CS 598 APK, Fall 2015

Improving Randomized SVD

Dec 19, 2015

Daeyun Shin

In this report, we discuss randomized methods for low-rank approximation and SVD.

Adaptive range finder

Using [Halko et al. http://arxiv.org/pdf/0909.4061v2.pdf], we compute a low rank approximation of an m by n matrix A by finding the orthonormal basis Q of $Y = A\Omega_{m\times k}$ where Ω is a randomly drawn Gaussian matrix, and Q such that $QR = Y$. k is an estimate of A's rank in the low rank representation. The *adaptive* part of adaptive range finding is computing Q when we don't know k beforehand and want to find one such that the a-posteriori error $|(I-QQ^T)A|_2$ is small enough. We can start with an initial estimate and incrementally add randomly drawn vectors to Ω. The time complexity of the randomized method is $O(mk^2)$ We can compare this with a baseline implementation which uses $O(mn^2)$ rank-revealing QR.

Incremental orthonormalization

For incrementally finding an orthonormal basis Q, we have two implementations.

Modified Gram Schmidt

We can modify Gram Schmidt to assume the first k columns are already orthonormal and only adjust columns $k+1, ..., k+p$. We normalize the column in each step for floating point stability. While this makes sense, it will be slow in practice if it doesn't take advantage of low level vectorization.

Running time with a rank 20 $A_{500\times 500}$, starting with $k=5$ columns and adding $p=5$ in each round,

```
5 loops, best of 5: 9.75 ms per loop
```

Incremental batch orthonormalization using orthogonal complement projection

This method allows us to batch orthonormalize incrementally, taking advantage of low level QR factorization routines by projecting $Y = A\Omega_{m\times p}$ to $Q^{\perp}$, i.e. $Q_i R_i = (I-QQ^T)Y$ i.e. $Y - Q(Q^T Y)$ using matrix associativity. Columns of $(I-QQ^T)Y$ will be orthogonal to Q but not orthogonal themselves which we take take of by computing QR. In case this is ill conditioned, we re-project the resulting Q_i to $Q^{\perp}$ and compute QR i.e. $Q_i R_i = Q_i - Q(Q^T Q_i)$. If R_i has any zero diagonal entries, remove the corresponding columns from Q_i . This happens when adding Q_i is an overestimation. Then we concatenate $Q = [Q\ Q_i]$ to obtain the new basis. See randomized_svd.py

In each iteration, both implementations approximate the a-posteriori error $|(I-QQ^T)A|_2 \approx |(I-QQ^T)A\omega|_2$. Note that $A - Q(Q^T A)$ doesn't have to be computed from scratch because only the last p columns of Q changed.

```
5 loops, best of 5: 5.96 ms per loop
```

RRQR

```
5 loops, best of 5: 20.1 ms per loop
```

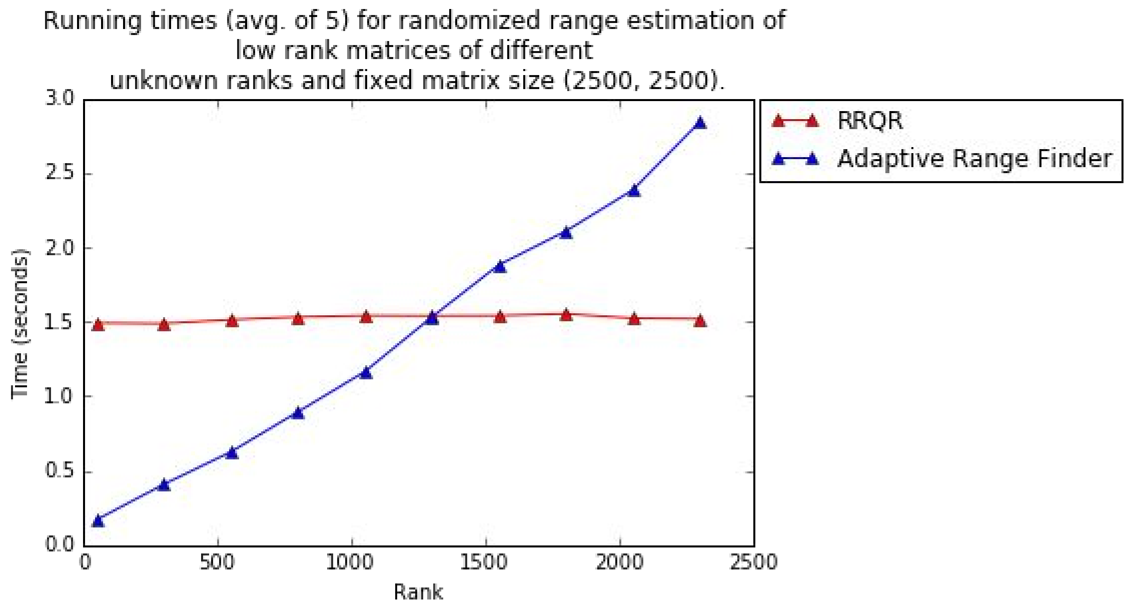
The naive modified Gram Schmidt implementation in Python doesn't scale well and takes longer than RRQR when A is large.

Using rank 500, $A_{2000\times 2000}$, $k = 50$, $p = 50$:

```
MGS: 5 loops, best of 5: 3.37 s per loop
Incremental QR: 5 loops, best of 5: 284 ms per loop
RRQR: 5 loops, best of 5: 576 ms per loop
```

Incremental QR vs RRQR

using $A_{2500\times 2500}$ with varying rank, i.e. A = $A_{2500\times k}A_{k\times 2500}$



Because of the overhead in the incremental search, it could be faster to switch to RRQR when $k + pq$ gets too big. In practice, with tall-and-skinny matrices, this doesn't seem to be a big deal. Example using $A_{30000\times 784}$ and Q with ~ 700 columns:

```
A=mnist.train.images[:30000,:]
```

```
%timeit Q = randomized_svd.adaptive_range(A, eps=1e-4, k=10, p=10)
```

```
1 loops, best of 3: 14.7 s per loop
```

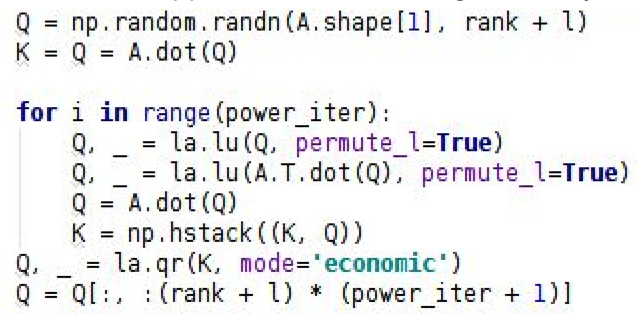
```
%timeit Q = rank_revealing_QR(A)
```

```
1 loops, best of 3: 24.1 s per loop
```

Randomized SVD using block Krylov iteration, interpolative decomposition, power iterations, and comparison with third party libraries.

Using the randomized block Krylov method [Musco et al. http://arxiv.org/pdf/1504.05477.pdf], we experiment with randomized SVD for small datasets. Another method we compare with is interpolative decomposition. We use a subset of MNIST and CIFAR-10 (first few thousand rows) for evaluation. Here, adaptive range finding is no longer used, we do fixed low rank approximation. The size of the orthonormal basis Q is on the x axis in all following figures. We experiment with varying number of power iterations, oversampling, and also using third party baselines (without Block Krylov iterations implemented).

Here is a snippet of the normalizing block Krylov method in Python.



```
Q = np.random.randn(A.shape[1], rank + l)
K = Q = A.dot(Q)

for i in range(power_iter):
    Q, _ = la.lu(Q, permute_l=True)
    Q, _ = la.lu(A.T.dot(Q), permute_l=True)
    Q = A.dot(Q)
    K = np.hstack((K, Q))
Q, _ = la.qr(K, mode='economic')
Q = Q[:, :(rank + l) * (power_iter + 1)]
```

Reference for the LU normalization method: [Szlam et al. http://arxiv.org/pdf/1412.3510v1.pdf]

Using 4000 rows of MNIST:
A.shape = (4000, 784)
Actual rank of this matrix according to NumPy's matrix_rank: 609

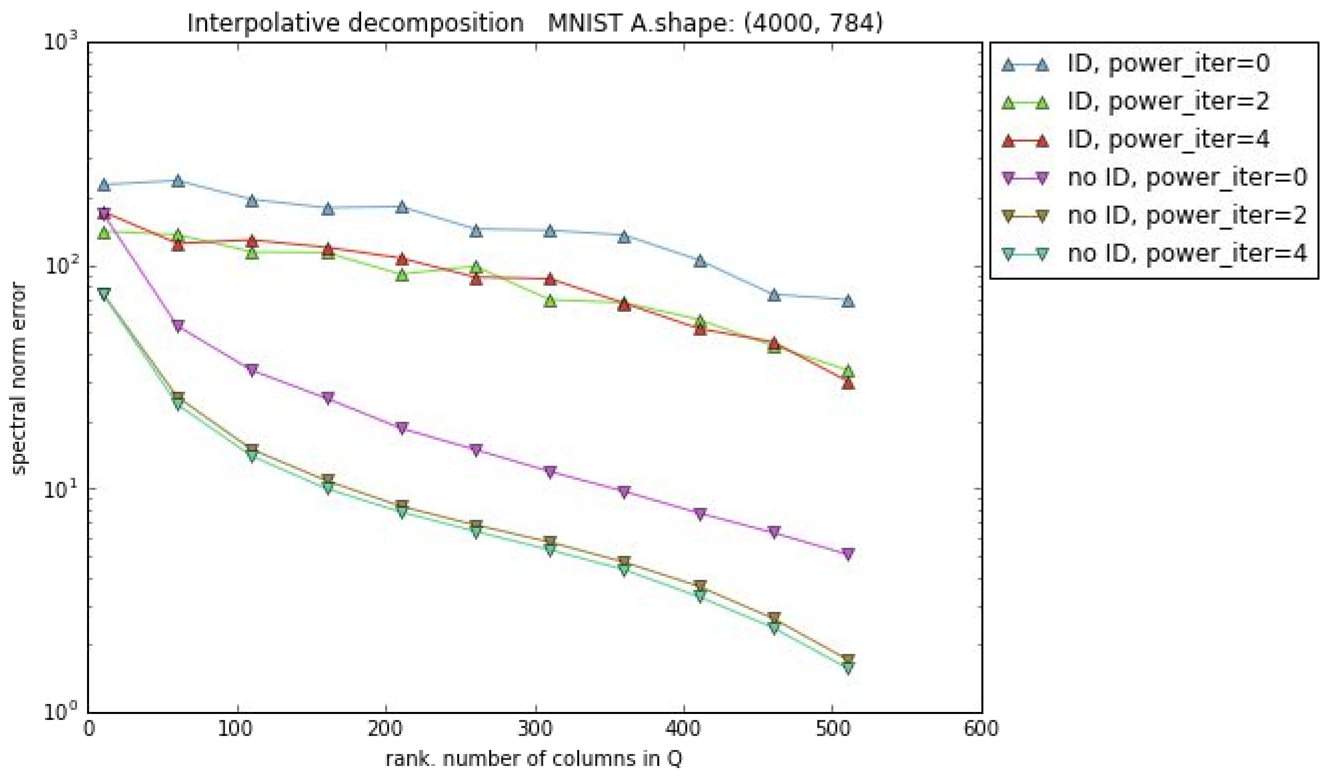
For CIFAR-10, we use A.shape=(5000,3072), matrix_rank(A)=3072

Error metric: $\left|A - U\Sigma V^T\right|_2$

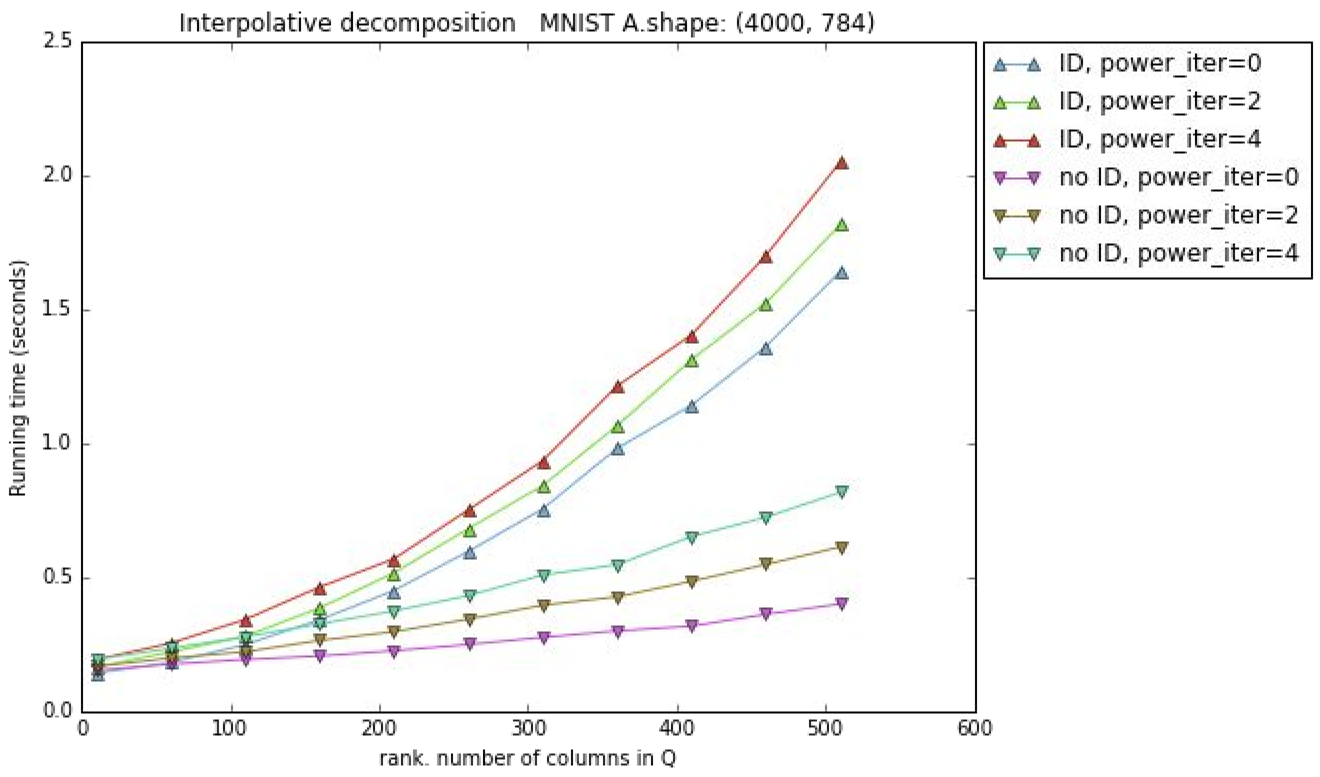
Interpolative decomposition:

It appears that the interpolative decomposition method doesn't perform as well as $svd(Q^T A)$. Scikit-learn and fbpca aren't using ID either.

Spectral norm error:
(mean of 2 rounds)



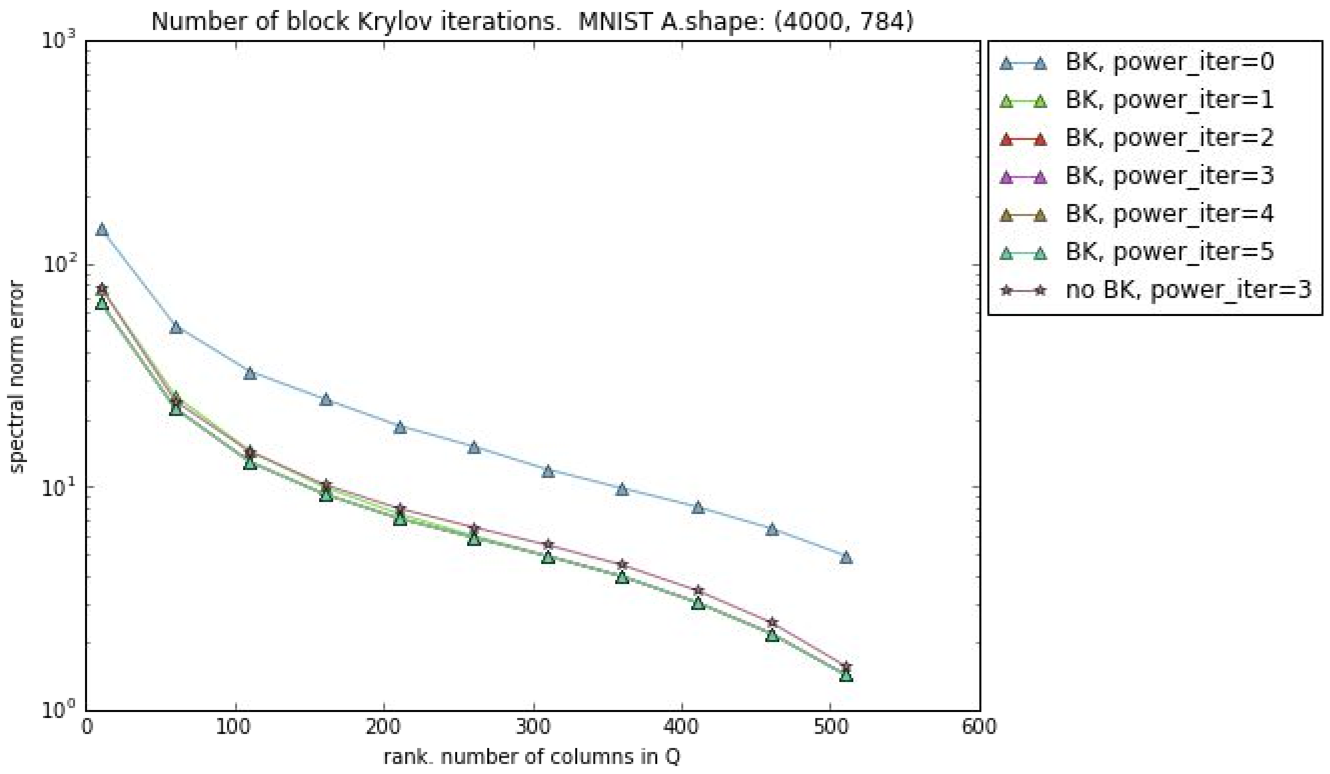
Running time (seconds):



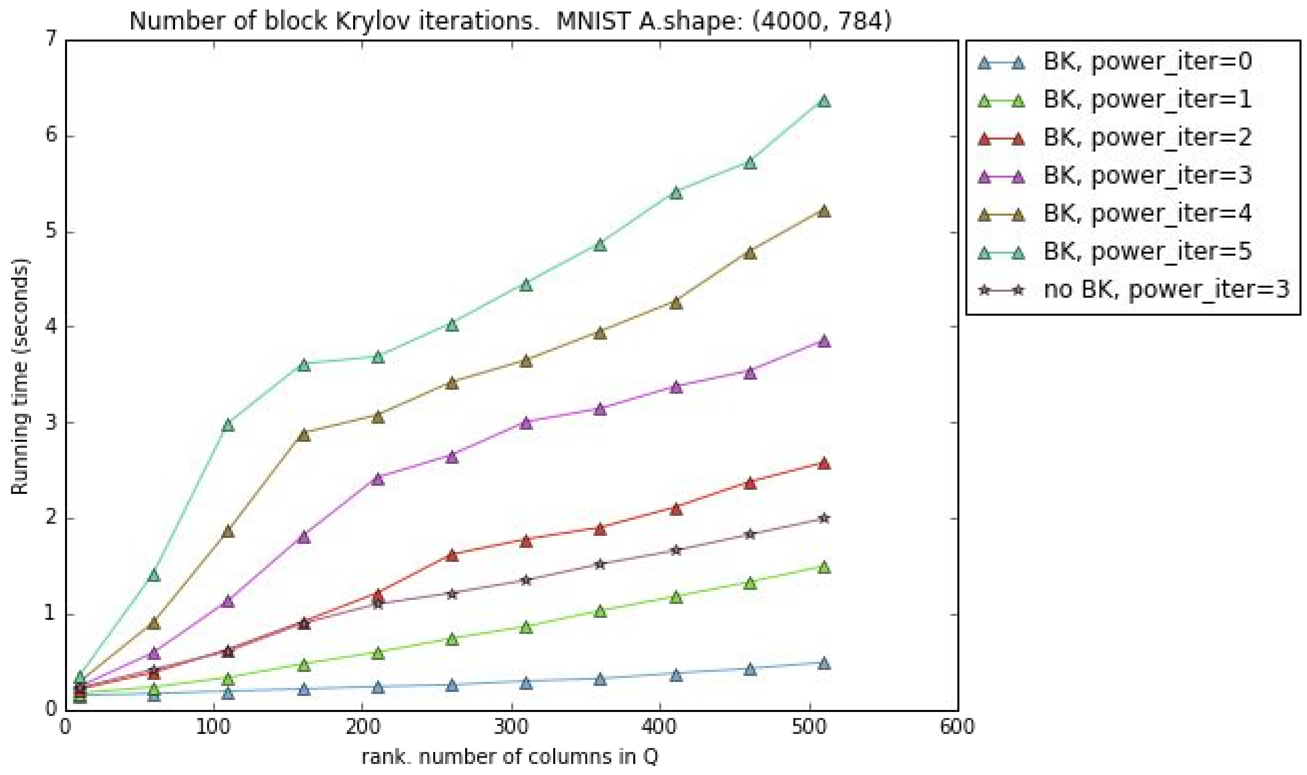
Number of block Krylov iterations:

Zero iterations (same as just $QR = A\Omega$) performs the worst, as expected, followed by 3 (simultaneous) power iterations, and both are outperformed when there's at least one block Krylov iteration. It appears that it is enough to run one or two iterations.

Spectral norm error:



Running time (seconds):
Linear as expected

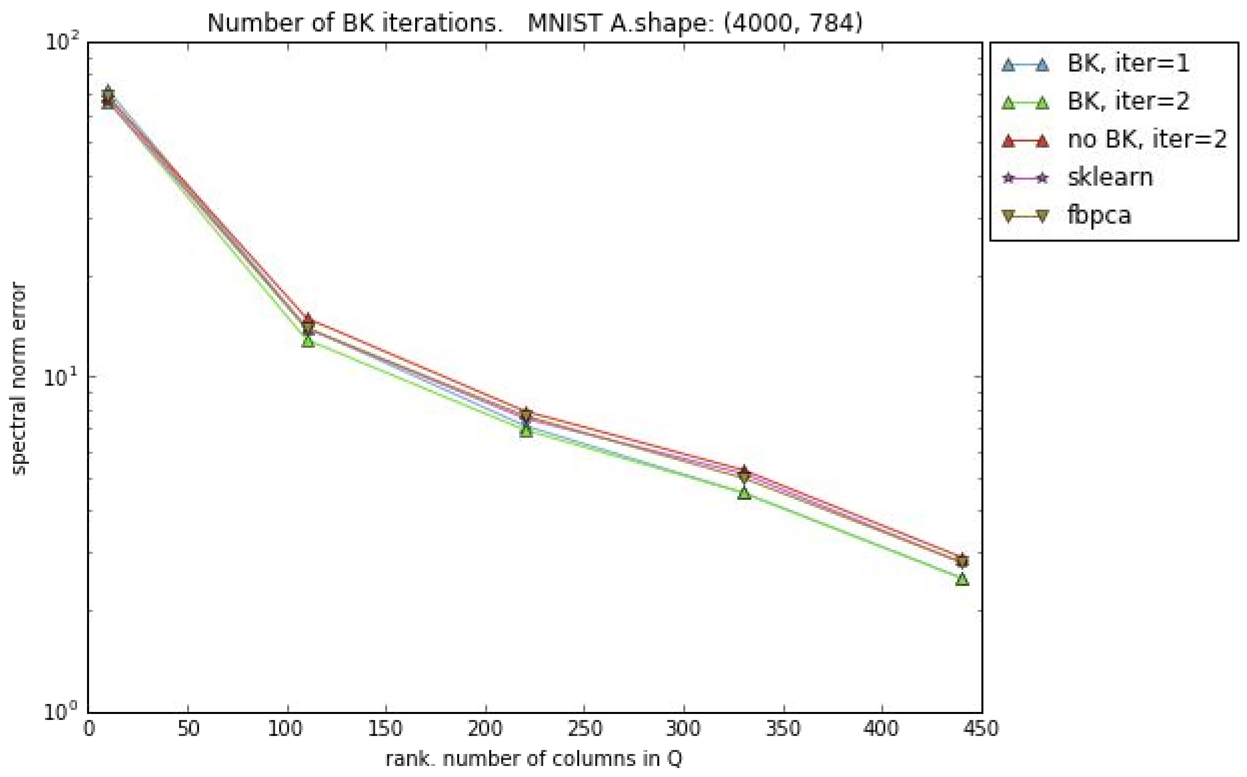


Third-party randomized SVD

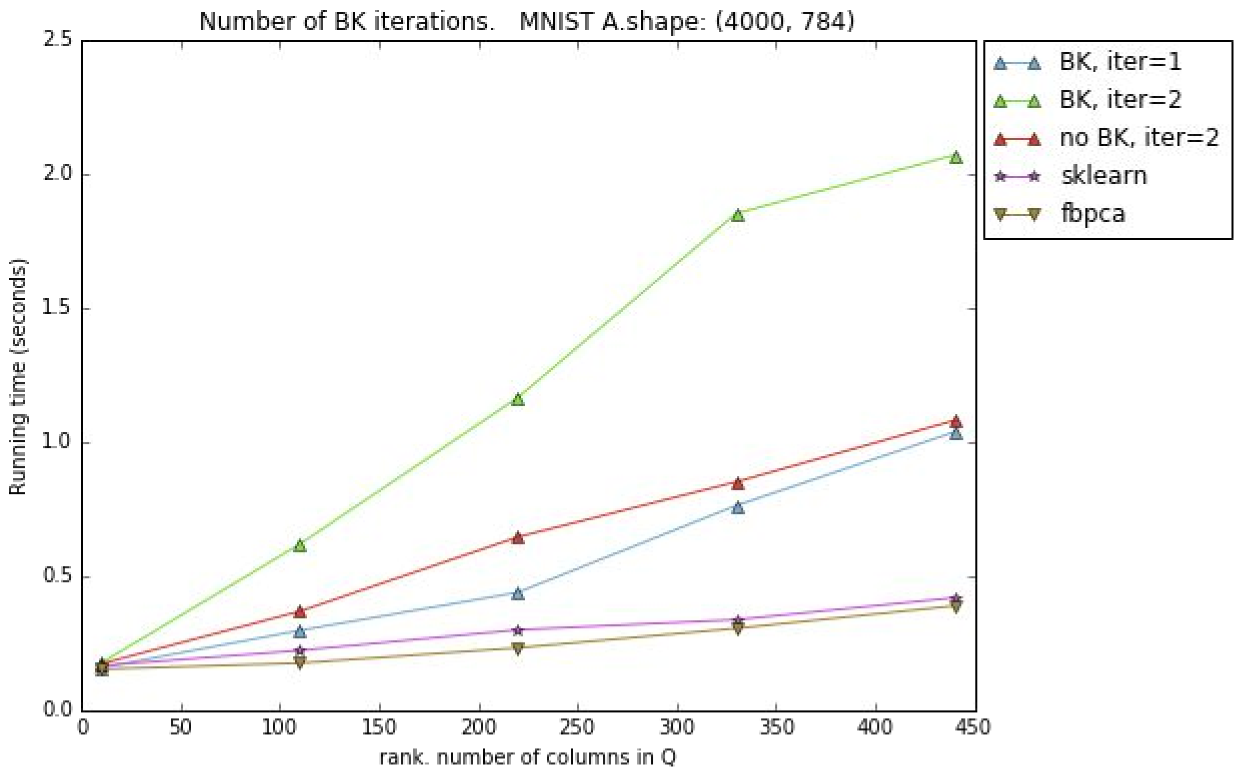
Randomized SVD variants using Halko et al.'s method perform roughly the same in ours, scikit-learn, and fbpca. We compare them with Block Krylov method in our implementation.

