Least Squares using the SVD

In [1]:

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

In [2]:

```
#keep
# tall and skinny w/nullspace
np.random.seed(12)
A = np.random.randn(6, 4)
b = np.random.randn(6)
A[3] = A[4] + A[5]
A[1] = A[5] + A[1]
A[2] = A[3] + A[1]
A[0] = A[3] + A[1]
```

Part I: Singular least squares using QR

Let's see how successfully we can solve the least squares problem **when the matrix has a nullspace** using QR:

In [3]:

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

In [4]:

```
#keep
R.round(3)
```

Out[4]:

```
array([[-4.526,  3.492, -0.204, -3.647],
       [ 0.   ,  0.796,  0.034,  0.603],
       [ 0.   ,  0.   , -1.459,  0.674],
       [ 0.   ,  0.   ,  0.   ,  0.   ]])
```

We can choose `x_qr[3]` as we please:

In [5]:

```
#keep
x_qr = np.zeros(A.shape[1])
```

In [6]:

```
x_qr[3] = 0
```

In [7]:

```
#keep
QTbnew = Q.T.dot(b)[:3,] - R[:3, 3] * x_qr[3]
x_qr[:3] = spla.solve_triangular(R[:3,:3], QTbnew, lower=False)
```

Let's take a look at the residual norm and the norm of `x_qr`:

In [8]:

```
#keep
R.dot(x_qr)-Q.T.dot(b)[:4]
```

Out[8]:

```
array([ -4.44089210e-16,   0.00000000e+00,   0.00000000e+00,
        -1.97736227e-01])
```

In [9]:

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

Out[9]:

```
2.1267152888030982
```

In [10]:

```
#keep
la.norm(x_qr, 2)
```

Out[10]:

```
0.82393512974131566
```

Choose a different `x_qr[3]` and compare residual and norm of `x_qr`.

Part II: Solving least squares using the SVD

Now compute the SVD of A:

In [11]:

```
U, sigma, VT = la.svd(A)
```

Make a matrix `Sigma` of the correct size:

In [12]:

```
#keep
Sigma = np.zeros(A.shape)
Sigma[:4,:4] = np.diag(sigma)
```

And check that we've actually factorized `A`:

In [13]:

```
#keep
(U.dot(Sigma).dot(VT) - A).round(4)
```

Out[13]:

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

Now define `Sigma_pinv` as the "pseudo-"inverse of `Sigma`, where "pseudo" means "don't divide by zero":

In [14]:

```
Sigma_pinv = np.zeros(A.shape).T
Sigma_pinv[:3,:3] = np.diag(1/sigma[:3])
Sigma_pinv.round(3)
```

Out[14]:

```
array([[ 0.147,  0.   ,  0.   ,  0.   ,  0.   ,  0.   ],
       [ 0.   ,  0.624,  0.   ,  0.   ,  0.   ,  0.   ],
       [ 0.   ,  0.   ,  1.055,  0.   ,  0.   ,  0.   ],
       [ 0.   ,  0.   ,  0.   ,  0.   ,  0.   ,  0.   ]])
```

Now compute the SVD-based solution for the least-squares problem:

In [15]:

```
x_svd = VT.T.dot(Sigma_pinv).dot(U.T).dot(b)
```

In [16]:

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

Out[16]:

```
2.1267152888030978
```

In [17]:

```
la.norm(x_svd)
```

Out[17]:

```
0.77354943014895816
```

- What do you observe about $\|\text{x\_svd}\|_2$ compared to $\|\text{x\_qr}\|_2$?
- Is $\|\text{x\_svd}\|_2$ compared to $\|\text{x\_qr}\|_2$?

In [ ]:

```

```