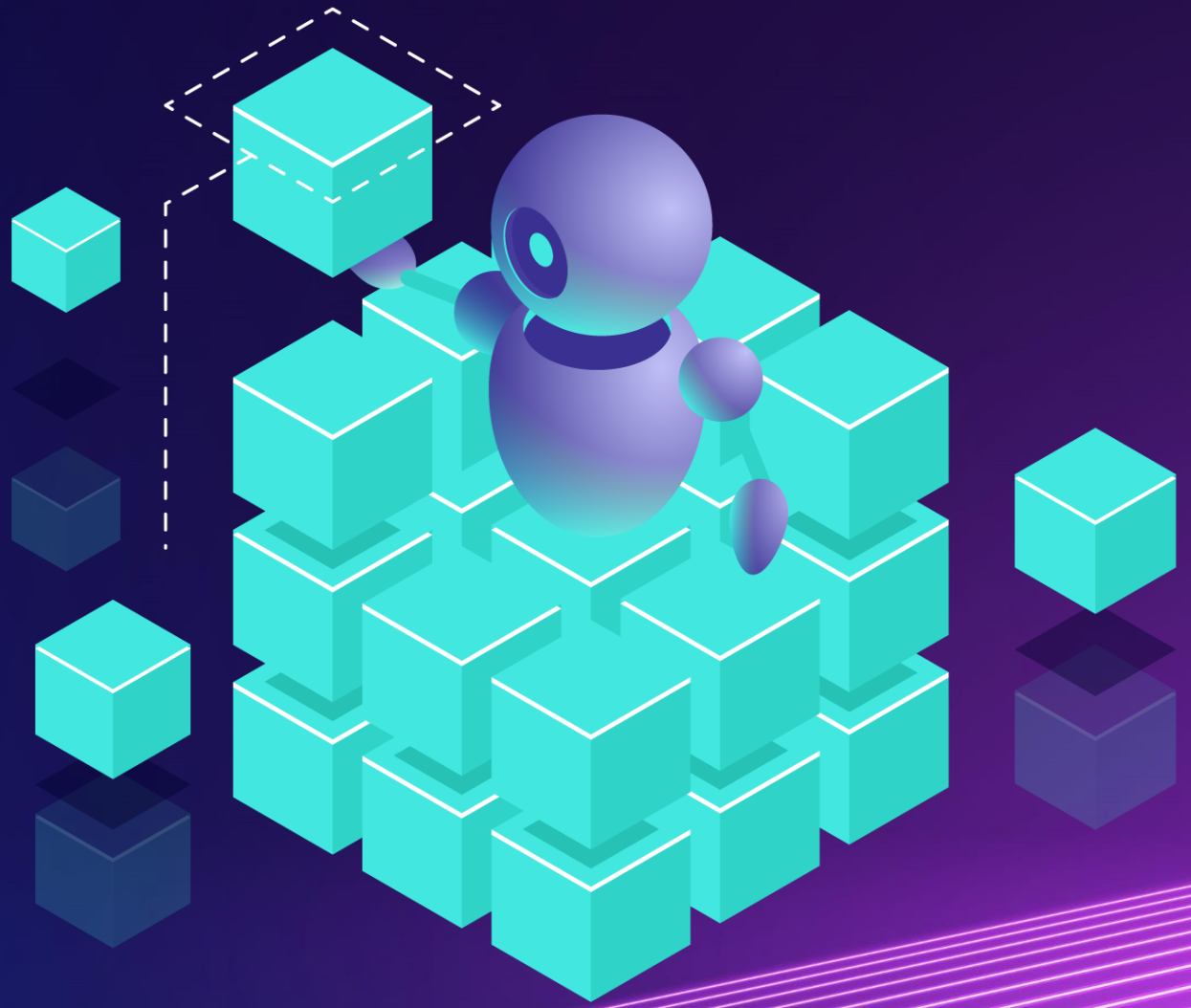
Fast Deterministic CUR Matrix Decomposition

Sandyiq Siziqtiq Algebra

Kuzmina Ekaterina
Dmitrii Gromyko
Ilya Barsky
Dmitrii Krukov

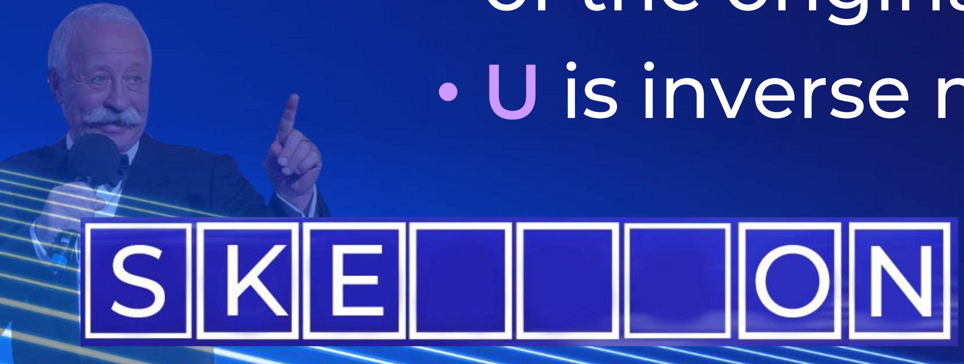
Why do we need matrix decomposition like SVD and CUR?

- Dealing with enormous matrices can be a **problem**
- **Singular Value Decomposition** in Feature extraction in machine learning and Data Compression.
- **CUR** decomposition is used in Model reduction, Speedup of computations in numerical analysis, Matrix factorization



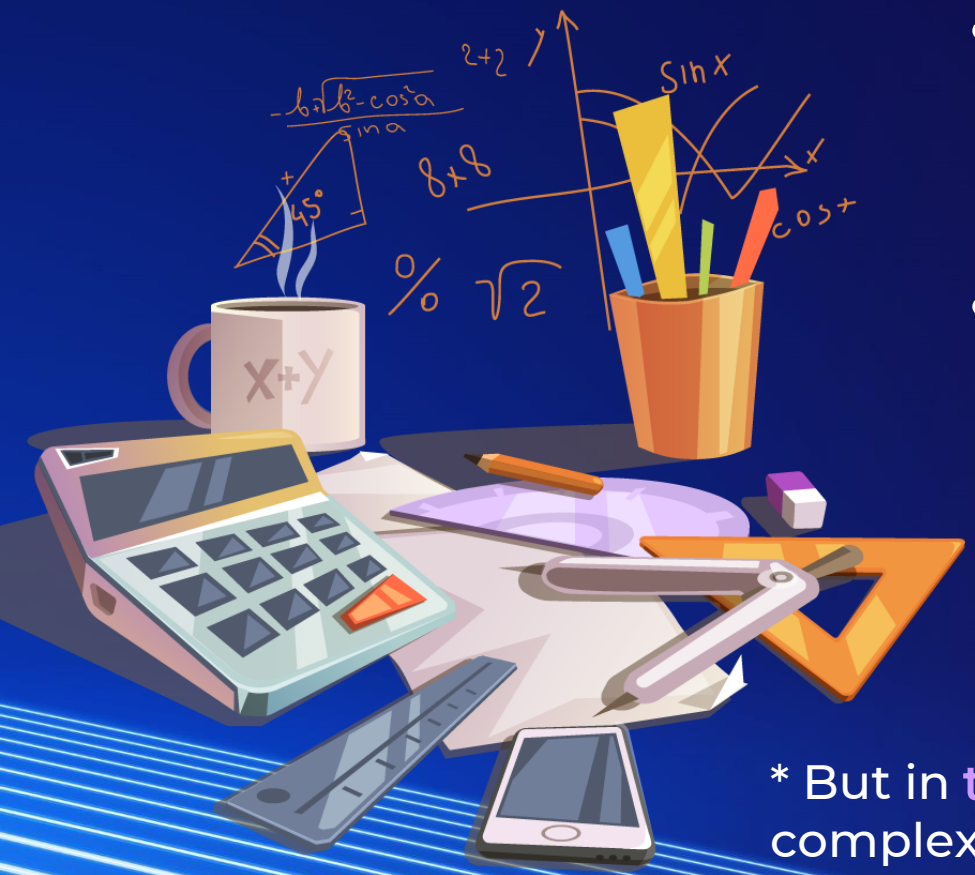
CUR Decomposition

- C and R are subsets of c columns and r rows of the original data matrix, respectively
- U is inverse matrix at the intersection



Idea of Deterministic CUR. Fast CUR

- The main idea is to reduce the problem of CUR decomposition to **convex optimization problem**

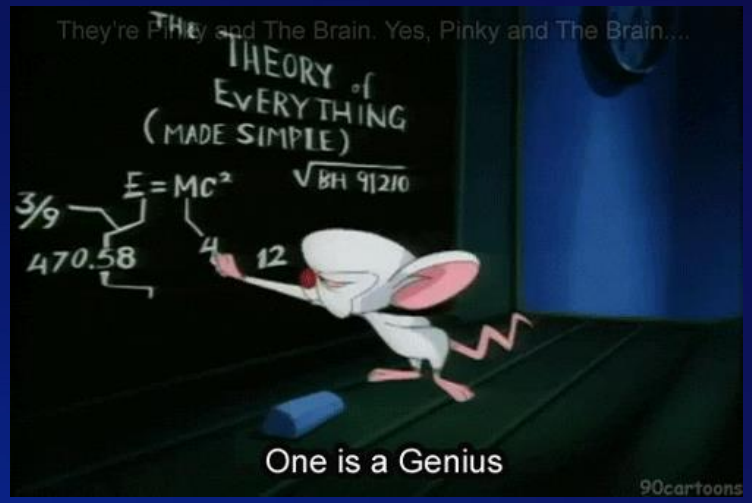


- The **original** deterministic CUR algorithm complexity is $O(np^2)$
- The **central article** of this project proposes*

$$O(p[n(t_m + pt_u S) + Q])$$

* But in **the worst case** the complexity reaches $O(np^2)$

Math behind Deterministic CUR



- Matrix decomposition is seen as a convex optimization problem with sparsity-inducing norms

$$\min_{W \in \mathbb{R}^{P \times P}} \frac{1}{2}\|X - XW\|_F^2 + \lambda \sum_{i=1}^{P} \|W_{(i)}\|_2$$

$$W_{(i)} = (1 - \lambda / \|Z_i\|_2)_+ Z_i$$

- Where

$$(1 - \lambda / \|Z_i\|_2)_+ = \begin{cases} 1 - \lambda / \|Z_i\|_2, & \text{if} > 0 \\ 0, & \text{otherwise} \end{cases}$$

$$Z_i = X^{(i)T}\left(X - \sum_{j \neq i}^{P} X^{(j)} W_{(j)}\right) \rightarrow O(np^2)$$

And more math. Fast CUR

$$\bar{K}_i = \tilde{K}_i + \|\Delta W_{(i)}\|_2 + \delta \|G_{(i)}\|_2 \rightarrow O(p)\,time$$

$$\delta = \sqrt{\|\Delta W\|_F^2 - \|\Delta W_{(j)}\|_2^2 + \|\Delta W'_{(j)}\|_2^2}$$

- Where $\bar{K}_i \leq \lambda,\ W_{(i)} = 0$ for the *i*-th row vector.

$$\underline{K}_i = \bar{K}_i - 2\|\Delta W_{(i)}\|_2 - 2\delta \|G_{(i)}\|_2 \rightarrow O(1)\,time$$

- Where $\underline{K}_i \leq \lambda,\ W_{(i)} \neq 0$ for the *i*-th row vector.

And more-more math. Stochastic CUR

1. Compute $v_1, \ldots, v_k$ (the top k right singular vectors of A) and the normalized statistical leverage scores

$$A^j = \sum_{\xi=1}^{r} (\sigma_\xi u^\xi) v_j^\xi \rightarrow \text{replace } r \text{ with } k<r$$

