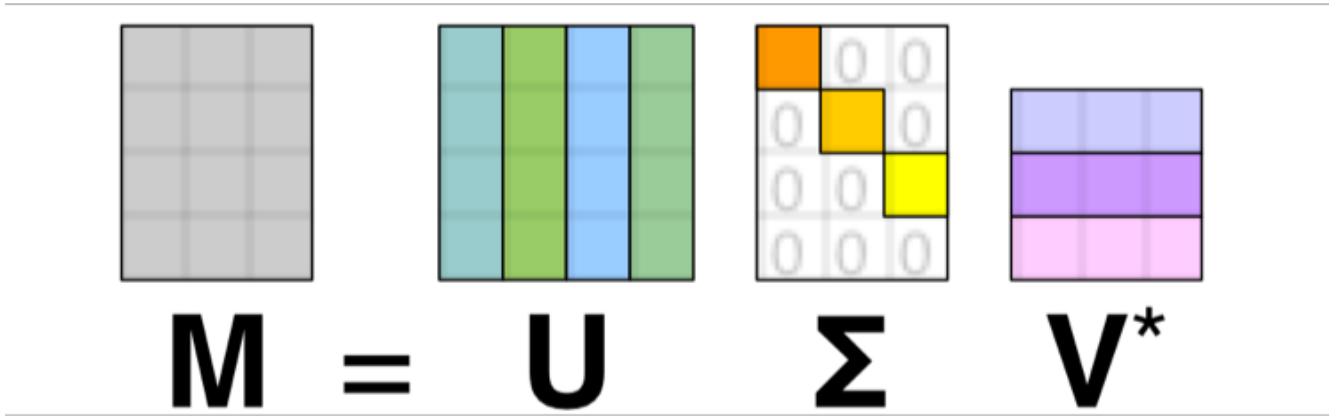
(fast) Randomized SVD



Ryan Levy, Algorithm Interest Group, Jan. 31 2019

Image: Wikipedia

Roadmap

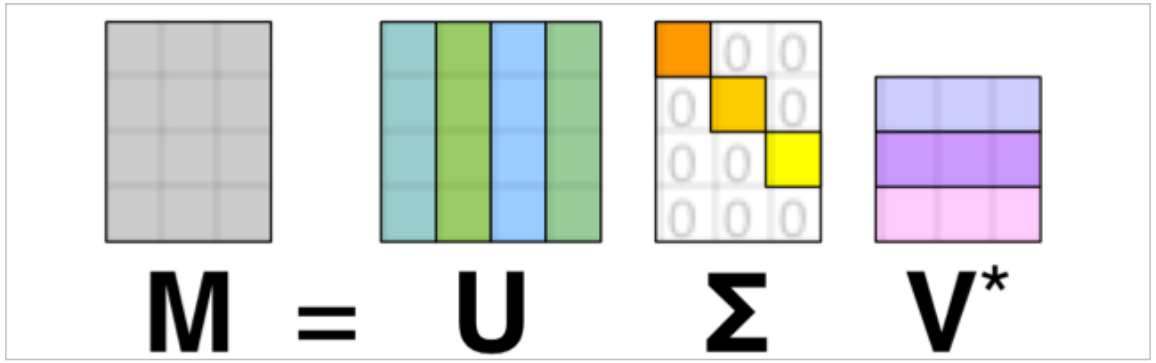
- Review SVD
  - It’s awesome - why you should love it
  - Singular values are almost math magic
- Bottleneck Scenarios – the need for stochastic methods
- Randomized SVD algorithms
  - Easy
  - Improvements
  - Pictures

SVD Review

That trick you learned in math class!

Eigendecomposition of a matrix is powerful, but matrix must be square

⇒Generalize to SVD



Any Matrix

Singular Values

U,V are unitary

If M is square the eigenvectors can be *U,V*

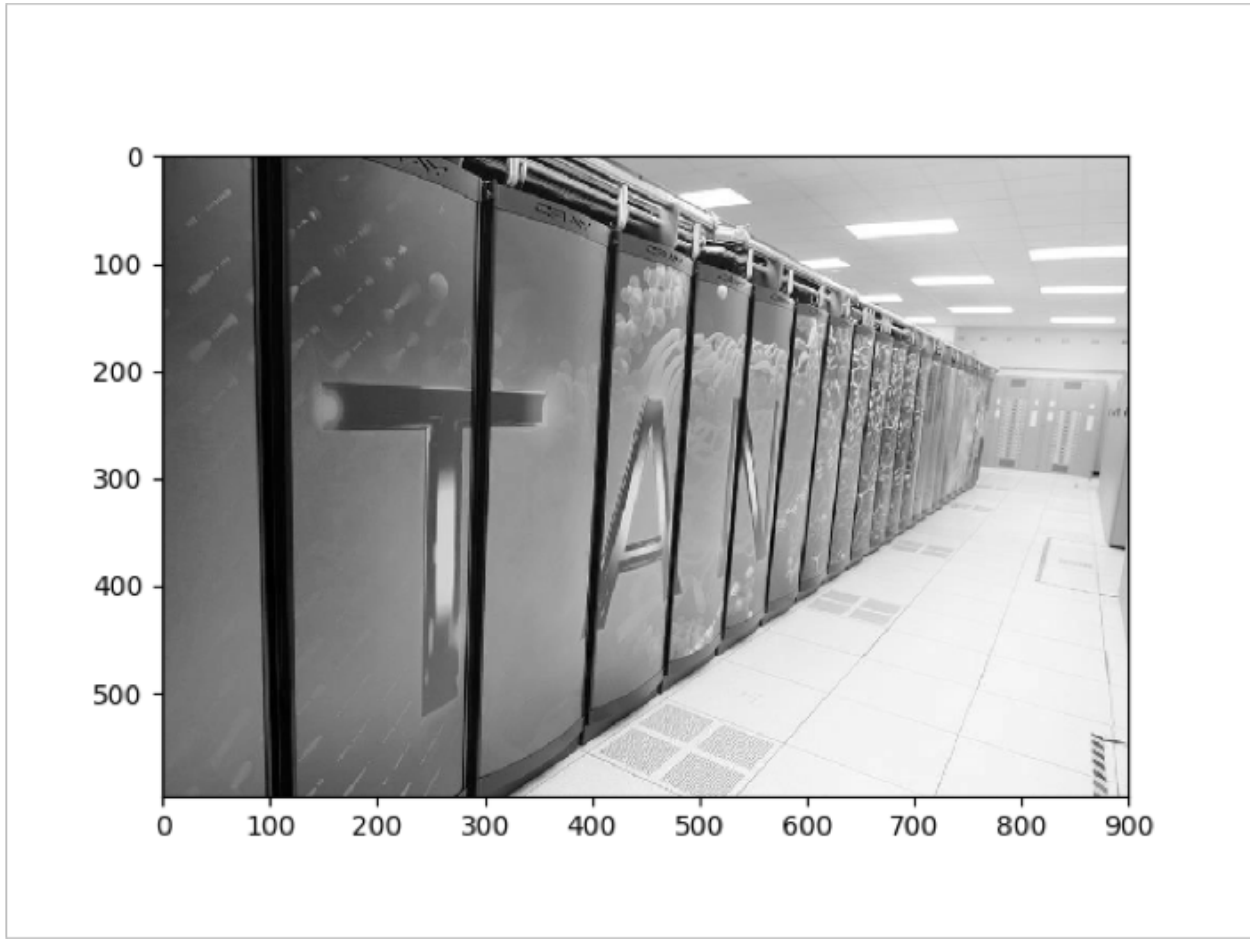
SVD can have a geometric interpretation for some *M*

