Customer purchase history

We are now interested in efficient computations. In our setting, note that the data matrix C is large but very sparse. The number of non zero-valued elements divided by the total number of elements is called the density d of the matrix C. Let w in $\mathbb{R}^K$ be a given weighting vector. Assume that we center the rows (removing the average row to every row), obtaining a new row-centered matrix C_m.

In [1]:
```
import numpy as np
import time
from scipy.sparse import csr_matrix
```

In [2]:
```
lmbda = 0.1
K = 10000
N = 10000
dims = [N, K]
```

Generate sparse matrix using poisson distribution

In [3]:
```
# in dense matrix format, no performance improvement
C = np.random.poisson(lmbda, dims)

# in sparse matrix format, certain operations should be faster
C_sparse = csr_matrix(C)

# average of the rows of C
r_avg = np.mean(C, axis=0)

# w, in this case we use the one vector
w = np.ones(K)
```

In [4]:
```
print("The sparsity is around", C_sparse.count_nonzero() / N / K)
```

```
The sparsity is around 0.09521792
```

Naive implementation:

In [5]:
```
# The jupyter-notebook's magic commands, %t expr,
# will print the amount of time needed to evaluate expr
%time (C - r_avg) @ w
```

```
CPU times: user 682 ms, sys: 1.04 s, total: 1.73 s
Wall time: 5.22 s
```

Out[5]:
```
array([-28.5493,  -3.5493,  25.4507, ...,  48.4507, -21.5493,  -5.5493])
```

Efficient implementation:

In [6]:
```
# TODO: Implement your proposed procedure here to compute the desired quantity.
# Make sure you always get the same results as the naive implementation
%time C_sparse @ w - C_sparse.mean(axis=0) @ w
```

```
CPU times: user 83.2 ms, sys: 105 ms, total: 188 ms
Wall time: 196 ms
```

Out[6]:
```
matrix([[-28.5493,  -3.5493,  25.4507, ...,  48.4507, -21.5493,  -5.5493]])
```

In [ ]: