Lecture 14: QR factorization and least squares

In this lecture we see that Given's rotations can be used to calculate the QR decomposition line-by-line, and how it can also be used for rectangular matrices.

We consider a (random) 4×4 matrix:

In [77]:

```
A=rand(4,4)
```

Out[77]:

```
4x4 Array{Float64,2}:
 0.779135  0.463434  0.792271  0.960972
 0.74052   0.994196  0.990579  0.486781
 0.052372  0.794275  0.532929  0.943658
 0.176553  0.697142  0.609449  0.402935
```

We need to construct a 4 x 4 orthogonal matrix that acts only the first two rows. We can do so by modifying the identity:

In [79]:

```
a,b=A[1:2,1]

U=[a b;
    -b a]/sqrt(a^2+b^2)

Q1=eye(4)
Q1[1:2,1:2] = U

Q1
```

Out[79]:

```
4x4 Array{Float64,2}:
  0.724841  0.688916  0.0  0.0
 -0.688916  0.724841  0.0  0.0
  0.0       0.0       1.0  0.0
  0.0       0.0       0.0  1.0
```

This satisfies the property that `Q1*A` has a zero in the (2,1) entry. We assign this updated matrix to B:

In [82]:

```
Q1*A
```

Out[82]:

```
4x4 Array{Float64,2}:
  1.0749        1.02083   1.2567     1.0319
 -1.11022e-16  0.401366  0.172204  -0.309191
  0.052372     0.794275  0.532929   0.943658
  0.176553     0.697142  0.609449   0.402935
```

We continue the process to insert a zero in the (3,1) entry by acting on the first and third rows of `B=Q1*A`:

In [83]:

```
B=Q1*A   #
a,b=B[[1,3],1]

U=[a b;
    -b a]/sqrt(a^2+b^2)
Q2=eye(4)
Q2[[1,3],[1,3]] = U

B=Q2*B
```

Out[83]:

```
4x4 Array{Float64,2}:
  1.07618       1.05828   1.28114    1.0766
 -1.11022e-16  0.401366  0.172204  -0.309191
  0.0          0.743655  0.47114    0.892323
  0.176553     0.697142  0.609449   0.402935
```

We now act on the 1st and 4th rows to introduce one more zero:

In [84]:

```
row1=1
row2=4

a,b=B[[row1,row2],1]

U=[a b;
    -b a]/sqrt(a^2+b^2)
Q3=eye(4)
Q3[[row1,row2],[row1,row2]] = U

B=Q3*B
```

Out[84]:

```
4x4 Array{Float64,2}:
  1.09057       1.15718   1.36291    1.12763
 -1.11022e-16  0.401366  0.172204  -0.309191
  0.0          0.743655  0.47114    0.892323
  0.0          0.51662   0.394004   0.223327
```

We now proceed to the 2nd column, introducing zeros entry by entry. The final result is an upper triangular matrix `R`:

In [85]:

```
col=2
row1,row2=2,3

a,b=B[[row1,row2],col]

U=[a b;
    -b a]/sqrt(a^2+b^2)
Q4=eye(4)
Q4[[row1,row2],[row1,row2]] = U

B=Q4*B

col=2
row1,row2=2,4

a,b=B[[row1,row2],col]

U=[a b;
    -b a]/sqrt(a^2+b^2)
Q5=eye(4)
Q5[[row1,row2],[row1,row2]] = U

B=Q5*B

col=3
row1,row2=3,4

a,b=B[[row1,row2],col]

U=[a b;
    -b a]/sqrt(a^2+b^2)

Q6=eye(4)
Q6[[row1,row2],[row1,row2]] = U

R=Q6*B
```

Out[85]:

```
4x4 Array{Float64,2}:
  1.09057        1.15718      1.36291    1.12763
 -4.49897e-17   0.990461     0.629033   0.661164
  8.68201e-17  -4.05456e-17  0.105754   0.371273
 -5.25751e-17  -3.79149e-17  0.0       -0.605585
```

In otherwords,

```
R=Q6*Q5*Q4*Q3*Q2*Q1*A
```

In [86]:

```
Q6*Q5*Q4*Q3*Q2*Q1*A
```

Out[86]:

```
4x4 Array{Float64,2}:
  1.09057      1.15718      1.36291    1.12763 
 -2.77556e-17  0.990461     0.629033   0.661164
  2.77556e-17  0.0          0.105754   0.371273
 -1.38778e-17  1.11022e-16  0.0       -0.605585
```

Thus the `Q` in `QR` is

```
Q=inv(Q6*Q5*Q4*Q3*Q2*Q1)=Q1'*Q2'*Q3'*Q4'*Q5'*Q6'
```

In [35]:

```
Q=Q1'*Q2'*Q3'*Q4'*Q5'*Q6'

norm(Q'*Q-eye(4))
```

Out[35]:

```
2.6923844779914502e-16
```

QR on rectangular matrices

We can apply the same procedure to a rectangular matrix, for example:

In [87]:

```
A[:,1:3]
```

Out[87]:

```
4x3 Array{Float64,2}:
 0.779135  0.463434  0.792271
 0.74052   0.994196  0.990579
 0.052372  0.794275  0.532929
 0.176553  0.697142  0.609449
```

Given's rotations proceed just as before:

In [88]:

```
B=A[:,1:3]


a,b=B[1:2,1]

U=[a b;
    -b a]/sqrt(a^2+b^2)
```

```
Q1=eye(4)

Q1[1:2,1:2] = U


B=Q1*B


a,b=B[[1,3],1]

U=[a b;
    -b a]/sqrt(a^2+b^2)
Q2=eye(4)
Q2[[1,3],[1,3]] = U

B=Q2*B

row1=1
row2=4

a,b=B[[row1,row2],1]

U=[a b;
    -b a]/sqrt(a^2+b^2)
Q3=eye(4)
Q3[[row1,row2],[row1,row2]] = U

B=Q3*B


col=2
row1,row2=2,3

a,b=B[[row1,row2],col]

U=[a b;
    -b a]/sqrt(a^2+b^2)
Q4=eye(4)
Q4[[row1,row2],[row1,row2]] = U

B=Q4*B

col=2
row1,row2=2,4

a,b=B[[row1,row2],col]

U=[a b;
    -b a]/sqrt(a^2+b^2)
Q5=eye(4)
Q5[[row1,row2],[row1,row2]] = U

B=Q5*B

col=3
row1,row2=3,4
```

```
a,b=B[[row1,row2],col]


U=[a b;
    -b a]/sqrt(a^2+b^2)
Q6=eye(4)
Q6[[row1,row2],[row1,row2]] = U

R=Q6*B
```

Out[88]:

```
4x3 Array{Float64,2}:
  1.09057       1.15718      1.36291
 -4.49897e-17   0.990461     0.629033
  8.68201e-17  -4.05456e-17  0.105754
 -5.25751e-17  -3.79149e-17  0.0
```

Now the notion of upper triangular has been changed to allow for rectangular `R`. We can verify that `Q*R` returns the desired result:

In [93]:

```
Q=Q1'*Q2'*Q3'*Q4'*Q5'*Q6'
norm(Q*R-A[:,1:3])
```

Out[93]:

```
6.922150386542620e-16
```

QR and least squares

We can use the fact that the 2-norm is invariant under orthogonal transformations to reduce the problem of least squares to a triangular system:

$$\|A\mathbf{x} - \mathbf{b}\|_2 = \|QR\mathbf{x} - \mathbf{b}\|_2 = \|Q(R\mathbf{x} - Q^\top \mathbf{b})\|_2 = \|R\mathbf{x} - Q^\top \mathbf{b}\|_2$$

Recall that \ is the inbuilt routine for solving least squares:

In [96]:

```
A=rand(4,3)
b=rand(4)

x=A\b
```

Out[96]:

```
3-element Array{Float64,1}:
  0.607407
  0.377505
 -0.49233
```

We can create a QR decomposition as follows:

In [98]:

```
Q,Rsmall=qr(A;thin=false)
R=zeros(4,3)
R[1:3,1:3]=Rsmall

Q,R
```

Out[98]:

```
(
4x4 Array{Float64,2}:
 -0.495448   0.600443    0.545061  -0.3113
 -0.413077  -0.0192636  -0.692123  -0.591576
 -0.600523   0.137519   -0.276001   0.737756
 -0.472516  -0.787519    0.384315  -0.0940497,

4x3 Array{Float64,2}:
 -1.63043  -1.14771   -1.2136
  0.0      -0.500113  -0.221979
  0.0       0.0        0.654625
  0.0       0.0        0.0     )
```

Indeed, we have:

In [99]:

```
(R\(Q'*b))-x
```

Out[99]:

```
3-element Array{Float64,1}:
 -1.11022e-16
  2.77556e-16
 -3.88578e-16
```

While `R` is rectangular, since the bottom rows are zero they do not contribute at all to `R*x`. Thus `R*x` is equivalent to `[R[1:3,1:3]*x; 0]`. And so we can determine that `x` can be found by inverting a square upper triangular matrix (using back substitution):

In [101]:

```
R[1:3,1:3]\(Q'*b)[1:3]
```

Out[101]:

```
3-element Array{Float64,1}:
  0.607407
  0.377505
 -0.49233
```

Thin QR

An alternative version of QR is for Q to be rectangular and R to be square. This is is equivalent to dropping the zero rows of R and taking the corresponding columns of Q: if

$$A = QR = (q_1 | \cdots | q_m | q_{m+1} | \cdots | q_n) \begin{pmatrix} \bar{R} \\ 0_{n-m \times m} \end{pmatrix}$$

where A is $n \times m$ and $\bar{R}$ is $m \times m$ then we also have

$$A = \bar{Q}\bar{R} \qquad \text{for} \qquad \bar{Q} = (q_1 | \cdots | q_n)$$

where $\bar{Q}$ is $n \times m$. This *thin QR* is what is returned by default by `qr`:

In [2]:

```
A=rand(5,3)

Q,R=qr(A)

size(Q),size(R)
```

Out[2]:

```
((5,3),(3,3))
```

Q is orthogonal in the sense that $Q^\top Q = I$:

In [3]:

```
norm(Q'*Q-eye(3))
```

Out[3]:

```
4.483099266212884e-16
```

On the other hand, $QQ^\top \neq I$:

In [76]:

```
Q*Q'
```

Out[76]:

```
4x4 Array{Float64,2}:
  0.942685    0.191028   -0.130566    0.0221612
  0.191028    0.363317    0.435167   -0.0738617
 -0.130566    0.435167    0.702567    0.0504838
  0.0221612  -0.0738617   0.0504838   0.991431
```