$Q^T A = R$

As with the LU factorization of A we have, after $(n-1)$ steps,

$$\begin{aligned} \mathcal{Q}^T A &= \mathcal{Q}^T A_0 \\ &= [Q_1 Q_2 \cdots Q_{n-1}]^T A_0 \\ &= [Q_{n-1} Q_{n-2} \cdots Q_1] A_0 \\ &= (Q_{n-1}(\cdots(Q_2\underbrace{(Q_1 A_0)}_{A_1}))\cdots) \\ &= A_{n-1} \\ &\equiv R \end{aligned}$$

Since $\mathcal{Q}^T A = R$ and $\mathcal{Q}^T = \mathcal{Q}^{-1}$ we have, after multiplying this expression by $\mathcal{Q}$,

$$\mathcal{Q}R = \mathcal{Q}(\mathcal{Q}^T A) = (\mathcal{Q}\mathcal{Q}^T)A = A.$$

Exercise: Show that the operation count for this decomposition is twice that for the LU decomposition.

Solving $Ax = b$ when $A = QR$

To solve the linear system $Ax = b$ using QR we note that Q need not be explicitly computed – it need only be represented by retaining the vectors $u_1, u_2 \cdots u_{n-1}$ and the scalars $\|u_1\|_2^2, \|u_2\|_2^2, \cdots \|u_{n-1}\|_2^2$. We observe that,

$$A = QR = Q_1 Q_2 \cdots Q_{n-1} R,$$

and this allows us to 'solve' $Ax = b$ as

$$QRx = b, \ \text{ or } \ Rx = Q^{-1} b.$$

But

$$Q^{-1} = Q^T = Q_{n-1} Q_{n-2} \cdots Q_1$$

and we have that,

$$Rx = (Q_{n-1}(Q_{n-2}(\cdots(\underbrace{Q_1 b}_{z_1})\cdots)).$$

Efficient Implementation of QR

This suggests the following efficient algorithm:

```
-set $z = b$ ;
-for $j = 1, 2 \cdots (n-1)$
    -set $z = Q_j z$   (‘solve’ $Q_j z_j = z_{j-1}$)
-end
-solve the triangular system $Rx = z$
```

Exercise:

Given that the Q_j are ‘represented’ by the vectors, u_j and the scalars $\|u_j\|_2^2$, determine the operation count for the above algorithm and compare it with the that for the standard LU algorithm. (Recall that computing $Q_j\ v$ for an arbitrary vector v can be done using fact that $Q_j v = v + \gamma u_j,$ where $\gamma = -2\frac{u_j^T v}{\|u_j\|_2^2}$.)

Error Bounds for the QR Method

One can show (analogous to GE) that, if the above QR algorithm is implemented in floating point arithmetic, then the computed solution, $\bar{x}$, will satisfy,

$$(A+E)\bar{x} = b \text{ where } E = (e_{i\,j}),$$

and

$$|e_{i\,j}| \le 1.02 \max_{r=0,1\cdots n-1}[\|A_r\|_2](2n^2+n)\|\mathcal{Q}\|_2\mu.$$

But $\|\mathcal{Q}\|_2 = 1$ and $\|A_r\|_2 \le \|Q_r\|_2\|A_{r-1}\|_2 = \|A_{r-1}\|_2$ for $r = 1, 2\cdots(n-1)$, so

$$\max_{r=0,1\cdots n-1}[\|A_r\|_2] = \|A_0\|_2 = \|A\|_2,$$

and

$$|e_{i\,j}| \le 1.02\|A\|_2(2n^2+2)\mu.$$

Linear Least Squares

In many applications the linear systems of equations that arise do not have the same number of equations as unknowns. We can then can 'solve' the equations in a least squares sense.

- The Basic Problem – Given m linear equations in n unknowns,

$$Ac \approx b, \quad A \in \Re^{m \times n}, \quad c \in \Re^{n}, \quad b \in \Re^{m}.$$

  Determine the vector c that minimises,

$$\|Ac - b\|_2.$$

  Note that other norms ($\| \cdot \|_\infty$, or $\| \cdot \|_1$ for example) are not differentiable and the corresponding minimisation problem is more difficult to analyse and the algorithms more complicated.