Using MNIST:
We keep the number of oversampling and power iterations fixed in all implementations rather than using default values.

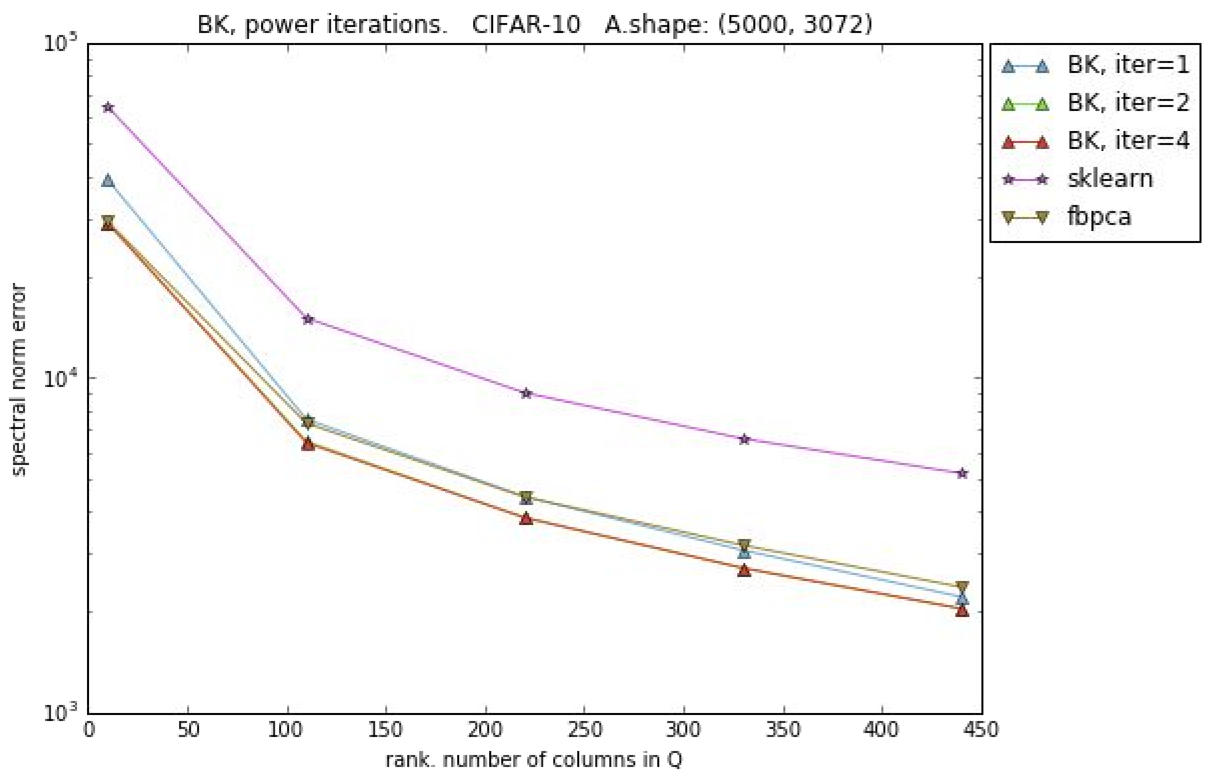
Spectral norm error:

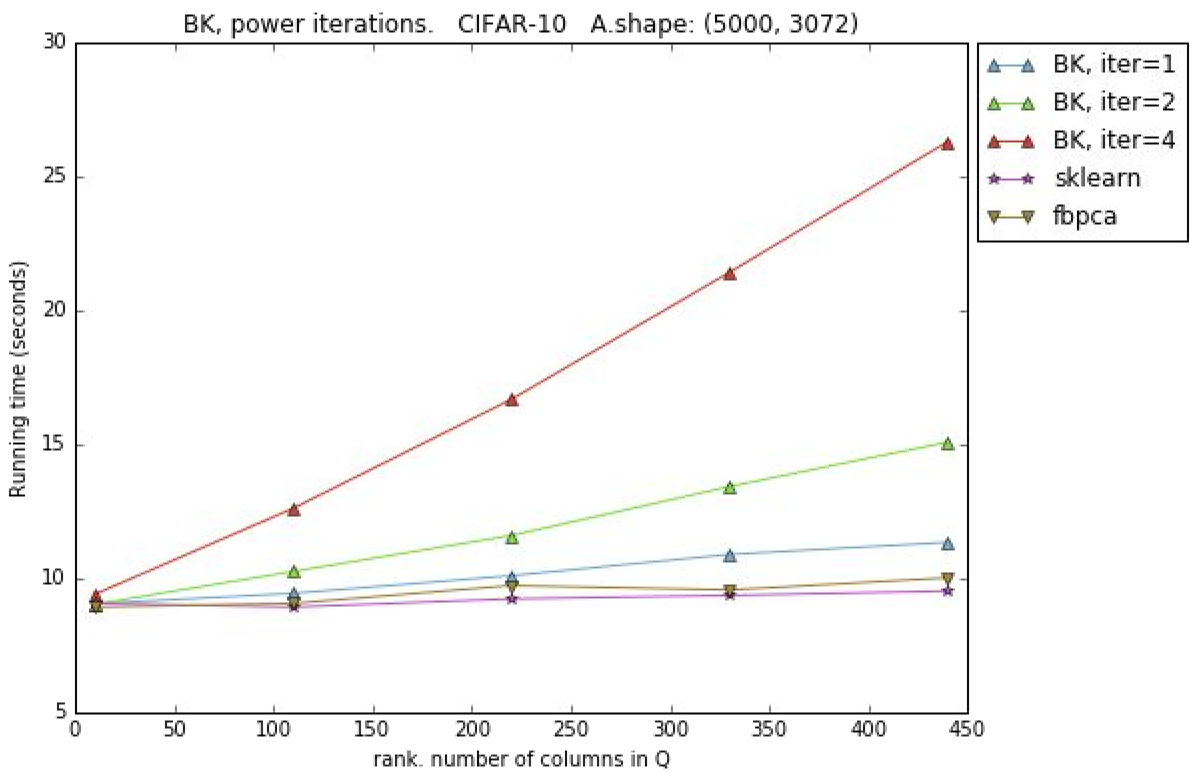


Running time:



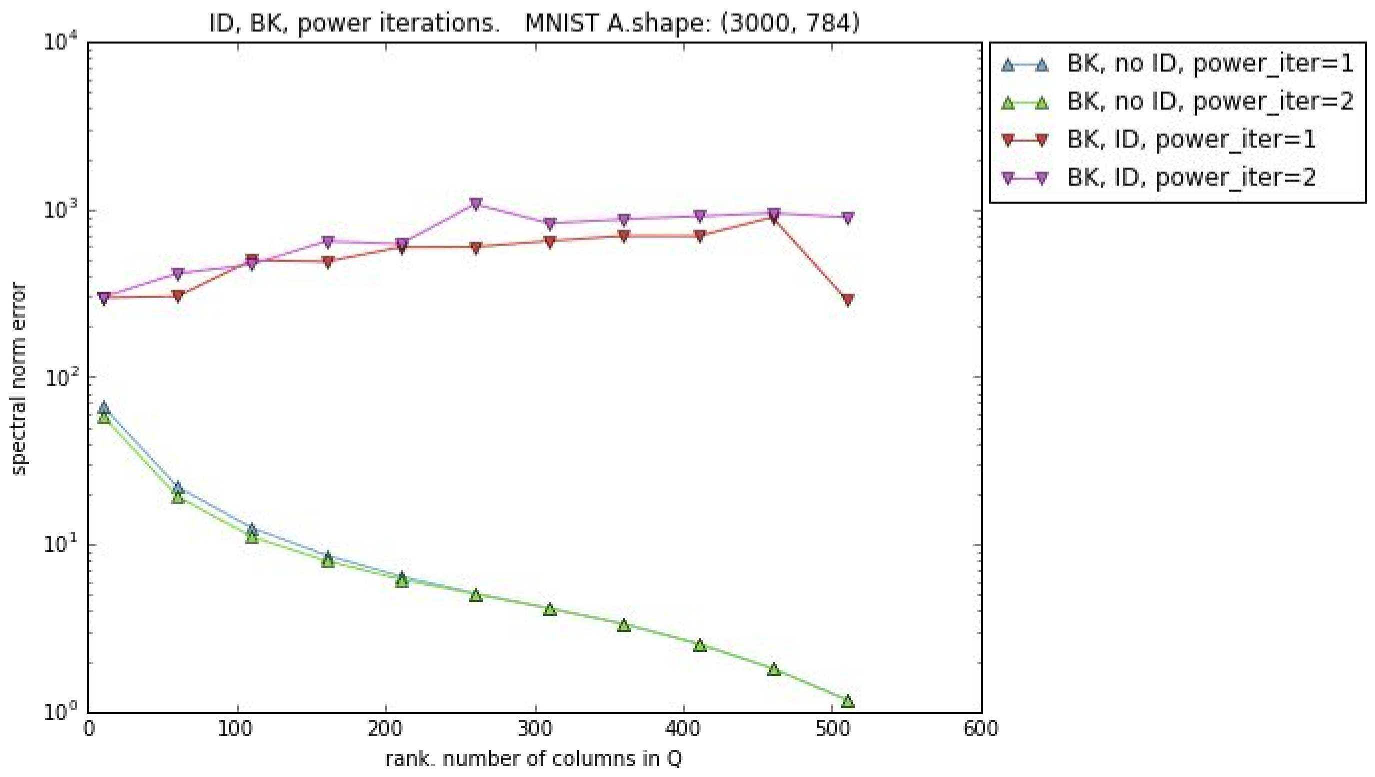
Using 5000 rows of CIFAR-10:
Here, we use the default values in each library.