Can approximate *M* by reducing singular values

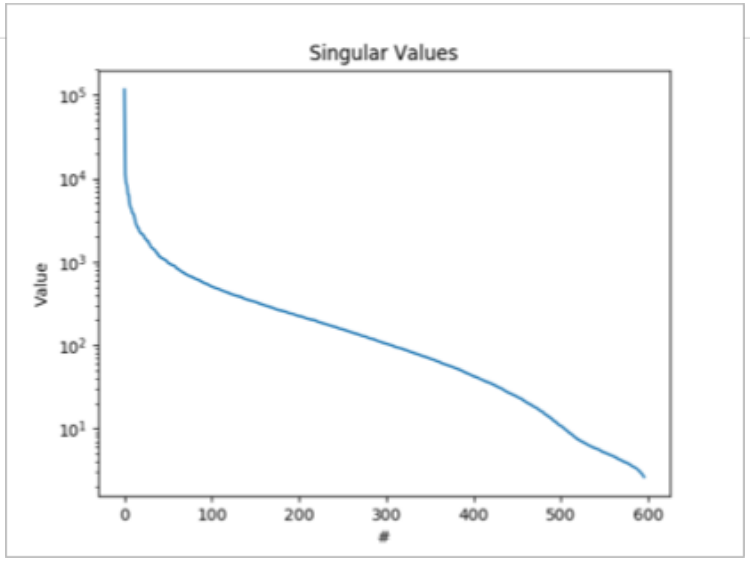
Example 1

$$M = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} = \underbrace{\begin{pmatrix} 0.32 & 0.88 & 0.41 \\ 0.52 & 0.24 & -0.82 \\ 0.82 & -0.4 & 0.41 \end{pmatrix}}_{U} \underbrace{\begin{pmatrix} 9.5 & 0 \\ 0 & 0.51 \\ 0 & 0 \end{pmatrix}}_{\Sigma} \underbrace{\begin{pmatrix} 0.62 & -0.78 \\ 0.78 & 0.62 \end{pmatrix}}_{V^\dagger}$$

Example 2



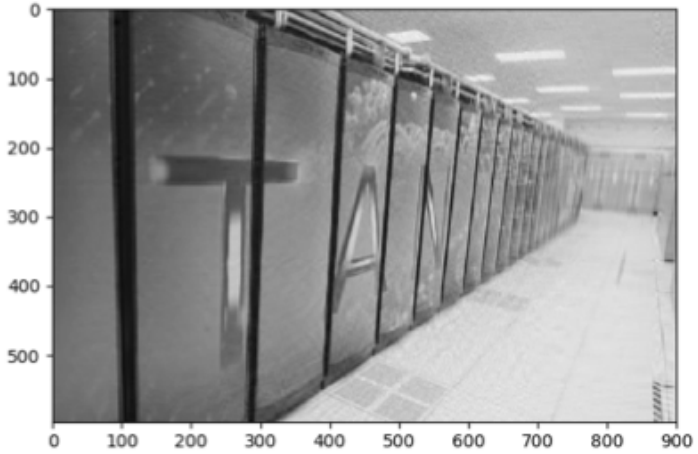
Example 2



Key: [% **Σ**=0] – [# remaining]

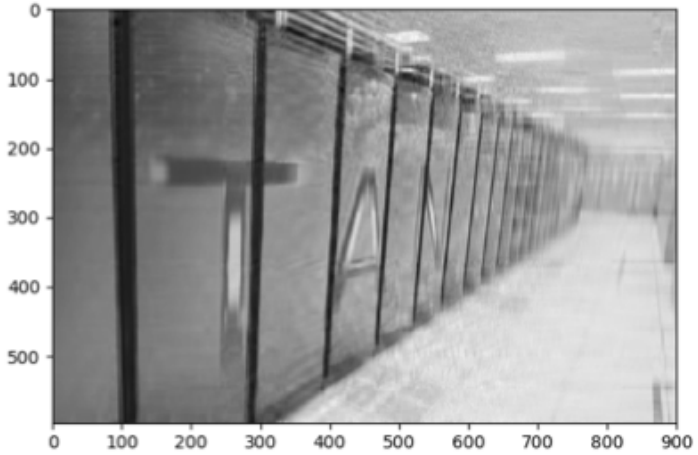
50% - 298

90% - 60



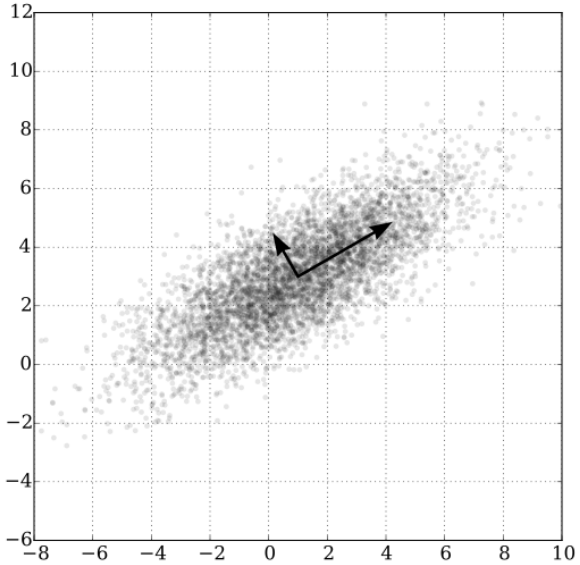
75% - 149

95% - 30



Where do we see SVDs in physics?