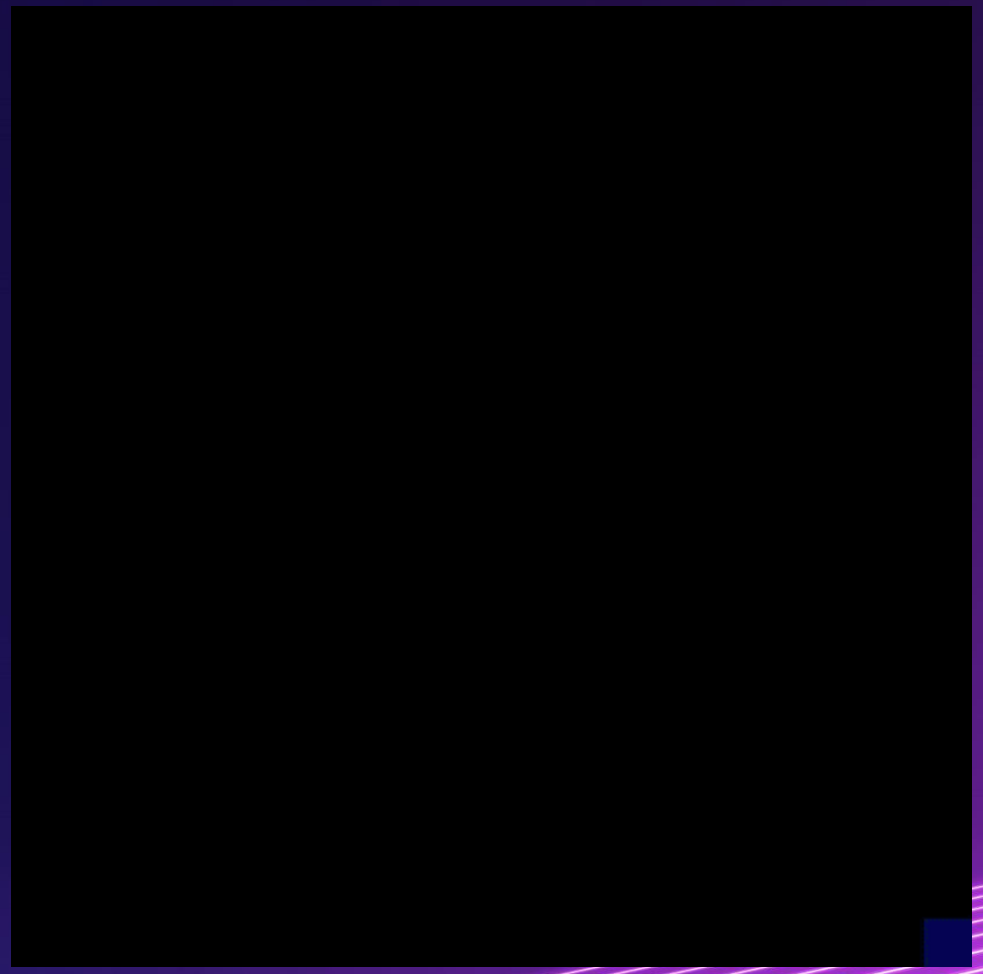
Normalized statistical leverage: $\pi_j = \frac{1}{k} \sum_{\xi=1}^{k} (v_j^\xi)^2$

2. Keep the *j*-th column of A with probability $p_j = \min\{1, c\pi_j\}$, for all $j \in \{1, \ldots, n\}$, where $c = O(k \cdot \log k/\epsilon^2)$

3. Return the matrix C consisting of the selected columns of A.

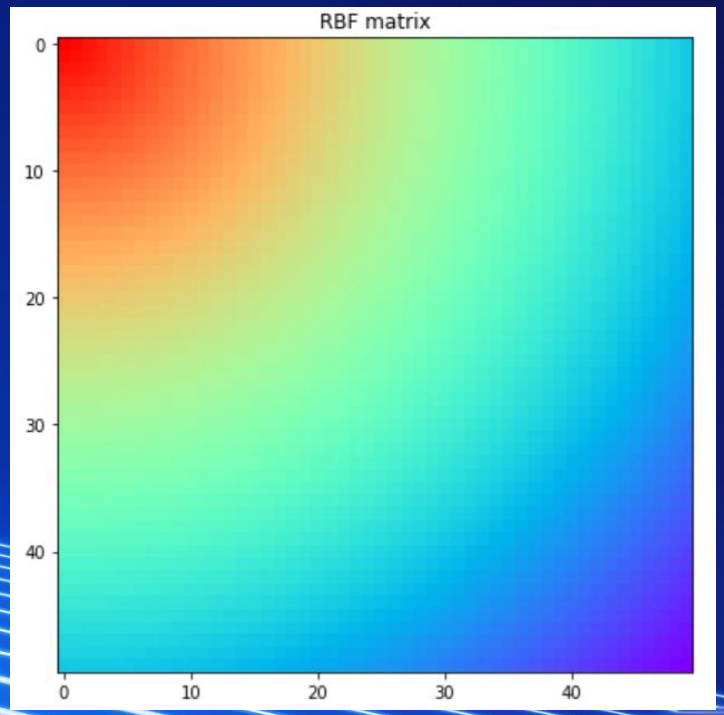
Stochastic vs Deterministic

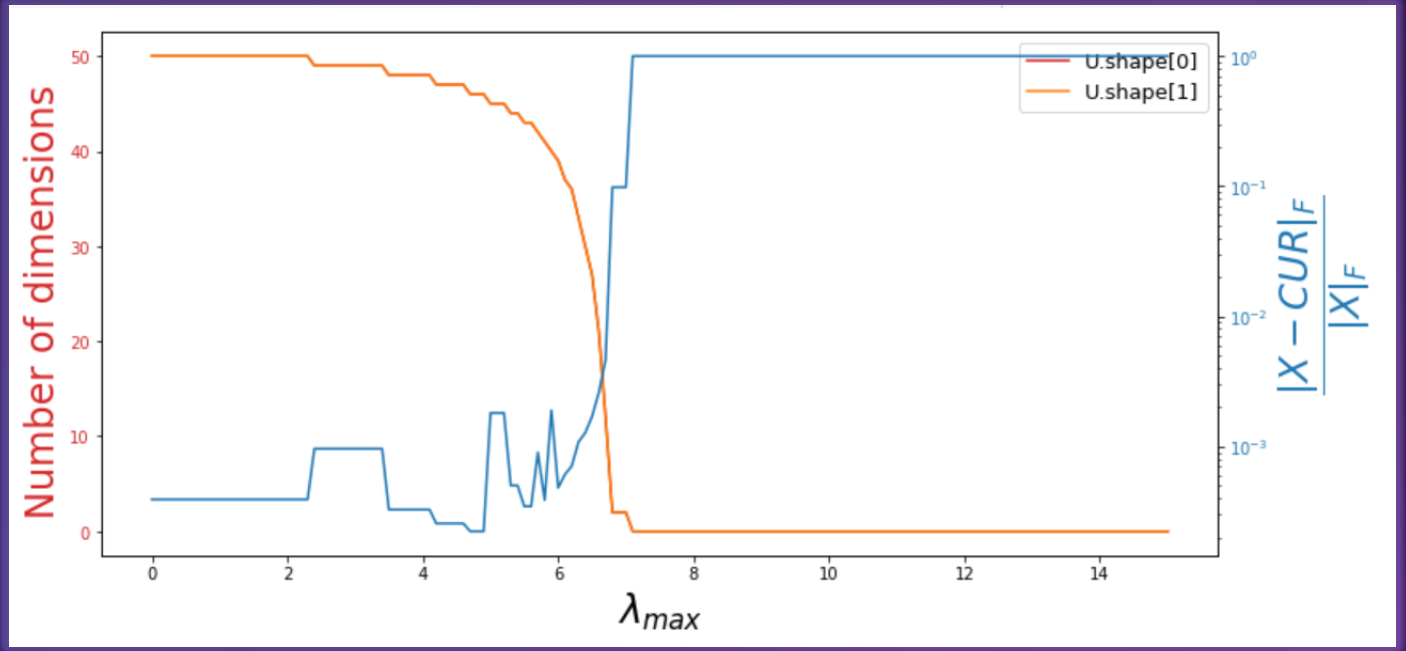
- **Stochastic** algorithm is much faster than Deterministic
- **Deterministic** doesn't use SVD
- **Deterministic** always has a unique solution