An Example - Data Fitting

Let the 'unkown' vector be $c \in \Re^n$, the coefficients defining a polynomial $p_n(s)$,

$$p_n(s) = c_1 + c_2 s \cdots + c_n s^{n-1}.$$

Assume we wish to approximate a function $f(s)$ by $p_n(s)$ and we 'know' that $f(s_i) \equiv f_i$ for $i = 1, 2 \cdots m$. Let $A \in \Re^{m \times n}$ be defined by

$$a_{i\,j} = s_i^{j-1}, \; i = 1, 2 \cdots m, \;\; j = 1, 2 \cdots n.$$

Then it is easy to see that if, $s = (s_1, s_2 \cdots s_m)^T$, $f = (f_1, f_2 \cdots f_m)^T$,

$$Ac = \begin{bmatrix} p_n(s_1) \\ p_n(s_2) \\ \vdots \\ p_n(s_m) \end{bmatrix},$$

and therefore our task is to determine c such that, $\|Ac - f\|_2$ is minimised.

This is a <u>discrete</u> least squares problem.

Data Fitting Ex. (cont.)

- If the s_i are distinct then A has full rank.
- A is a <u>Vandermonde</u> Matrix.
- If $m = n$, A can be badly conditioned.

For the general LLSQ Problem we have 3 cases:

- $m = n$: Standard case with a unique solution if A has full rank, n, (use LU or QR).
- $m > n$: Overdetermined case (more equations than unknowns). A can have full rank (rank n), or it can be rank deficient (rank $< n$).
- $m < n$: Underdetermined case (fewer equations than unknowns). A can have full rank, m, or it can be rank deficient.

The General Overdetermined LLSQ

Let the unknown vector be $x \in \Re^n$. We wish to solve $Ax \approx b$, where $b \in \Re^m$,

$$Ax = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \vdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \approx \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ \vdots \\ b_m \end{bmatrix}.$$

Let

$$\begin{aligned} \Phi(x) &= \|Ax - b\|_2^2 \\ &= (Ax-b)^T(Ax-b) \equiv r^T(x)r(x) \\ &= \sum_{i=1}^{m} r_i^2(x). \end{aligned}$$

Overdetermined Systems (cont.)

From standard results in calculus we know that $\Phi(x)$ is a minimum when,

$$\frac{\partial \Phi}{\partial x_j} = 0 \ \text{ for } \ \ j = 1, 2, \cdots n.$$

Since $\Phi(x) = \sum_{i=1}^{m} r_i^2(x)$ we have,

$$\begin{aligned}
\frac{\partial \Phi}{\partial x_j} &= \frac{\partial}{\partial x_j}\left[\sum_{i=1}^{m} r_i^2(x)\right] \\
&= \sum_{i=1}^{m} \frac{\partial}{\partial x_j}(r_i^2(x)), \\
&= 2\sum_{i=1}^{m} r_i(x)\frac{\partial r_i(x)}{\partial x_j}.
\end{aligned}$$

But $r_i(x) = -b_i + a_{i\,1}x_1 + a_{i\,2}x_2 \cdots + a_{i\,n}x_n$
($= -b_i +$ inner product of i^{th} row of A with x, which $= -b_i +$inner product of i^{th} column of A^T with x).

Overdetermined Systems (cont.)

Therefore we have,

$$\begin{aligned}\frac{\partial r_i(x)}{\partial x_j} &= \frac{\partial}{\partial x_j}\left[Ax-b\right]_i ,\\ &= (a_{i\ j}),\end{aligned}$$

for $i = 1, 2, \cdots m; j = 1, 2, \cdots n$.

It then follows that

$$\frac{\partial \Phi}{\partial x_j} = 2\sum_{i=1}^{m} r_i(x) a_{i\ j} = 2\left(A^T r\right)_j .$$

Therefore to achieve $\frac{\partial \Phi}{\partial x_j} = 0$ for $j = 1, 2, \cdots n$ we must have,

$$\left(A^T \underline{r}\right)_j = 0,\ \text{for } j = 1, 2, \cdots n.$$

This is equivalent to asking that,

$$A^T r = 0 \ \text{ or } \ A^T\left(Ax - b\right) = 0.$$

Normal Equations

Note that the matrix $A^T A$ is $n \times n$ and nonsingular (if rank(A)=n) and we can solve our LLSQ problem with $m > n$ by solving the linear system,

$$\boxed{\boldsymbol{A^T A x = A^T b}}$$

These are the <u>Normal Equations</u>. We have shown that any solution to the LLSQ problem must be a solution to the Normal Equations. The converse is also true (exercise).

