QR factorization example

In [2]:
```
a1=[1,2,3]
```

Out[2]:
```
3-element Array{Int64,1}:
 1
 2
 3
```

In [3]:
```
a2=[2,-1,-3]
```

Out[3]:
```
3-element Array{Int64,1}:
  2
 -1
 -3
```

In [4]:
```
a3=[1,1,1]
```

Out[4]:
```
3-element Array{Int64,1}:
 1
 1
 1
```

In [5]:
```
tq1=copy(a1)
```

Out[5]:
```
3-element Array{Int64,1}:
 1
 2
 3
```

In [7]:
```
using LinearAlgebra
```

In [8]:
```
q1=tq1/norm(tq1)
```

Out[8]:
```
3-element Array{Float64,1}:
 0.2672612419124244
 0.5345224838248488
 0.8017837257372732
```

In [9]:
```
[1,2,3]/sqrt(14)
```

Out[9]:
```
3-element Array{Float64,1}:
 0.2672612419124244
```

```
0.5345224838248488
```

In [10]:
```
tq2=a2-(q1'*a2)*q1
```

Out[10]:
```
3-element Array{Float64,1}:
  2.642857142857143
  0.2857142857142858
 -1.0714285714285714
```

In [11]:
```
q2=tq2/norm(tq2)
```

Out[11]:
```
3-element Array{Float64,1}:
  0.9221228546165697
  0.09968895725584541
 -0.37383358970942016
```

In [12]:
```
tq3=a3-(q1'*a3)*q1-(q2'*a3)*q2
```

Out[12]:
```
3-element Array{Float64,1}:
 -0.026086956521739313
  0.07826086956521729
 -0.043478260869565244
```

In [13]:
```
q3=tq3/norm(tq3)
```

Out[13]:
```
3-element Array{Float64,1}:
 -0.27975144247209616
  0.8392543274162815
 -0.4662524041201573
```

In [14]:
```
A=[a1 a2 a3]
```

Out[14]:
```
3×3 Array{Int64,2}:
 1   2  1
 2  -1  1
 3  -3  1
```

In [15]:
```
Q=[q1 q2 q3]
```

Out[15]:
```
3×3 Array{Float64,2}:
 0.267261   0.922123  -0.279751
 0.534522   0.099689   0.839254
 0.801784  -0.373834  -0.466252
```

In [16]:
```
Q'*Q
```

Out[16]:
```
3×3 Array{Float64,2}:
  1.0           0.0          -1.33227e-15
  0.0           1.0          -1.80411e-15
 -1.33227e-15  -1.80411e-15   1.0
```

Note that e-15 means 10^{-15} so those off-diagonal entries are zero to within the limits of computer rounding errors. Thus $Q^TQ \approx I$ except for rounding.

In [17]:
```
R=Q'*A
```

Out[17]:
```
3×3 Array{Float64,2}:
  3.74166      -2.40535     1.60357
  2.22045e-16   2.86606     0.647978
 -5.10703e-15  -1.9984e-15  0.0932505
```

Note that the three numbers in the lower left corner of the matrix are essentially zero (up to rounding errors).

In [18]:
```
q1'*a2
```

Out[18]:
```
-2.4053511772118195
```

In [19]:
```
q2'*a3
```

Out[19]:
```
0.6479782221629949
```

In [20]:
```
norm(tq2)
```

Out[20]:
```
2.866057521105554
```

the QR factorization is so important that Julia has a build in subroutine that just does it.

In [22]:
```
Q2,R2=qr(A)
```

Out[22]:
```
LinearAlgebra.QRCompactWY{Float64,Array{Float64,2}}
Q factor:
3×3 LinearAlgebra.QRCompactWYQ{Float64,Array{Float64,2}}:
 -0.267261   0.922123   0.279751
 -0.534522   0.099689  -0.839254
 -0.801784  -0.373834   0.466252
R factor:
3×3 Array{Float64,2}:
```

```
 -3.74166  2.40535  -1.60357
  0.0      2.86606   0.647978
  0.0      0.0      -0.0932505
```

In [23]:
```
Q2=Matrix(Q2)
```

Out[23]:
```
3×3 Array{Float64,2}:
 -0.267261   0.922123   0.279751
 -0.534522   0.099689  -0.839254
 -0.801784  -0.373834   0.466252
```