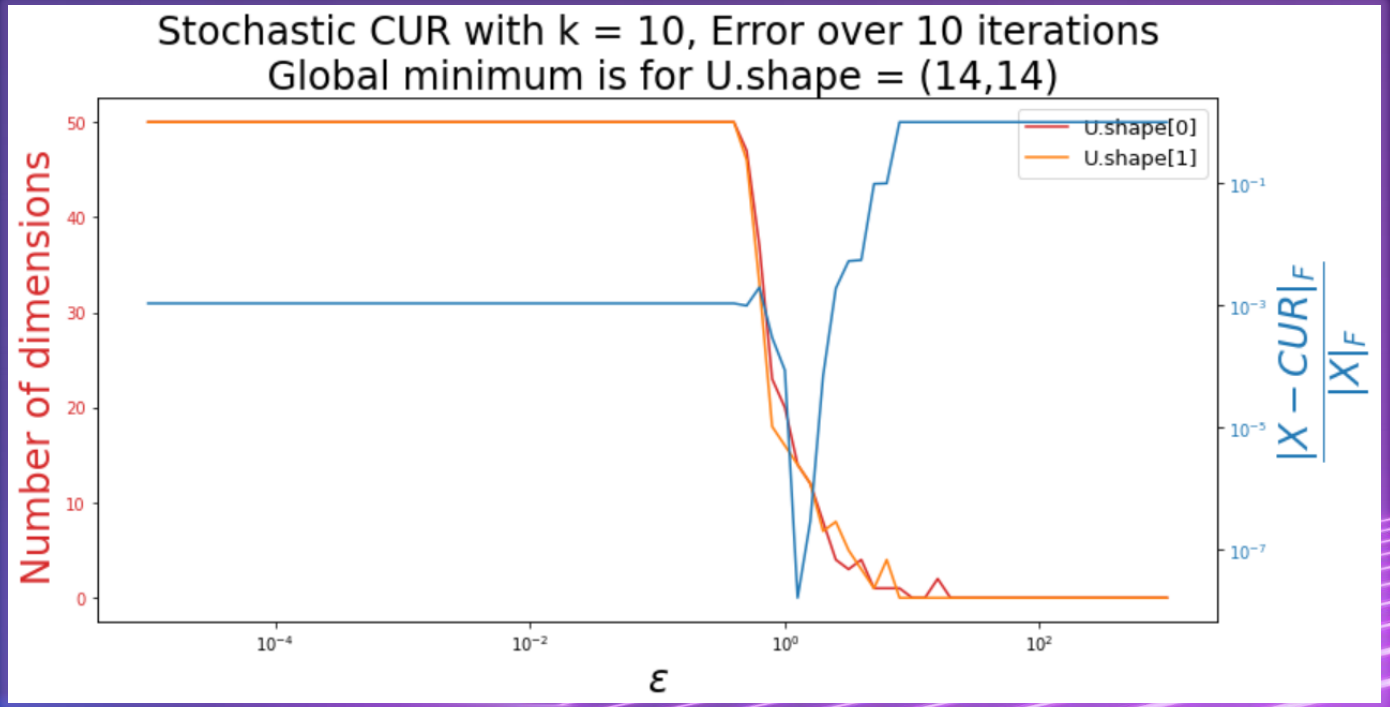


Stochastic vs Deterministic. RBF Matrix

```
#RBF matrix for R+
N = 50
xi = np.array([i/(N-1) for i in range(N)])
xj = np.array([i/(N-1) for i in range(N)])
xx, yy = np.meshgrid(xi, xj)
A_rbf = np.exp(-np.sqrt(xx**2 + yy**2))
```