- The $(i,j)^{th}$ entry of $A^T A = a_i^T a_j = a_j^T a_i$.
- $A^T A$ is symmetric (since $(A^T A)^T = A^T (A^T)^T = A^T A$).
- The cost to determine $A^T A$ is $[n + (n^2 - n)/2]m = \frac{mn^2+mn}{2}$ flops.

It can be shown that, once $A^T A$ has been formed, solving $A^T A x = A^T b$ can be accomplished in $\frac{n^3}{6} + O(n^2)$ flops.

QR for Least Squares

We will introduce a QR based algorithm that doesn't require the explicit computation of $A^T A$.
Consider forming the $\mathcal{QR}$ factorization (or Schur decomposition) of the $m \times n$ matrix A,

$$A = \mathcal{QR} = (Q_1 Q_2 \cdots Q_n)\mathcal{R},$$

where $\mathcal{Q}$ is an orthogonal matrix and $\mathcal{R}$ is an upper triangular matrix. That is, we will determine $\mathcal{Q}$ as a product of n Householder reflections:

$$\mathcal{Q}A = \mathcal{R} \Leftrightarrow Q_n^T(Q_{n-1}^T \cdots Q_1^T A)) \cdots) = \mathcal{R},$$

where each $Q_i = Q_i^T$ is an $m \times m$ Householder reflection and $\mathcal{R}$ is an $m \times n$ (rectangular) upper triangular matrix,

QR for Least Squares (cont.)

$$\mathcal{R} = \begin{bmatrix} \times & \times & \cdots & \times \\ 0 & \times & \cdots & \times \\ 0 & 0 & \cdots & \times \\ \vdots & \vdots & \vdots & \times \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \equiv \begin{bmatrix} R \\ 0 \end{bmatrix},$$

and R is a square $n \times n$ upper triangular matrix.

With such a factorization of A we have,

$$A^T A = (\mathcal{QR})^T \mathcal{QR} = \mathcal{R}^T \mathcal{Q}^T \mathcal{QR} = \mathcal{R}^T \mathcal{R},$$

QR for Least Squares (cont.)

where

$$\mathcal{R}^T\mathcal{R} = \begin{bmatrix} \times & 0 & 0 & \cdots & 0 \\ \times & \times & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & 0 \\ \times & \times & \times & \cdots & 0 \end{bmatrix} \begin{bmatrix} \times & \times & \cdots & \times \\ 0 & \times & \cdots & \times \\ 0 & 0 & \cdots & \times \\ \vdots & \vdots & \cdots & \times \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

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

an $n \times n$ symmetric matrix.

QR for Least Squares (cont.)

Now solving the Normal Equations to determine x can be done by solving the equivalent linear system,

$$R^T R x = A^T b.$$

This requires the computation of $A^T b$ (at a cost of nm flops) and two triangular linear systems (at a cost of n^2 flops). The cost of determining the $\mathcal{QR}$ factorization of A is $n^2 m + O(nm)$ and therefore the total cost of this algorithm to determine x is $n^2 m + O(nm)$ flops.

Note that in most applications m is much larger than n. With this approach we have computed the LU (or Cholesky) decomposition of $A^T A$ (= $R^T R$) without forming $A^T A$.

Rank Deficient Problems

Note that deficient rank implies that for overdetermined problems
$rank(A) = rank(A^T A) < n$,
while for underdetermined problems
$rank(A) = rank(AA^T) < m$.
In either case the $\mathcal{R}$ we obtain from the QR algorithm will have the same deficient rank.

In exact arithmetic the rank of a triangular matrix is the number of non-zeros on the diagonal. In floating point arithmetic we don't expect to see an exact zero but a reliable indication of rank deficiency is to observe a large value for the ratio of the largest to smallest magnitudes of the diagonal entries.

In problems where rank deficiency is detected (using this idea) the algorithm can either exit with a warning or attempt to produce a solution to a nearby exactly rank deficient problem.