In [24]:
```
Q2*R2
```

Out[24]:
```
3×3 Array{Float64,2}:
 1.0   2.0  1.0
 2.0  -1.0  1.0
 3.0  -3.0  1.0
```

In [25]:
```
Q*R
```

Out[25]:
```
3×3 Array{Float64,2}:
 1.0   2.0  1.0
 2.0  -1.0  1.0
 3.0  -3.0  1.0
```

In [26]:
```
Q
```

Out[26]:
```
3×3 Array{Float64,2}:
 0.267261   0.922123  -0.279751
 0.534522   0.099689   0.839254
 0.801784  -0.373834  -0.466252
```

In [27]:
```
Q2
```

Out[27]:
```
3×3 Array{Float64,2}:
 -0.267261   0.922123   0.279751
 -0.534522   0.099689  -0.839254
 -0.801784  -0.373834   0.466252
```

In [28]:
```
R
```

Out[28]:
```
3×3 Array{Float64,2}:
  3.74166      -2.40535     1.60357
  2.22045e-16   2.86606     0.647978
 -5.10703e-15  -1.9984e-15  0.0932505
```

In [29]:
```
R2
```

Out[29]:
```
3×3 Array{Float64,2}:
 -3.74166  2.40535  -1.60357
  0.0      2.86606   0.647978
  0.0      0.0      -0.0932505
```

Note that the sign changes were done by the computer to reduce rouding error. More about that sort of thing and many more things are discussing in Math 466.

Here is an example of a non-square matrix, to anticipate some interesting results about left and right inverses that will be coming up in Chapter 11.

In [31]:
```
A=rand(5,3)
```

Out[31]:
```
5×3 Array{Float64,2}:
 0.96454   0.71409   0.729139
 0.15811   0.551558  0.540658
 0.468164  0.907535  0.198505
 0.95522   0.842411  0.767145
 0.10794   0.447302  0.911846
```

In [33]:
```
Q3,R3=qr(A);
```

In [34]:
```
Q3=Matrix(Q3)
```

Out[34]:
```
5×3 Array{Float64,2}:
 -0.665816    0.313529   0.130694
 -0.109143   -0.541256   0.169316
 -0.323171   -0.61292   -0.632682
 -0.659383    0.126062   0.0663205
 -0.0745102  -0.466025   0.741327
```

In [35]:
```
Q3*R3
```

Out[35]:
```
5×3 Array{Float64,2}:
 0.96454   0.71409   0.729139
 0.15811   0.551558  0.540658
 0.468164  0.907535  0.198505
 0.95522   0.842411  0.767145
 0.10794   0.447302  0.911846
```

In [36]:
```
R3
```

Out[36]:
```
3×3 Array{Float64,2}:
```

```
 -1.44866  -1.41774  -1.18242
  0.0      -0.73315  -0.513931
  0.0       0.0       0.788099
```

In [37]:
```
Q3'*Q3
```

Out[37]:
```
3×3 Array{Float64,2}:
  1.0          -7.35995e-17  -7.36745e-17
 -7.35995e-17   1.0          -7.74265e-17
 -7.36745e-17  -7.74265e-17   1.0
```

In [38]:
```
Q3*Q3'
```

Out[38]:
```
5×5 Array{Float64,2}:
  0.558692     -0.0749024  -0.0596833  0.487219    0.000384543
 -0.0749024     0.333538    0.259895   0.0149642   0.385889
 -0.0596833     0.259895    0.880397   0.0938681  -0.159309
  0.487219      0.0149642   0.0938681  0.455076    0.0395482
  0.000384543   0.385889   -0.159309   0.0395482   0.772297
```

Therefore, $Q^TQ = I$ but you don't get the identity when the order is reversed. This is because $Q \in \mathbf{R}^{5\times 3}$ is not square in this case. On the other hand the Q (denoted by Q2) earlier was square and in that case $QQ^T = I$.

Chapter 11 starts with the definition of left and right inverses because they are different when the matrices are not square.

In [39]:
```
Q2'*Q2
```

Out[39]:
```
3×3 Array{Float64,2}:
 1.0          0.0          1.11022e-16
 0.0          1.0          1.11022e-16
 1.11022e-16  1.11022e-16  1.0
```

In [40]:
```
Q2*Q2'
```

Out[40]:
```
3×3 Array{Float64,2}:
  1.0          -1.11022e-16  0.0
 -1.11022e-16   1.0          0.0
  0.0           0.0          1.0
```

In [ ]:
```

```