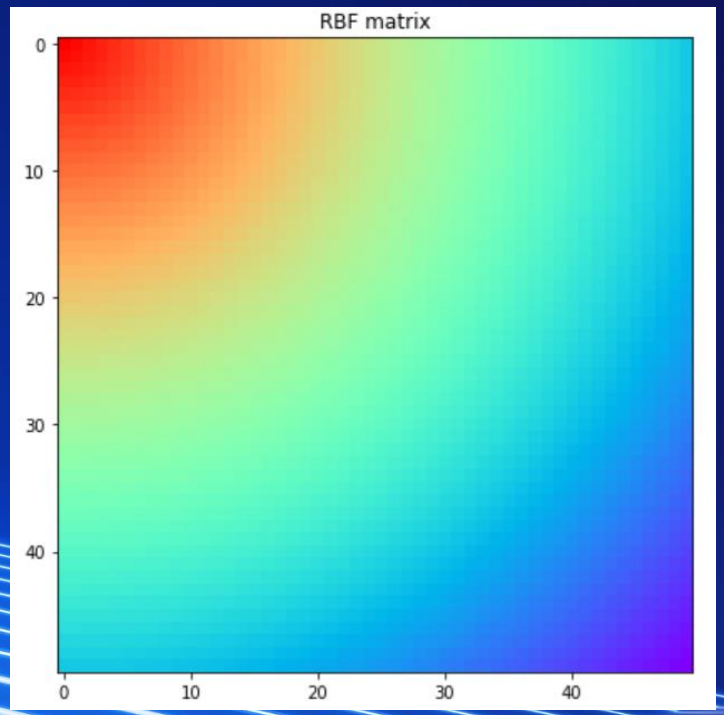


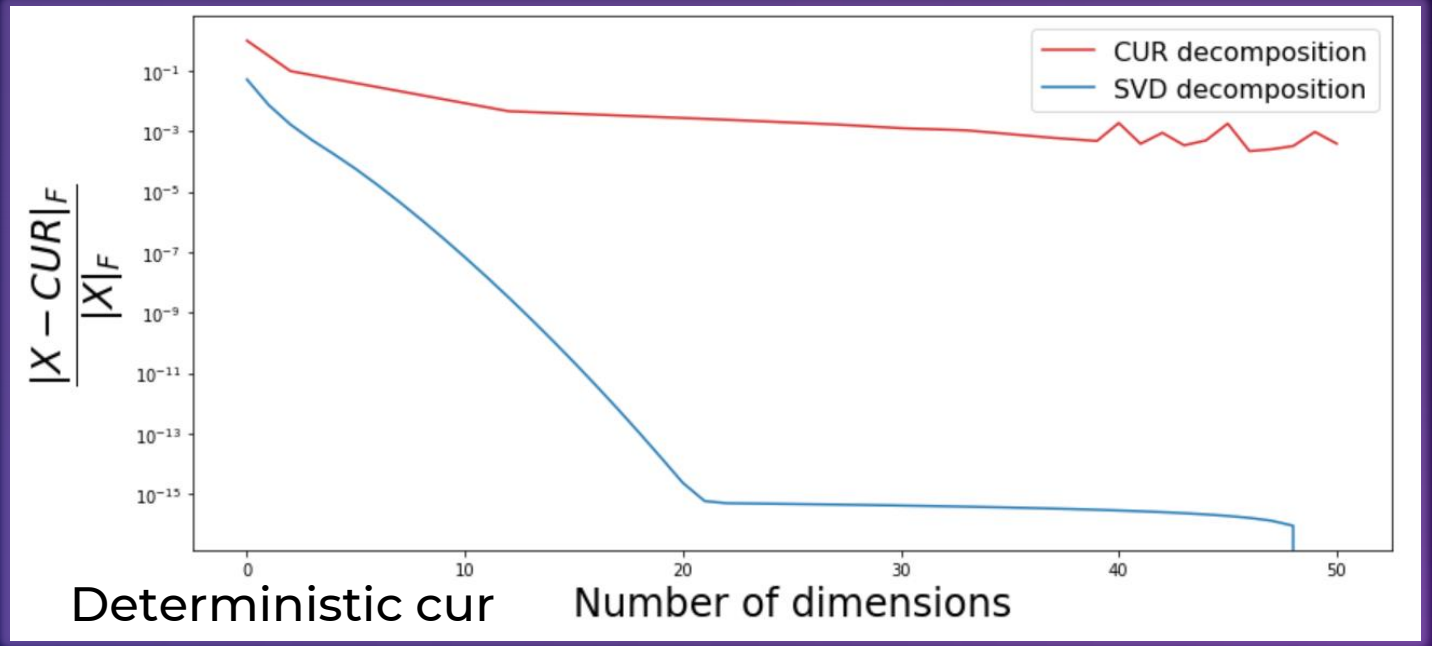


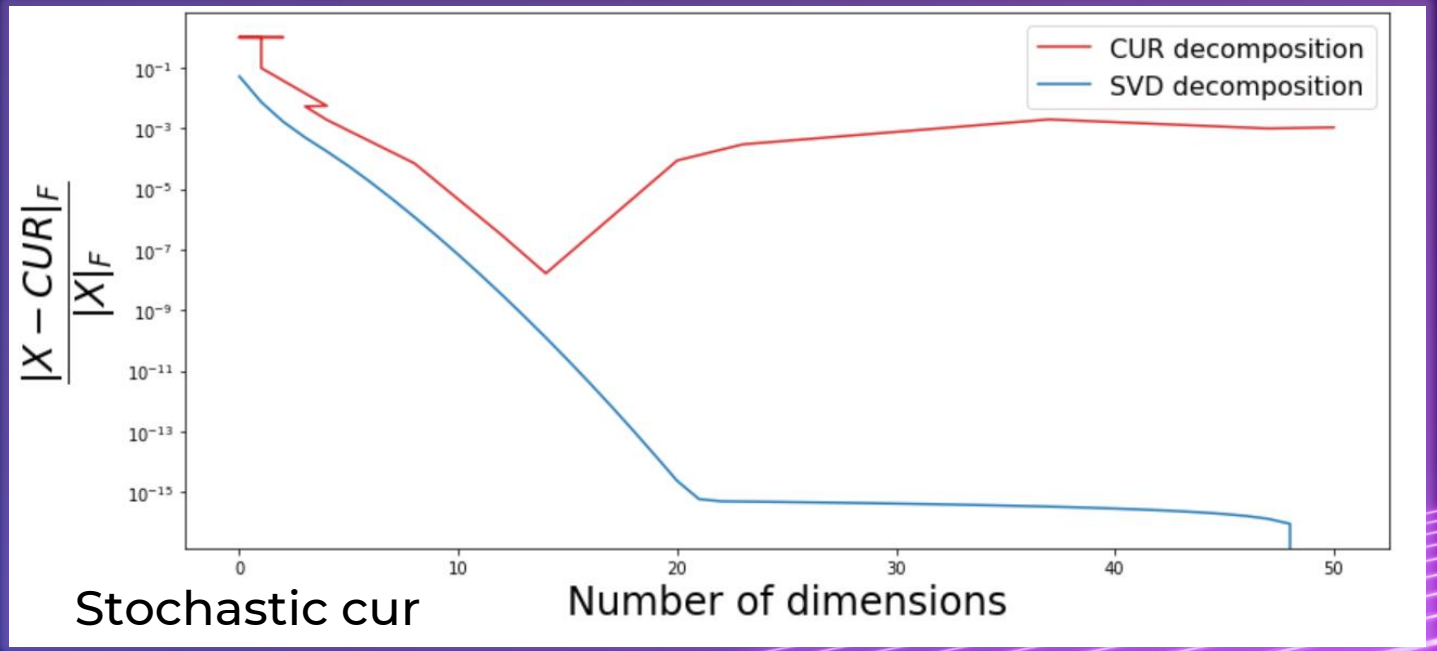


Stochastic vs Deterministic. RBF Matrix

```
#RBF matrix for R+
N = 50
xi = np.array([i/(N-1) for i in range(N)])
xj = np.array([i/(N-1) for i in range(N)])
xx, yy = np.meshgrid(xi, xj)
A_rbf = np.exp(-np.sqrt(xx**2 + yy**2))
```