- Principal Component Analysis (PCA)
  - Look at dominate principal components – large singular values – to analyze multi dimension problem
- Easier linear algebra (matrix exponential, approximating data, etc)
- Clustering problems (similar to PCA)
- Calculating Entanglement Entropy
  - Schmidt Decomposition
- Pseudo-inverse



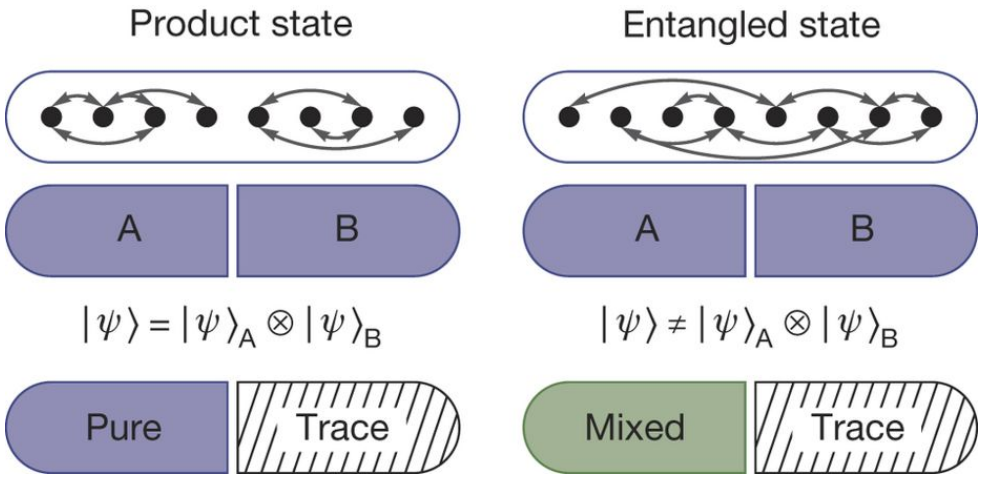
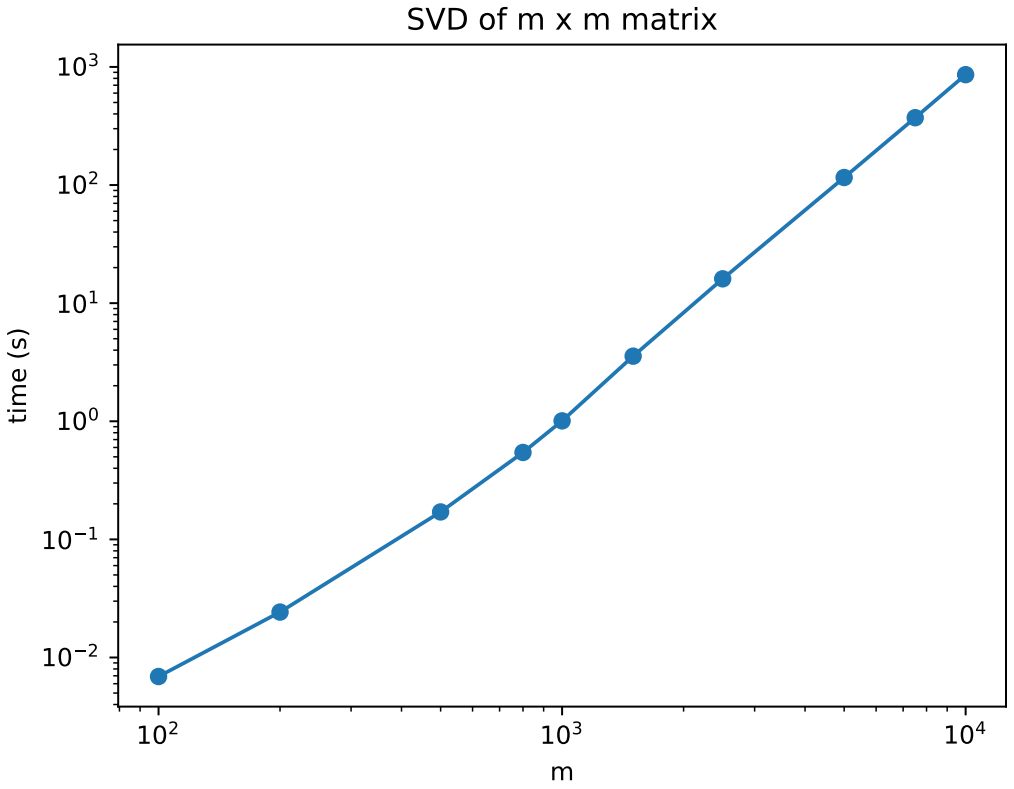


Image: Wikipedia, doi:10.1038/nature15750

SVD Bottlenecks – Full SVD Algorithm

$\sim O(mn^2)$



Large matrices take huge computational cost
Sometimes have hundreds of large matrices to SVD (e.g. facebook)

"…the adjacency matrix of Facebook users to Facebook pages induced by likes, with size **$O(10^9) \times O(10^8)$**"

Source: Facebook research

$\sim O(m)$ Passes through matrix

SVD Algorithm

~complicated~

Method 1 – Power Method Lanczos(!)

1. Notice that an SVD is the same as

$$M = U\Sigma V^\dagger \Leftrightarrow Av = Ev$$

$$\begin{pmatrix} 0 & M \\ M^\dagger & 0 \end{pmatrix} \begin{pmatrix} U \\ V \end{pmatrix} = \Sigma_{ii} \begin{pmatrix} U \\ V \end{pmatrix}$$

2. Notice that solving an eigenvalue problem is the same as

$$M^N x \to Ex,\ N \gg 1$$

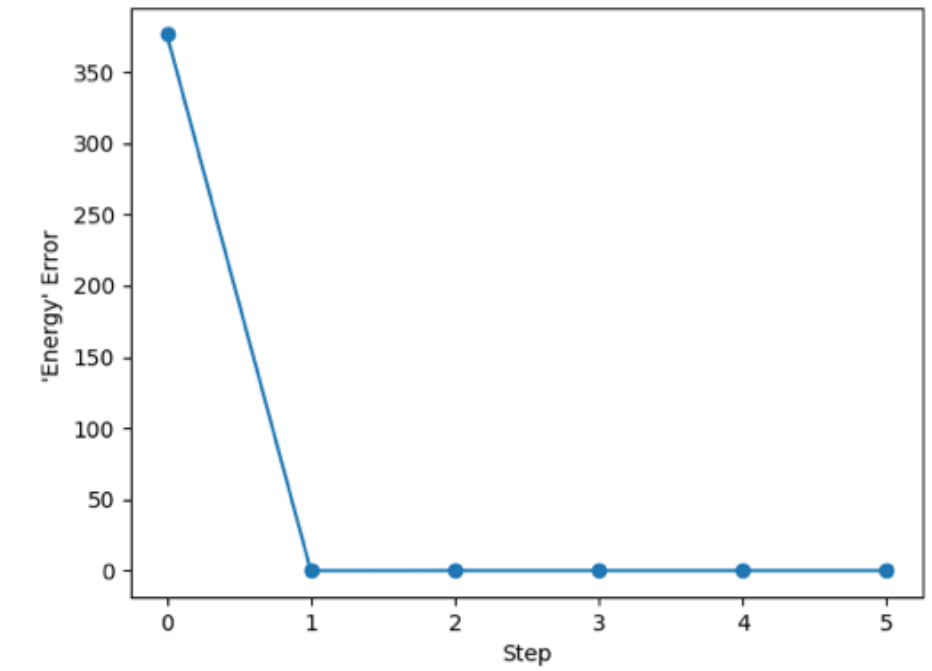
3. Start with a random vector x then apply the Hamiltonian, normalizing after each step

Pro:

- Physicists know how to do this!
- Parallelizes very well

Cons:

- Hard to find many principal components
- Large degeneracies will slow down convergence
- Larger storage/GMM cost



“Easy” Randomized SVD

Goal: obtain SVD for k singular values of a m x n matrix M, assuming $m > n$

1. Create a n x k matrix of random [normal] samples Ω
2. Do a QR decomposition on the sample matrix ΩM
   a. Reminder that QR = (orthogonal matrix) (upper triangular)
   b. QR is slow but accurate
   c. Orthogonal matrix Q is m x k
3. Create “smaller” k x n matrix $B = \mathrm{Q}^{\dagger} M$
4. Do SVD on $B = u \Sigma V^{\dagger}$
5. Get original $\mathrm{U} = \mathrm{Q} u$

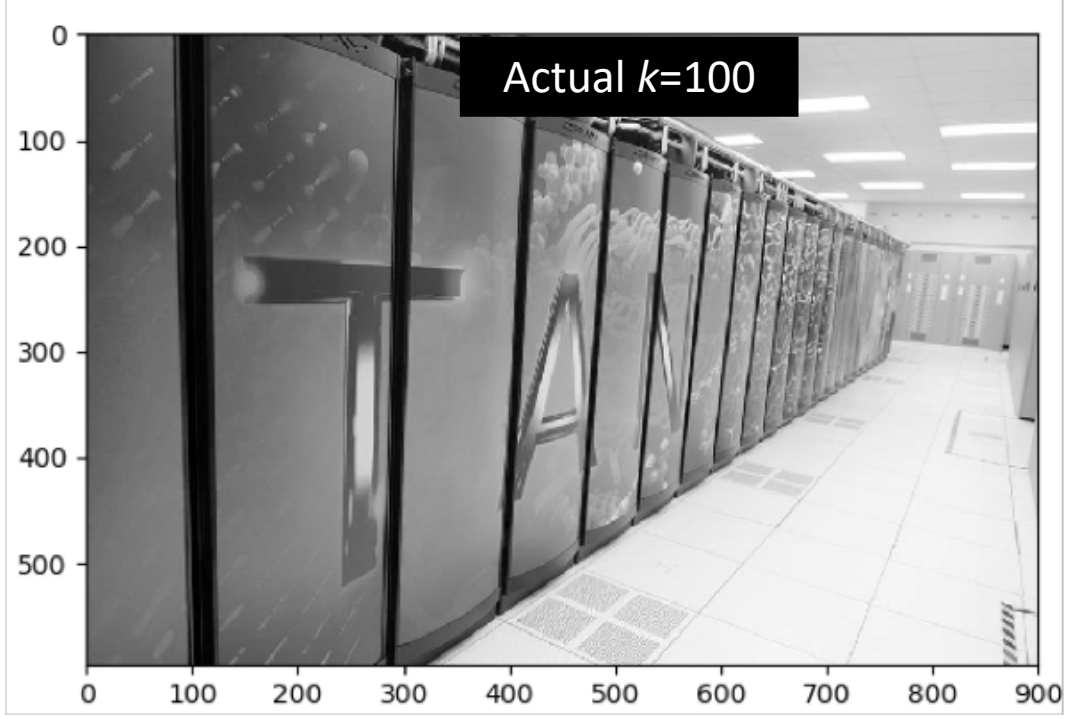
Random values are hopefully superposition of correct basis vectors

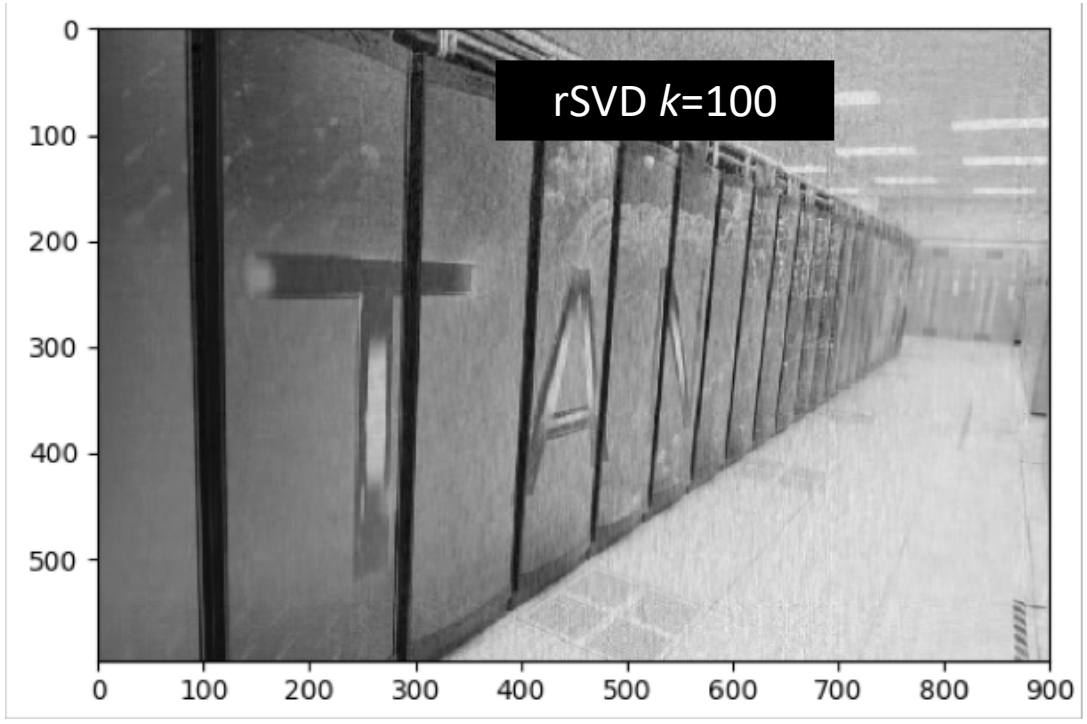
“Randomized Range Finder”

Source: Halko, Martinsson and Tropp (2009)

Visual Example

```python
def easy_randomSVD(M,k):
    m,n = M.shape

    if m < n:
        M=M.T
    Ω   = np.random.normal(0,1,(M.shape[1],k))
    Q,R = np.linalg.qr(M @ Ω)
    B   = Q.T @ M
    u,S,Vt = np.linalg.svd(B,full_matrices=False)

    U = Q @ u
    if m<n:
        return Vt.T,S.T,U.T,
    else:
        return U,S,Vt
```

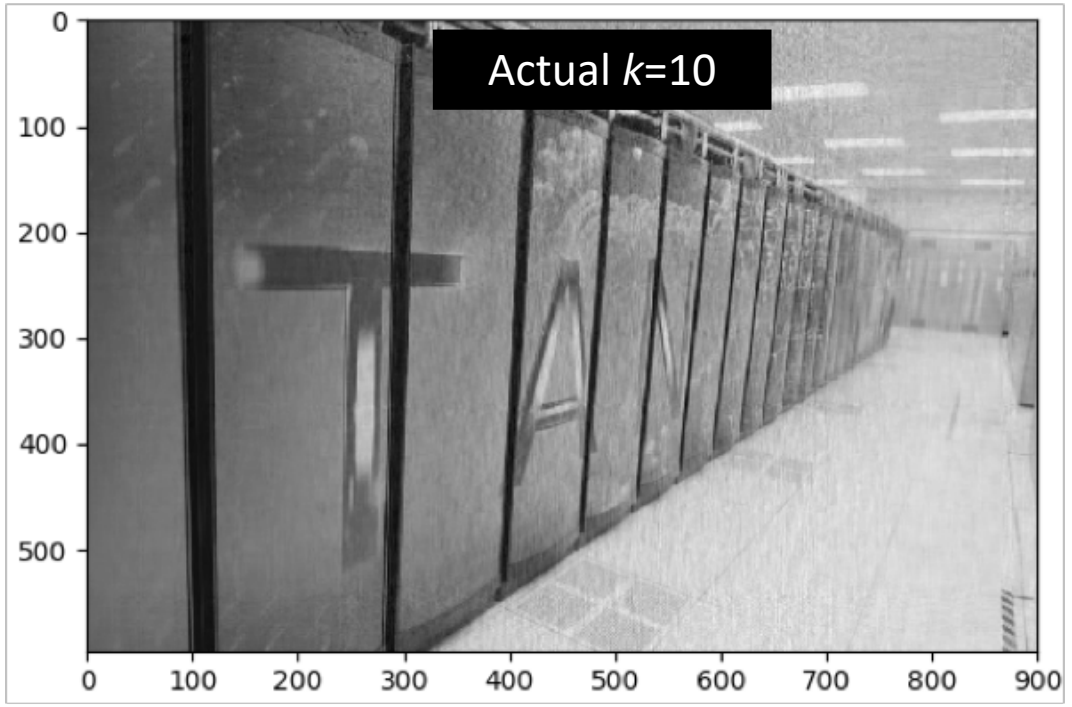


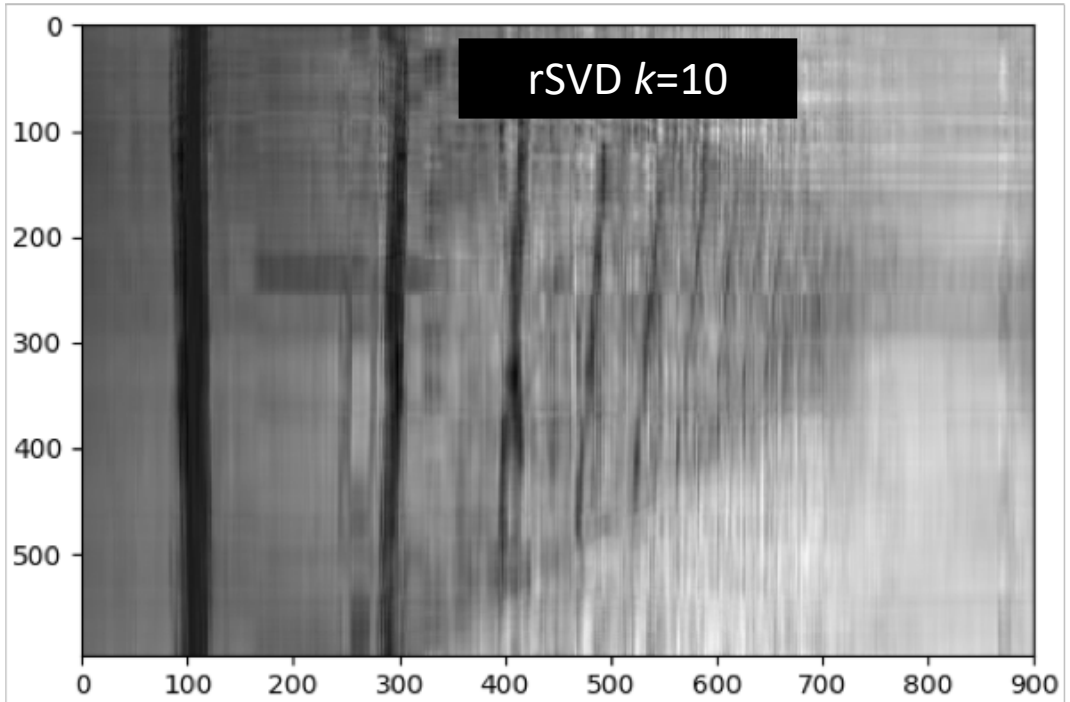


Thanks to smortezavi's example code

Visual Example

```python
def easy_randomSVD(M,k):
    m,n = M.shape

    if m < n:
        M=M.T
    Ω   = np.random.normal(0,1,(M.shape[1],k))
    Q,R = np.linalg.qr(M @ Ω)
    B   = Q.T @ M
    u,S,Vt = np.linalg.svd(B,full_matrices=False)

    U = Q @ u
    if m<n:
        return Vt.T,S.T,U.T,
    else:
        return U,S,Vt
```





Thanks to smortezavi's example code

Comments on Randomized SVD

- By using certain random sample matrices we can speed up the algorithm and form less intermediate matrices

- How good can we do?
  - Bounded by error of using a k rank matrix
  - Can sample several times to get another error estimate
  - Con - Assumes singular values decay slowly
    - What if we use the Lanzcos idea and project into the M subspace?
    - Best: Combine both techniques!

Improve Range Suspace

Power Method

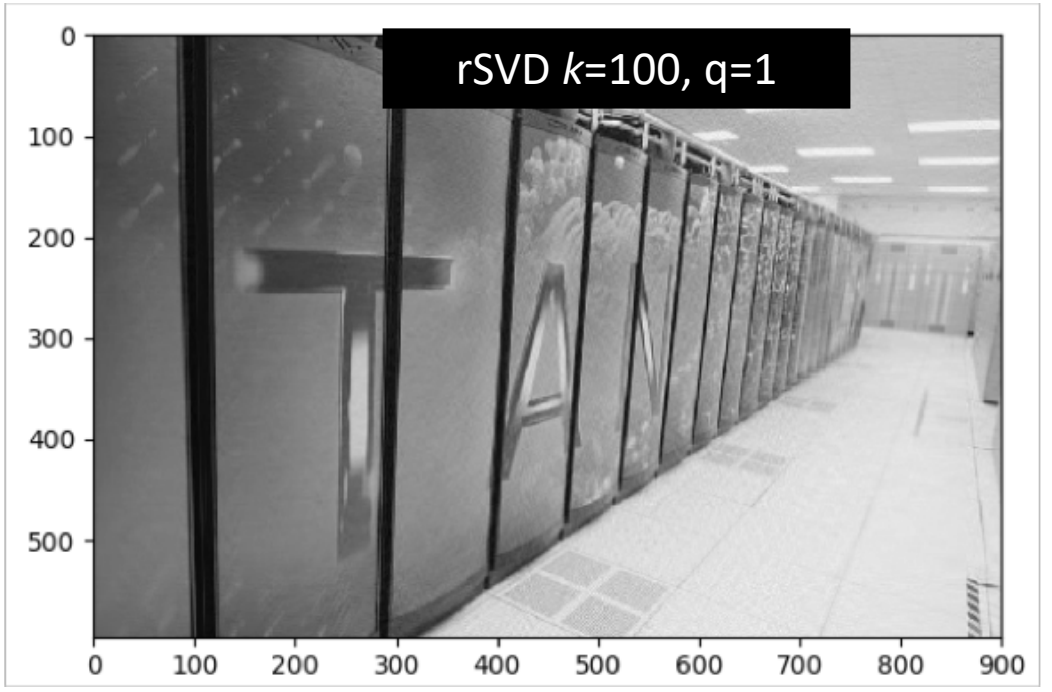
$$\Omega M \rightarrow (MM^{\dagger})^{q}\,\Omega M$$

Rounding error problem
Instead do QR every step, alternate M, Mt

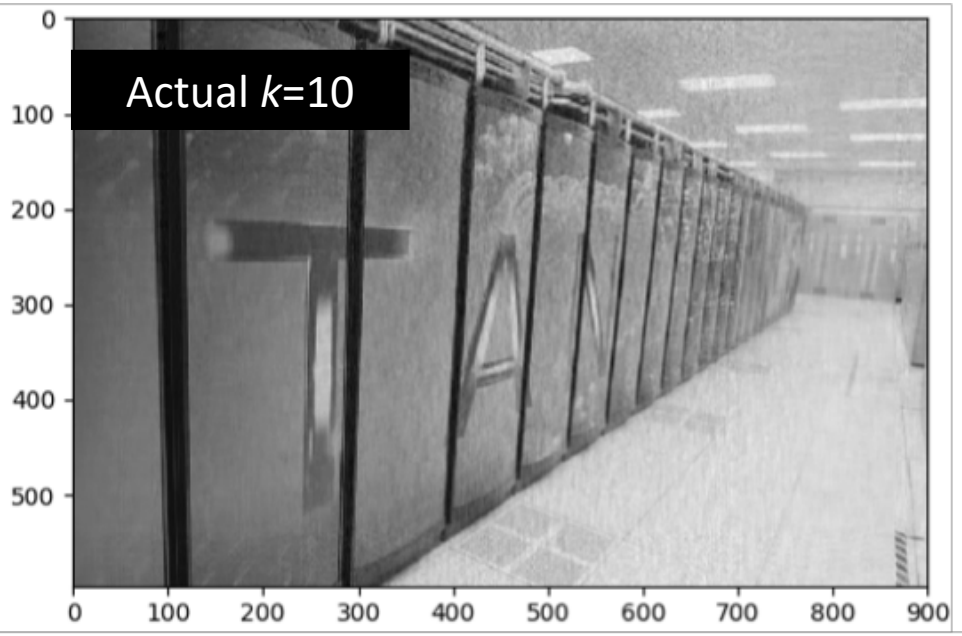
```python
def full_unstable_randomSVD(M,k,q):
    m,n = M.shape

    if m < n:
        M=M.T
    Ω = np.random.normal(0,1,(M.shape[1],k))
    Y = M @ Ω
    #============================
    MMt = (M @ M.T) #cache
    for j in range(q):
        Y = MMt @ Y
    #============================
    Q,R = np.linalg.qr(Y)
    B   = Q.T @ M
    u,S,Vt = np.linalg.svd(B,full_matrices=False)

    U = Q @ u
    if m<n:
        return Vt.T,S.T,U.T,
    else:
        return U,S,Vt
```

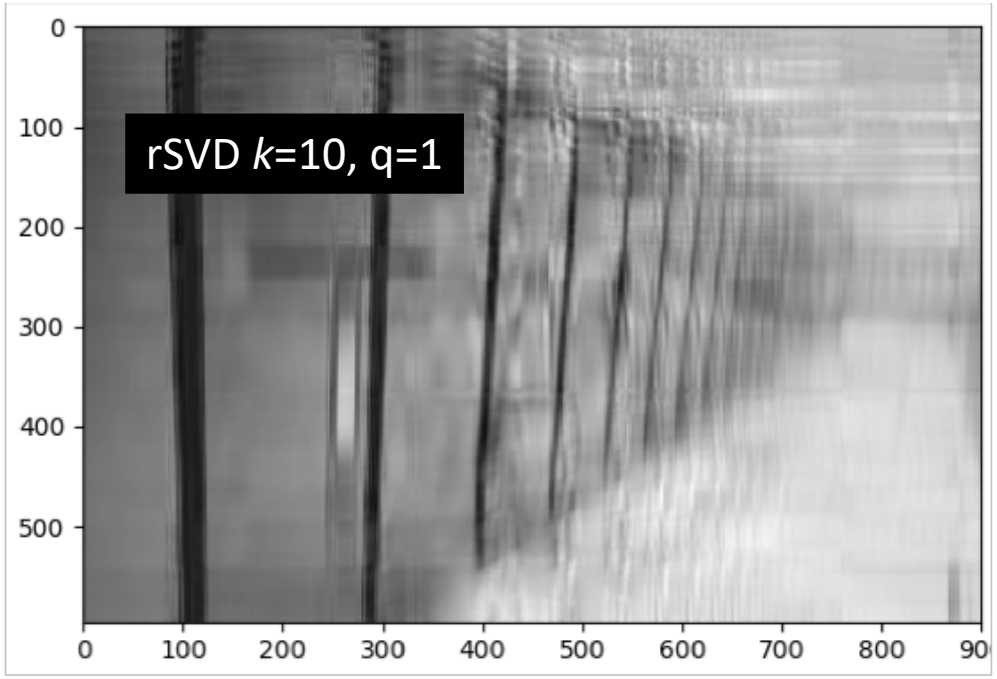
Visual Example

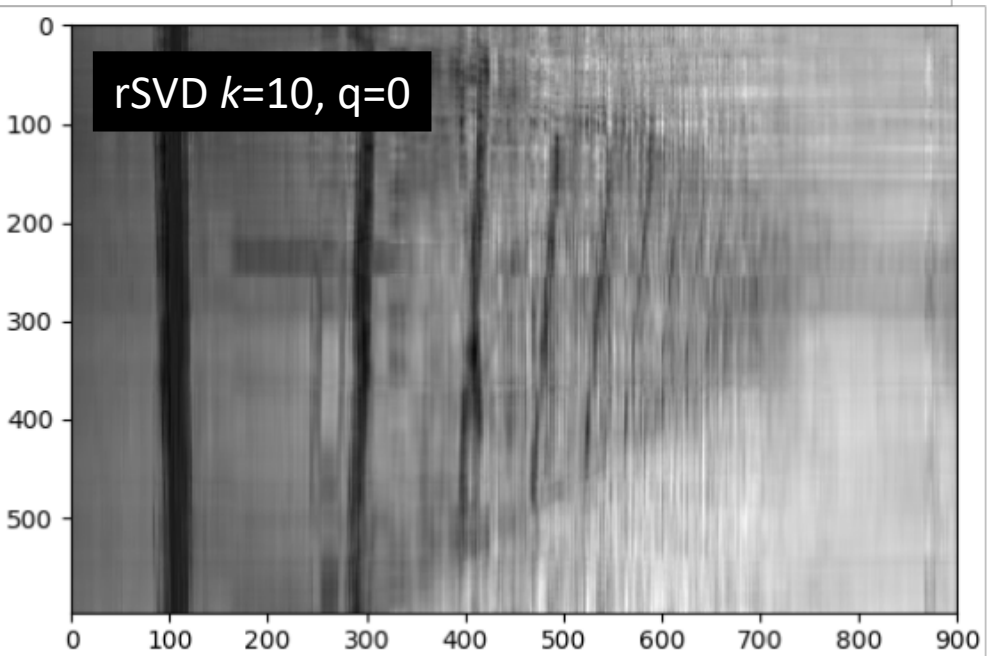
```python
def full_randomSVD(M,k,q):
    m,n = M.shape
    if m < n:
        M=M.T

    Ω    = np.random.normal(0,1,(M.shape[1],k))
    Q,R = np.linalg.qr(M @ Ω)

    #Randomized Subspace Iteration
    for j in range(q):
        Q,R = np.linalg.qr(M.T @ Q)
        Q,R = np.linalg.qr(M   @ Q)

    B       = Q.T @ M
    u,S,Vt = np.linalg.svd(B,full_matrices=False)

    U = Q @ u
    if m<n:
        return Vt.T,S.T,U.T,
    else:
        return U,S,Vt
```

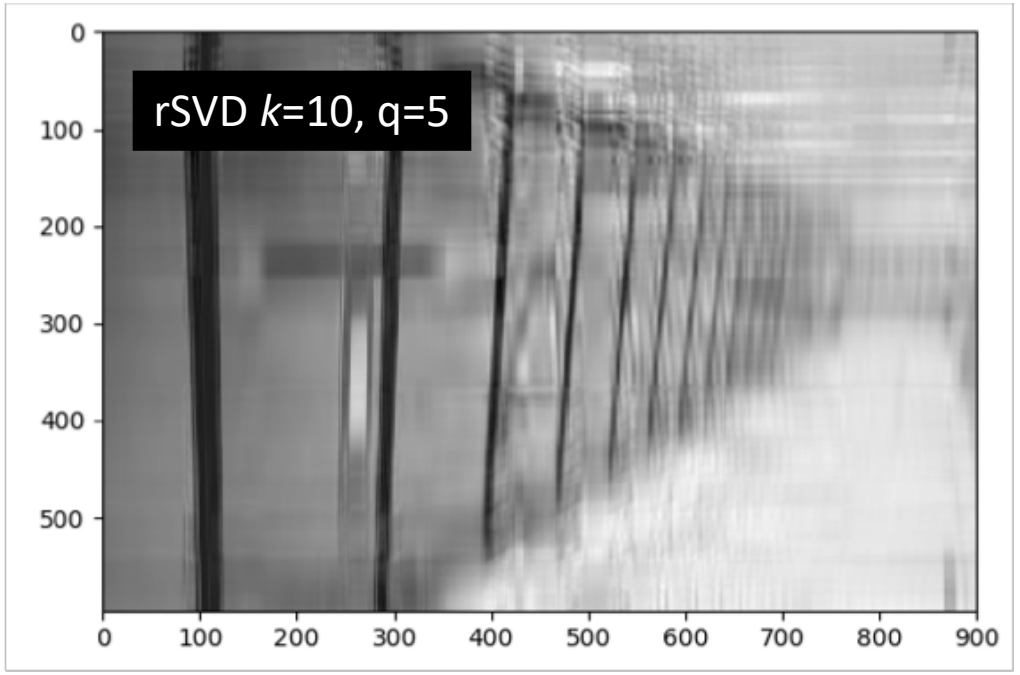


Visual Example



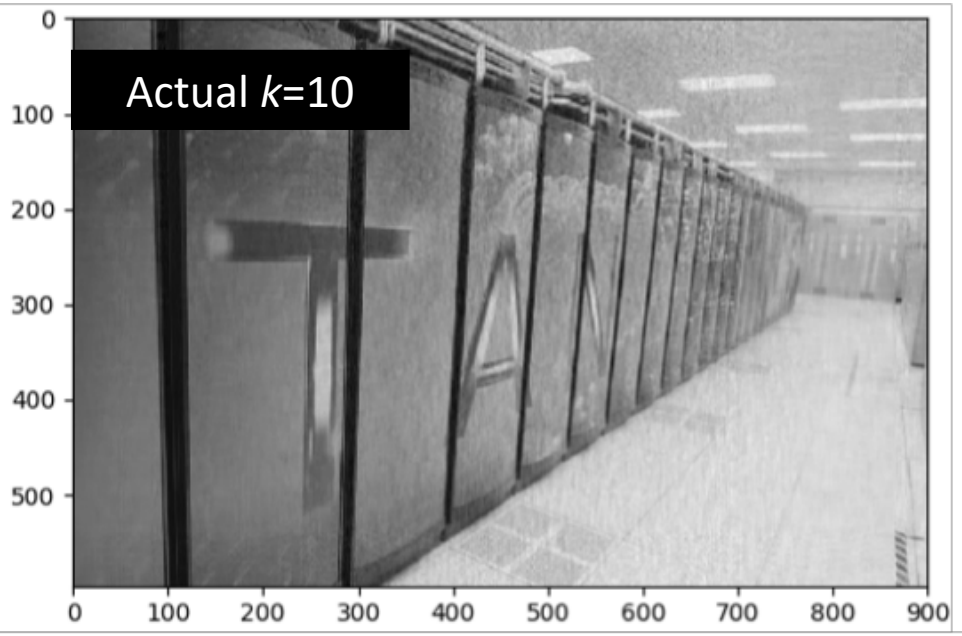


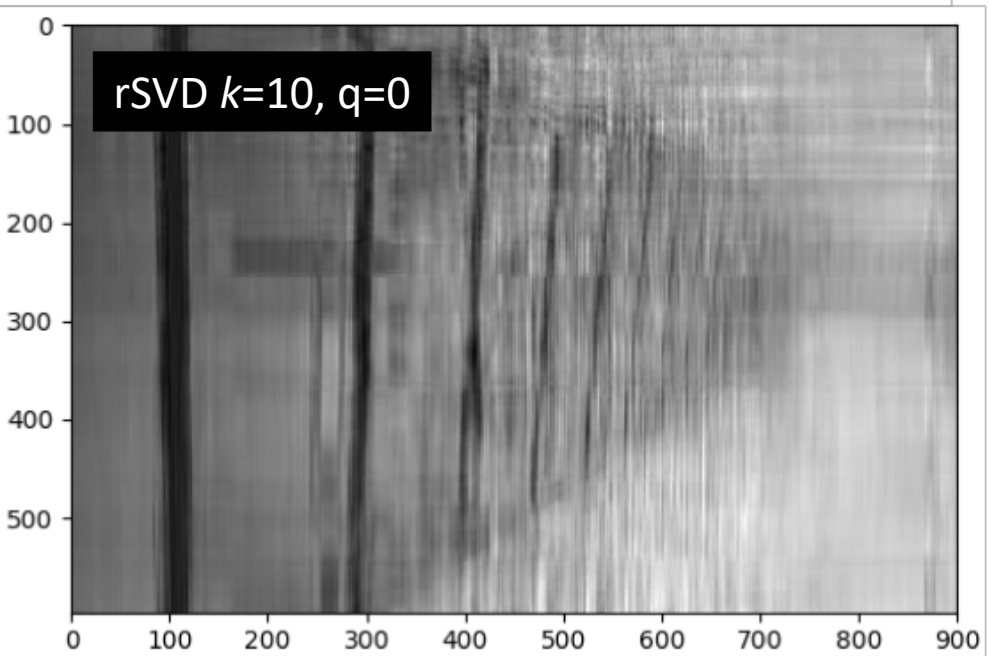


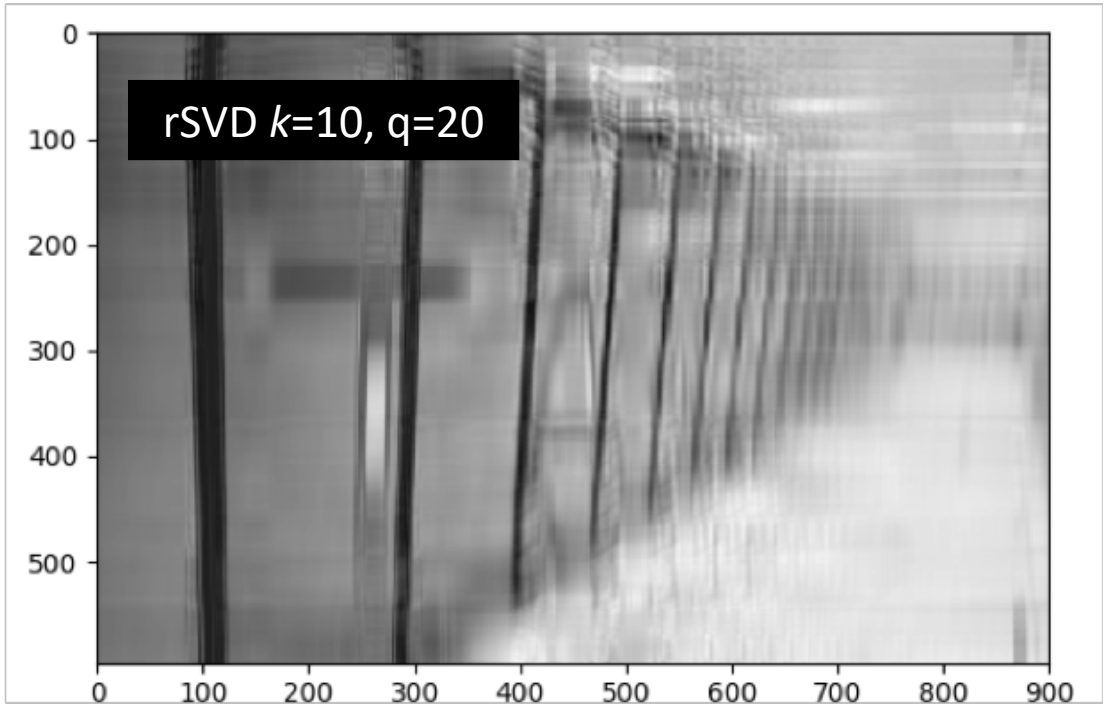


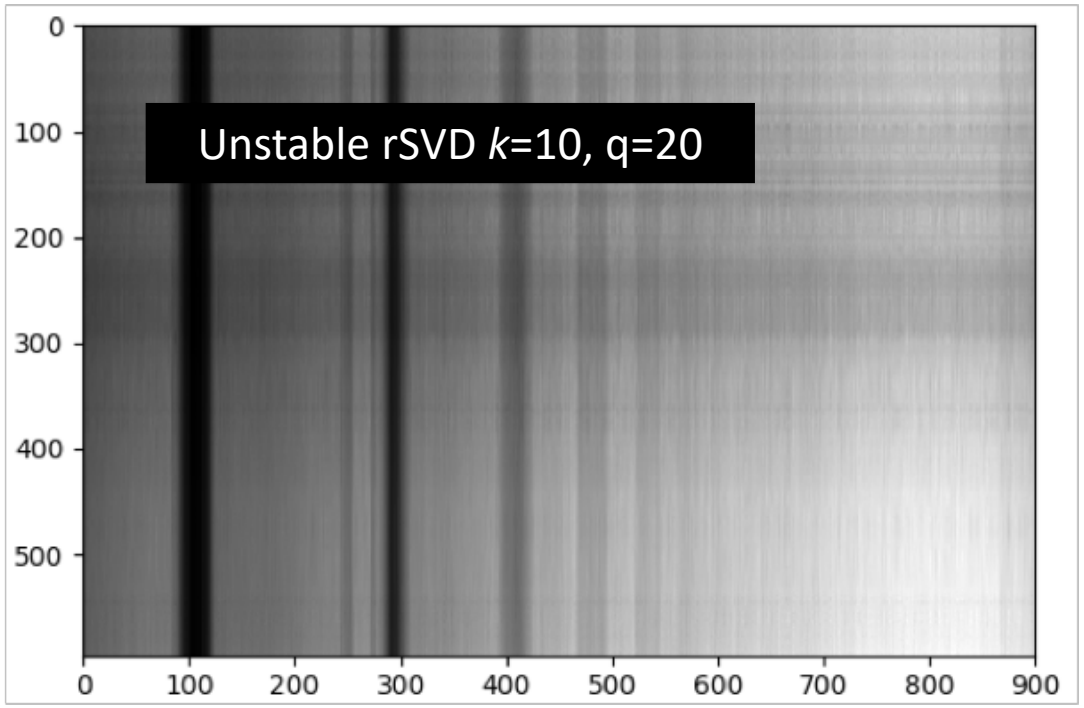
Timing (s):
SVD: 0.613
rSVD q=0: 0.0096
rSVD q=1: 0.022

Visual Example









Conclusions

- SVD is a powerful technique but slow for large matrices
- Because we don’t always need all the singular values
  we can guess how many we need to make a faster algorithm
- Randomized SVD estimates smaller subspace to perform a full SVD
  - Can be sped up by using smart random sampling
  - Can be improved by using a power method or oversampling

Thanks!