatrix} -1 & 0 & 2 & 2 \\ 0 & 3 & 0 & 0 \\ 2 & 0 & 2 & -1 \\ 2 & 0 & -1 & 2 \end{bmatrix}$$

so that

$$Q_1 A = \begin{bmatrix} -3 & -6 & -9 \\ & 15 & 18 \\ & 0 & 0 \\ & 0 & -6 \end{bmatrix}$$

In this example the next column is already upper-triangular, but because of our choice of reflection we will end up swapping the sign, that is

$$\tilde{Q}_2 = \begin{bmatrix} 1 & & & \\ & -1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix}$$

so that

$$\tilde{Q}_2 Q_1 A = \begin{bmatrix} -3 & -6 & -9 \\ & -15 & -18 \\ & 0 & 0 \\ & 0 & -6 \end{bmatrix}$$

The final reflection is

$$\tilde{Q}_3 = \begin{bmatrix} I_{2\times 2} & \\ & I - \begin{bmatrix} 1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & -1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \text{å}1 & & & \\ & 1 & & \\ & & 0 & 1 \\ & & 1 & 0 \end{bmatrix}$$

giving us

$$\tilde{Q}_3 \tilde{Q}_2 Q_1 A = \underbrace{\begin{bmatrix} -3 & -6 & -9 \\ & -15 & -18 \\ & & -6 \\ & & 0 \end{bmatrix}}_{R}$$

That is,

$$A = Q_1 \tilde{Q}_2 \tilde{Q}_3 R = \underbrace{\frac{1}{3} \begin{bmatrix} -1 & 0 & 2 & 2 \\ 0 & 3 & 0 & 0 \\ 2 & 0 & -1 & 2 \\ 2 & 0 & 2 & -1 \end{bmatrix}}_{Q} \underbrace{\begin{bmatrix} -3 & -6 & -9 \\ & -15 & -18 \\ & & -6 \\ & & 0 \end{bmatrix}}_{R}$$

$$= \underbrace{\frac{1}{3} \begin{bmatrix} -1 & 0 & 2 \\ 0 & 3 & 0 \\ 2 & 0 & -1 \\ 2 & 0 & 2 \end{bmatrix}}_{\hat{Q}} \underbrace{\begin{bmatrix} -3 & -6 & -9 \\ & -15 & -18 \\ & & -6 \end{bmatrix}}_{\hat{R}}$$