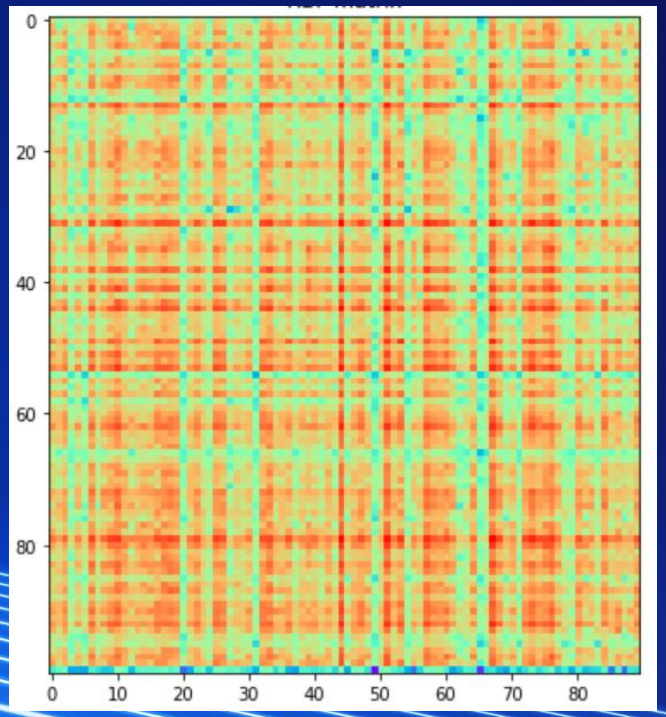


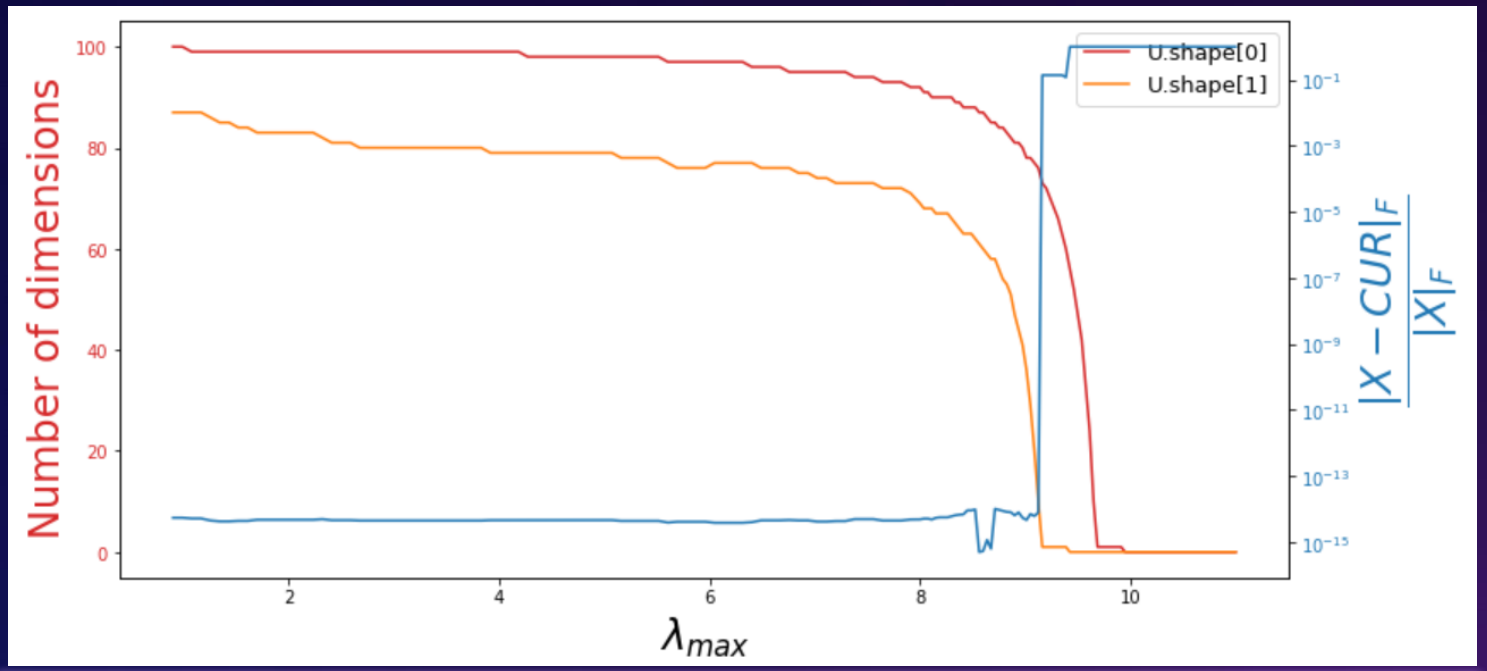


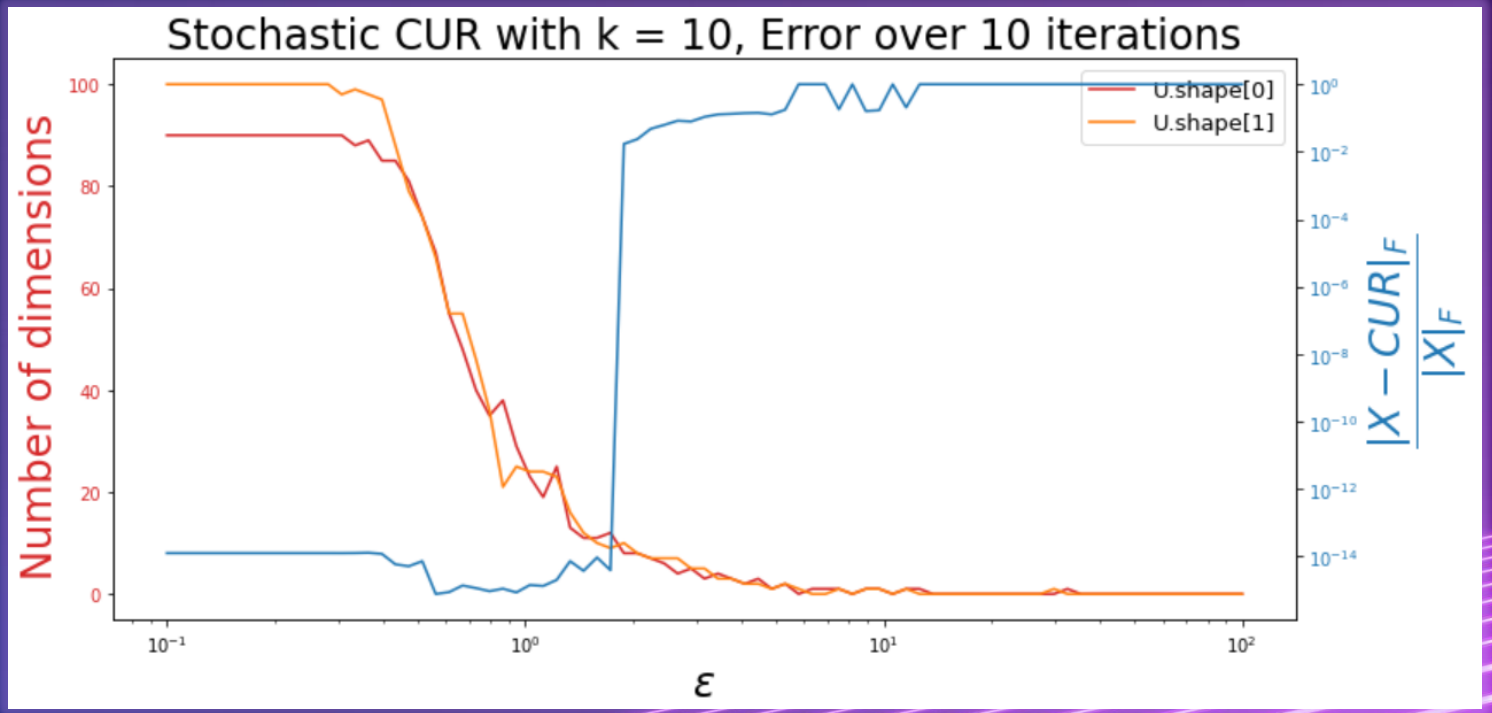


Stochastic vs Deterministic. Low-Rank Matrix (R = 10)

```
s1=100;
s2=90;
for j in range(10):
    a = j*np.random.rand(s1)
    b = np.random.rand(s2)
    Matrix_bra_ket =
Matrix_bra_ket+(a[:,None] * b)
```

