Solving Least-Squares Problems

In [24]:

```
#keep
import numpy as np
import numpy.linalg as la
import scipy.linalg as spla
```

In [25]:

```
#keep
m = 6
n = 4

A = np.random.randn(m, n)
b = np.random.randn(m)
```

Let's try solving that as a linear system using `la.solve`:

In [26]:

```
la.solve(A, b)
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent cal
l last)
<ipython-input-26-32000b03e56a> in <module>()
----> 1 la.solve(A, b)

/usr/lib/python3/dist-packages/numpy/linalg/linalg.py in solve(a, b)
    353     a, _ = _makearray(a)
    354     _assertRankAtLeast2(a)
--> 355     _assertNdSquareness(a)
    356     b, wrap = _makearray(b)
    357     t, result_t = _commonType(a, b)

/usr/lib/python3/dist-packages/numpy/linalg/linalg.py in _assertNdSq
uareness(*arrays)
    210     for a in arrays:
    211         if max(a.shape[-2:]) != min(a.shape[-2:]):
--> 212             raise LinAlgError('Last 2 dimensions of the arra
y must be square')
    213 
    214 def _assertFinite(*arrays):

LinAlgError: Last 2 dimensions of the array must be square
```

OK, let's do QR-based least-squares then.

In [27]:

```
Q, R = la.qr(A)
```

What did we get? Full QR or reduced QR?

In [28]:

```
#keep
Q.shape
```

Out[28]:

```
(6, 4)
```

In [29]:

```
#keep
R.shape
```

Out[29]:

```
(4, 4)
```

Is that a problem?

- Do we really need the bottom part of R? (A bunch of zeros)
- Do we really need the far right part of Q? (=the bottom part of Q^T)

OK, so find the minimizing x:

In [39]:

```
x = spla.solve_triangular(R, Q.T.dot(b), lower=False)
```

We predicted that $\|Ax - b\|_2$ would be the same as $\|Rx - Q^T b\|_2$:

In [45]:

```
la.norm(A.dot(x)-b, 2)
```

Out[45]:

```
1.4448079009090737
```

In [47]:

```
la.norm(R.dot(x) - Q.T.dot(b))
```

Out[47]:

```
1.5700924586837752e-16
```

Heh--*reduced* QR left out the right half of Q. Let's try again with complete QR:

In [59]:

```
#keep
Q2, R2 = la.qr(A, mode="complete")
```

In [60]:

```
#keep
x2 = spla.solve_triangular(R[:n], Q.T[:n].dot(b), lower=False)
```

In [63]:

```
#keep
la.norm(A.dot(x)-b, 2)
```

Out[63]:

```
1.4448079009090737
```

In [64]:

```
#keep
la.norm(R2.dot(x2) - Q2.T.dot(b))
```

Out[64]:

```
1.444807900909074
```

Did we get the same `x` both times?

In [69]:

```
x - x2
```

Out[69]:

```
array([ 0.,  0.,  0.,  0.])
```

Finally, let's compare against the normal equations:

In [70]:

```
#keep
x3 = la.solve(A.T.dot(A), A.T.dot(b))
```

In [71]:

```
x3 - x
```

Out[71]:

```
array([  4.99600361e-16,  -1.66533454e-16,   7.77156117e-16,
        -1.11022302e-16])
```

In [ ]:

```

```