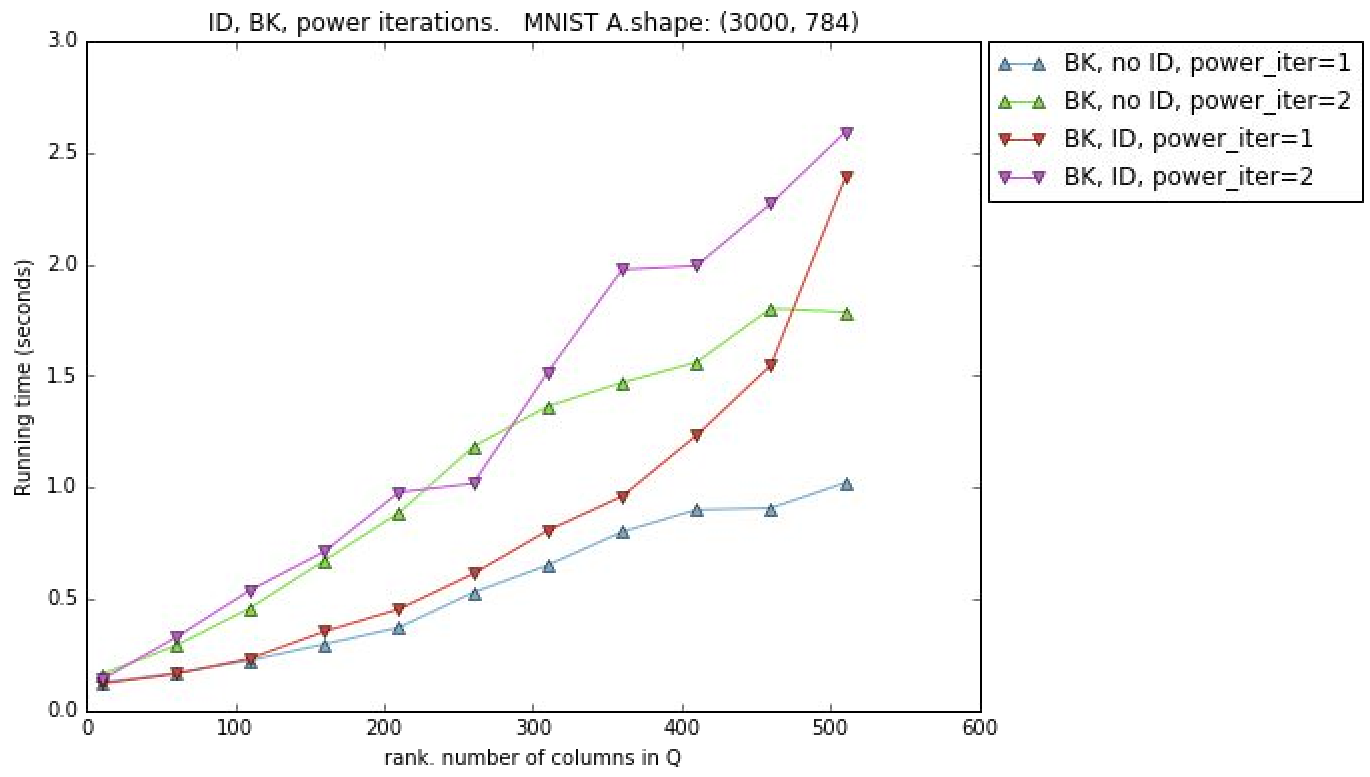




Block Krylov method with interpolative decomposition

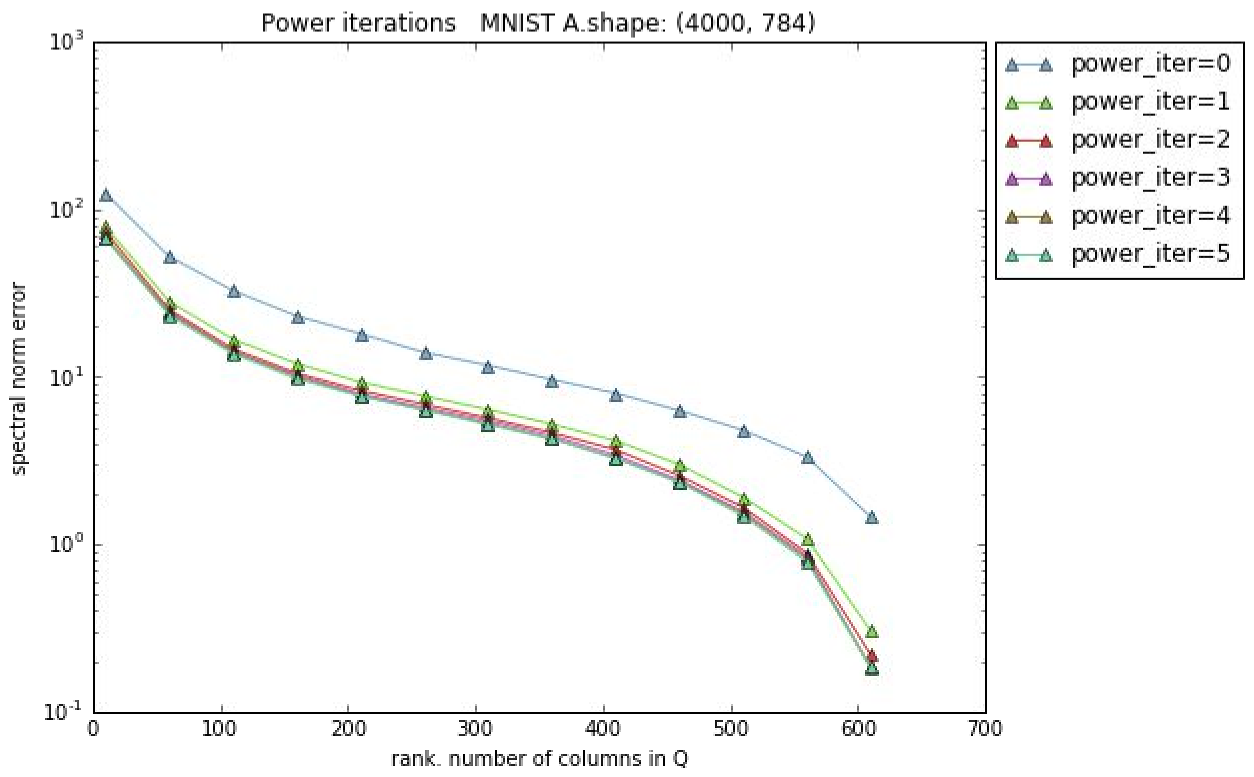
Using ID on the output of block krylov iterations (Q.shape[1]=q*k) doesn't seem to help.





Power iterations without Block Krylov method:

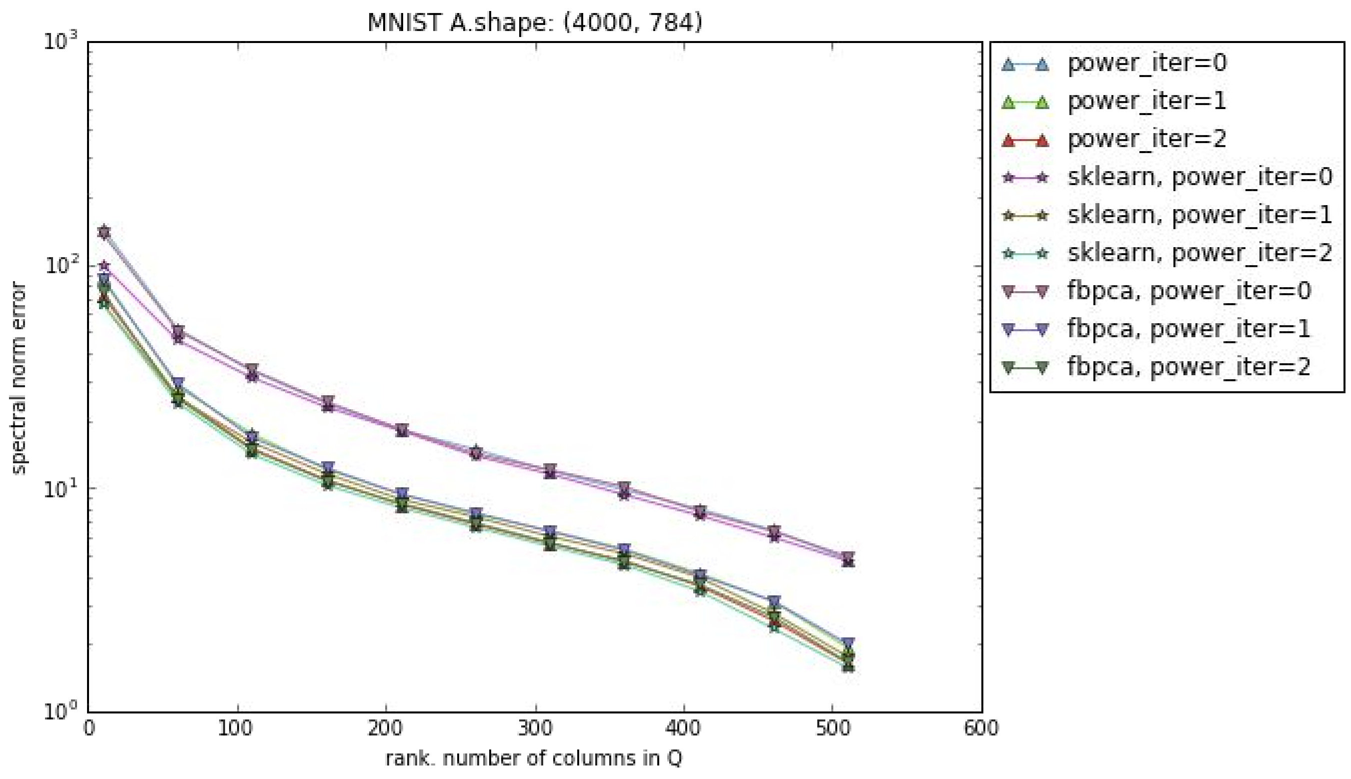
Two rounds of best of 2



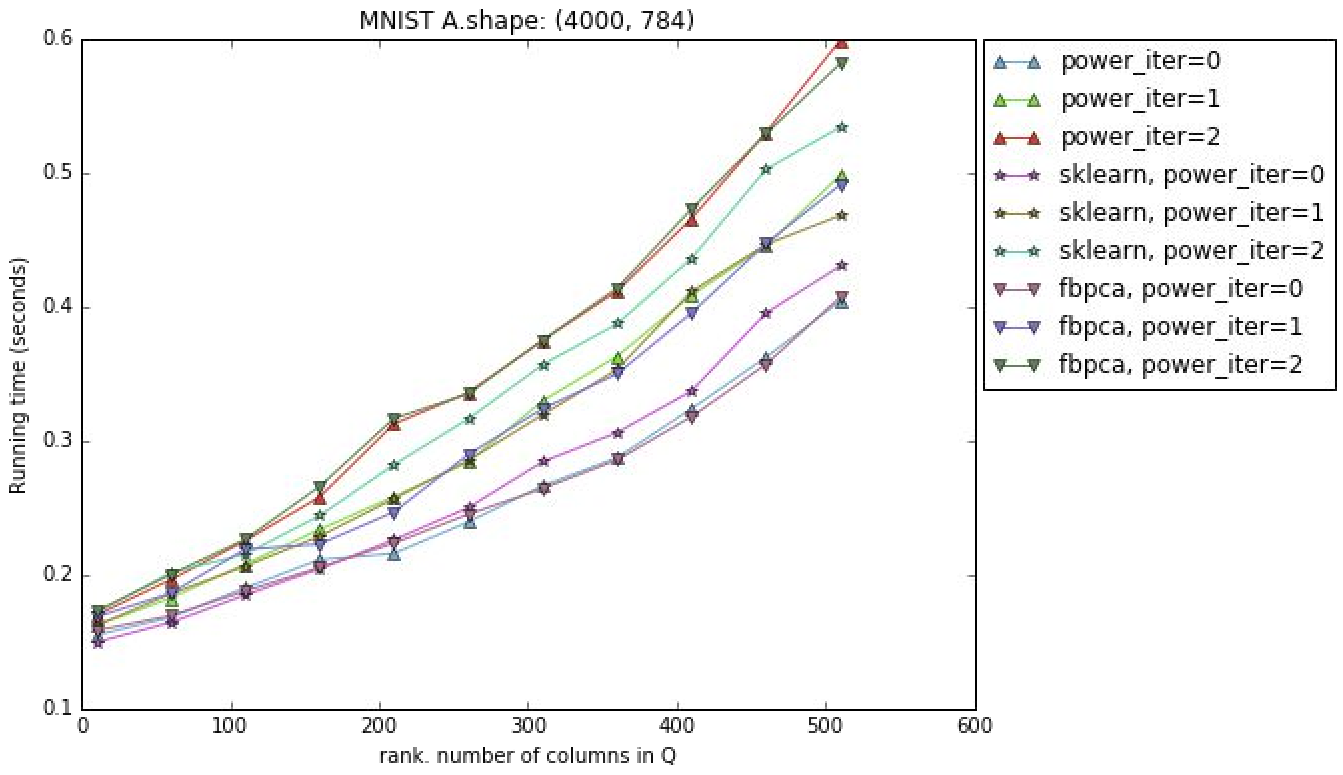
vs. other implementations:

Similar behavior in all three implementations (the top most three lines are power_iter=0 from each).

Spectral norm error:

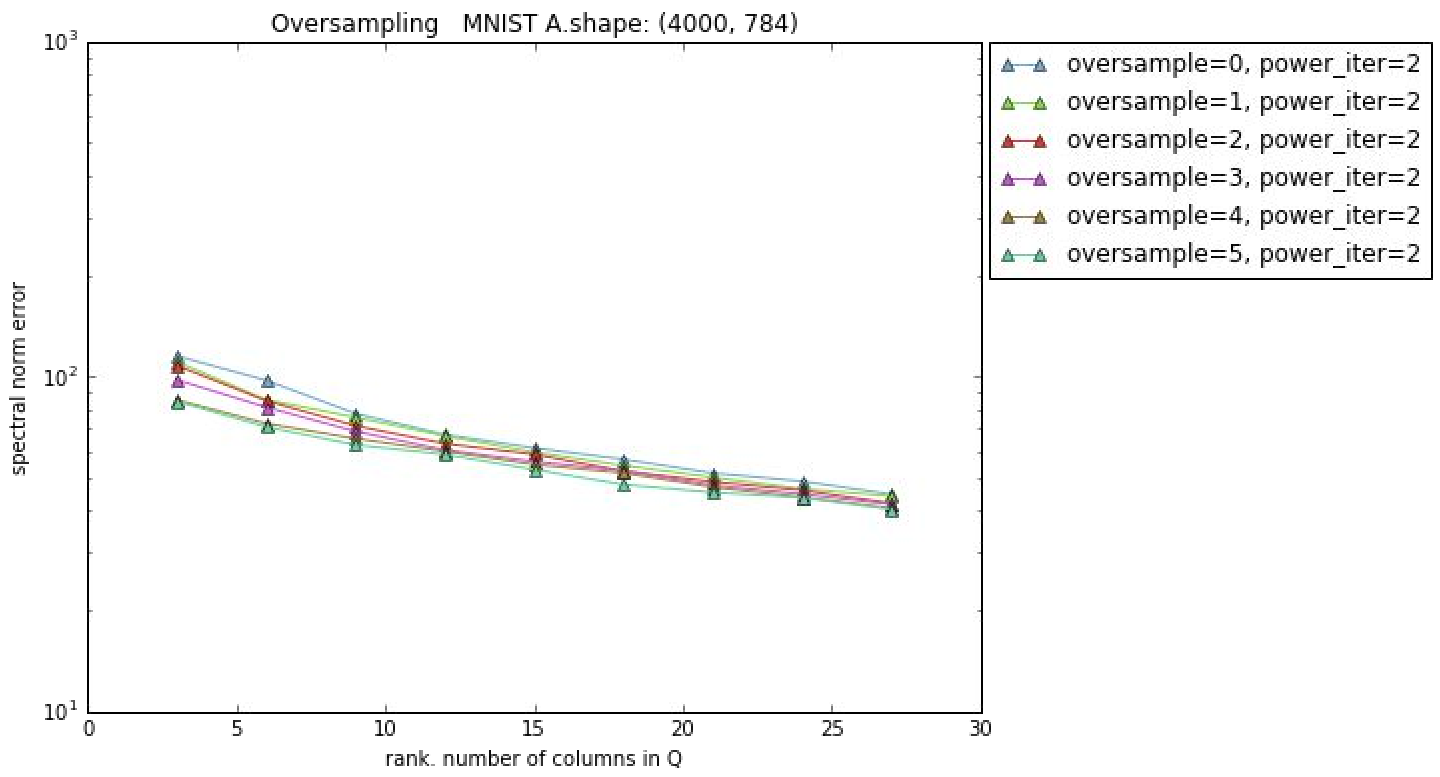


Running time:



We also experiment with oversampling. i.e. computing SVD using a rank $k+l$ orthonormal basis then keeping the first k components. Only helps with low k, as expected.

Spectral norm (best of 2, 4 rounds):


Oversampling   MNIST A.shape: (4000, 784)
10^3
10^2
10^1
spectral norm error
0
5
10
15
20
25
30
rank. number of columns in Q
oversample=0, power_iter=2
oversample=1, power_iter=2
oversample=2, power_iter=2
oversample=3, power_iter=2
oversample=4, power_iter=2
oversample=5, power_iter=2