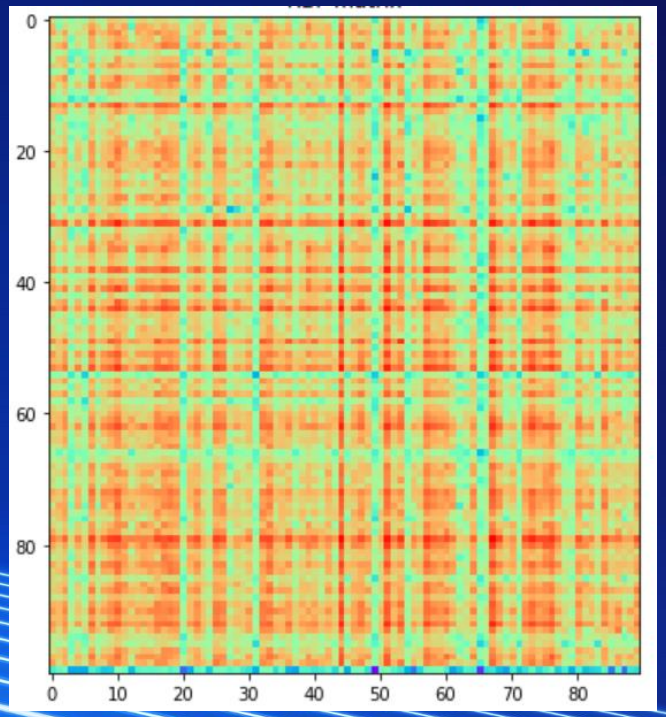


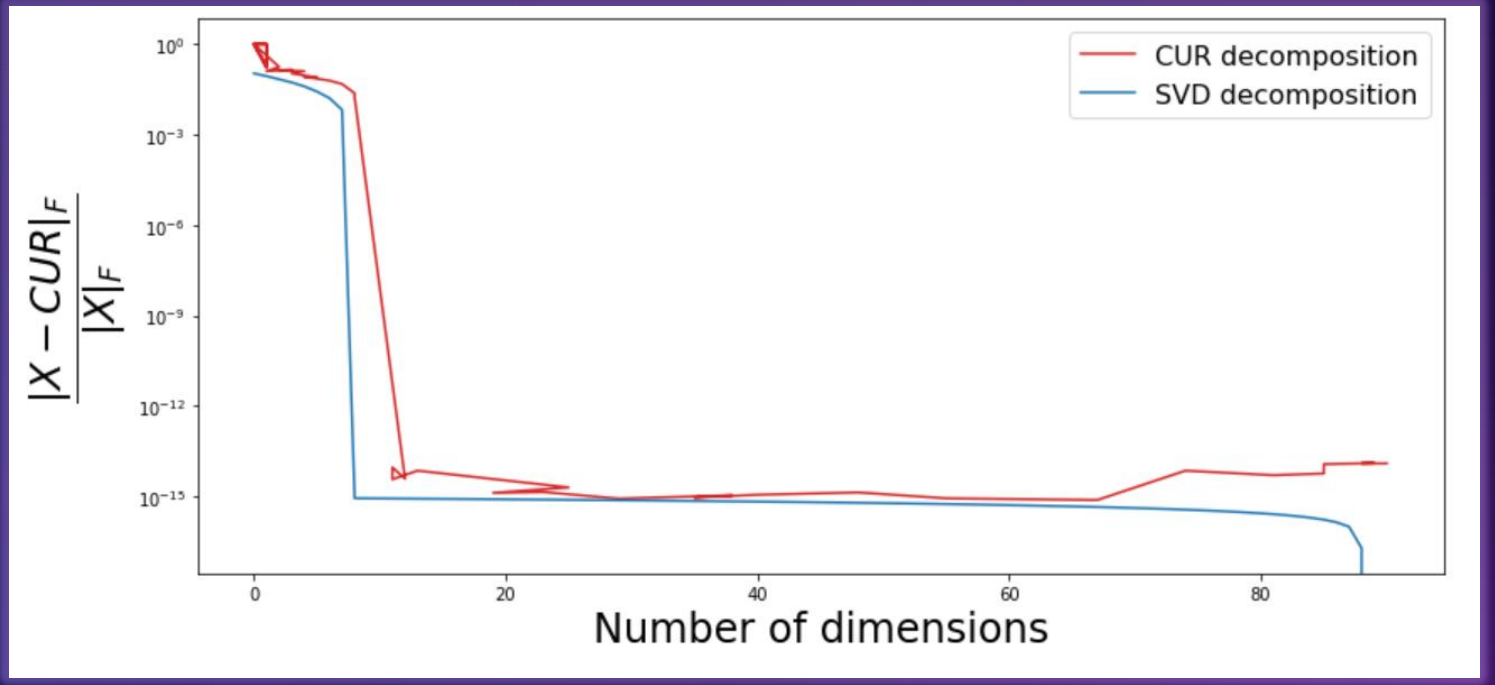


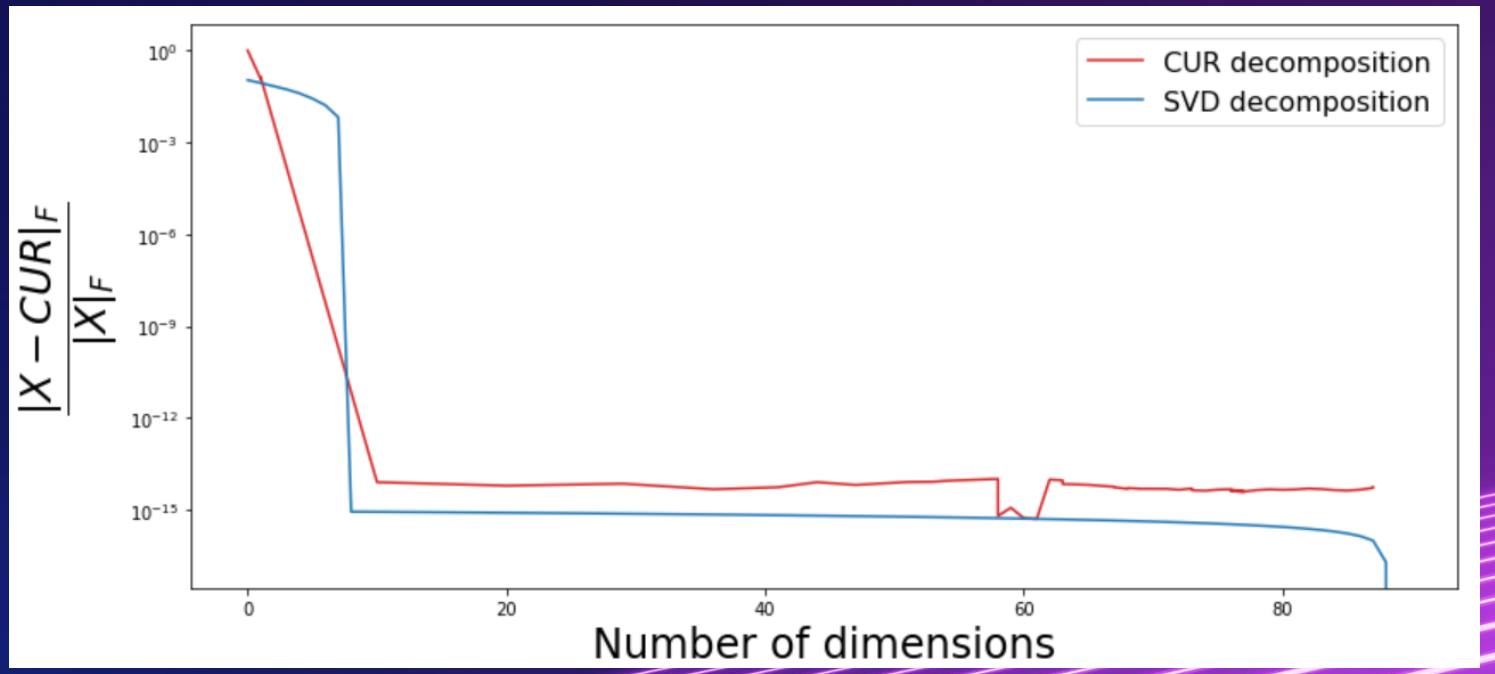


Stochastic vs Deterministic. Low-Rank Matrix (R = 10)

```
