Load .mat file and extract array into X

In [1]:
```
import scipy.io
import numpy as np
import time

data = scipy.io.loadmat('data.mat')

X = data['X']
```

Sanity check for X:

In [2]:
```
[nRow,nCol] = np.shape(X)
print(nRow, nCol)
```

```
10 1000000
```

Method 1: Sum up ℓ_2 column norm by explicit for loop accumulator

In [3]:
```
t0 = time.time()
S = 0
for i in range(nCol):
    S += np.linalg.norm(X[:,i])
t1 = time.time()
print("time taken for loop: ", t1-t0, "  value: ", S)
```

```
time taken for loop:  10.143542051315308   value:  1380518.3753343287
```

Method 2: Use built in function to compute column-wise ℓ_2 for entire matrix X, then np.sum to accumulate

In [4]:
```
t0 = time.time()
A = np.sum(np.linalg.norm(X,axis=0))
t1 = time.time()
print("time taken built-in: ", t1-t0, "  value: ", A)
```

```
time taken built-in:  0.15878701210021973   value:  1380518.375334337
```

Summary

On average, the sum function version performed around 50-120 times faster than the for loop.

We see from this lab that it is always better to use built-in functions whenever applicable. This is because sum is a function so commonly used so that python provides an optimized version of it implemented in C, and C is a compiled language much faster than an interpreted language such as matlab. Also there should be some optimization involved by computing the ℓ_2 norm of columns of X all at once instead of individually.