Keeping track of coefficients in Gram-Schmidt

In [1]:

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

In [2]:

```
#keep
A = np.random.randn(3, 3)
```

Let's start from regular old (modified) Gram-Schmidt:

In [3]:

```
#keep

Q = np.zeros(A.shape)

q = A[:, 0]
Q[:, 0] = q/la.norm(q)

# -----------

q = A[:, 1]
coeff = np.dot(Q[:, 0], q)
q = q - coeff*Q[:, 0]
Q[:, 1] = q/la.norm(q)

# -----------

q = A[:, 2]
coeff = np.dot(Q[:, 0], q)
q = q - coeff*Q[:, 0]
coeff = np.dot(Q[:, 1], q)
q = q - coeff*Q[:, 1]
Q[:, 2] = q/la.norm(q)
```

In [4]:

```
#keep
Q.dot(Q.T)
```

Out[4]:

```
array([[  1.00000000e+00,  -2.77555756e-17,  -5.55111512e-17],
       [ -2.77555756e-17,   1.00000000e+00,  -5.55111512e-17],
       [ -5.55111512e-17,  -5.55111512e-17,   1.00000000e+00]])
```

Now we want to keep track of what vector got added to what other vector, in the style of an elimination matrix.

Let's call that matrix R.

- Would it be $A = QR$ or $A = RQ$? Why?
- Where are R's nonzeros?

In [5]:

```
R = np.zeros((A.shape[0], A.shape[0]))
```

In [6]:

```
Q = np.zeros(A.shape)

q = A[:, 0]
Q[:, 0] = q/la.norm(q)

R[0,0] = la.norm(q)

# -----------

q = A[:, 1]
coeff = np.dot(Q[:, 0], q)
R[0,1] = coeff
q = q - coeff*Q[:, 0]
Q[:, 1] = q/la.norm(q)

R[1,1] = la.norm(q)

# -----------

q = A[:, 2]
coeff = np.dot(Q[:, 0], q)
R[0,2] = coeff
q = q - coeff*Q[:, 0]
coeff = np.dot(Q[:, 1], q)
R[1,2] = coeff
q = q- coeff*Q[:, 1]
Q[:, 2] = q/la.norm(q)

R[2,2] = la.norm(q)
```

In [7]:

```
R
```

Out[7]:

```
array([[ 0.81919017, -1.17431844, -1.02300235],
       [ 0.        ,  1.11510507,  0.82496372],
       [ 0.        ,  0.        ,  1.98359192]])
```

In [8]:

```
la.norm(Q.dot(R) - A)
```

Out[8]:

```
2.026584714835721e-16
```

This is called QR factorization (https://en.wikipedia.org/wiki/QR_decomposition).

- When does it break?
- Does it need something like pivoting?
- Can we use it for something?

In [ ]:

```

```