s1=100;
s2=90;
for j in range(10):
    a = j*np.random.rand(s1)
    b = np.random.rand(s2)
    Matrix_bra_ket =
Matrix_bra_ket+(a[:,None] * b)
```

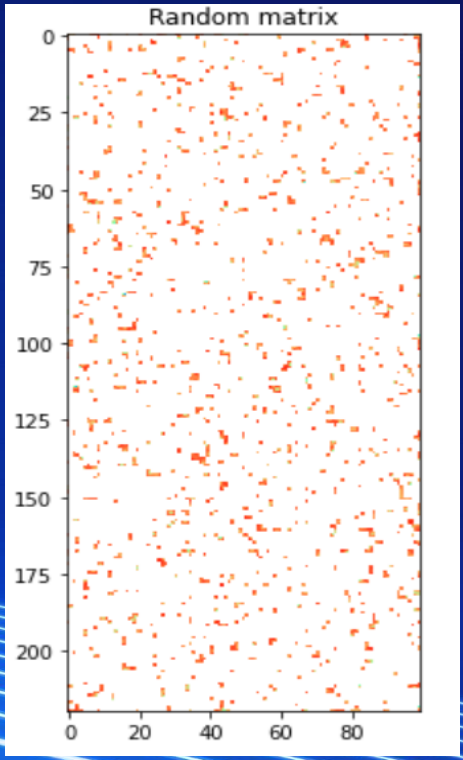


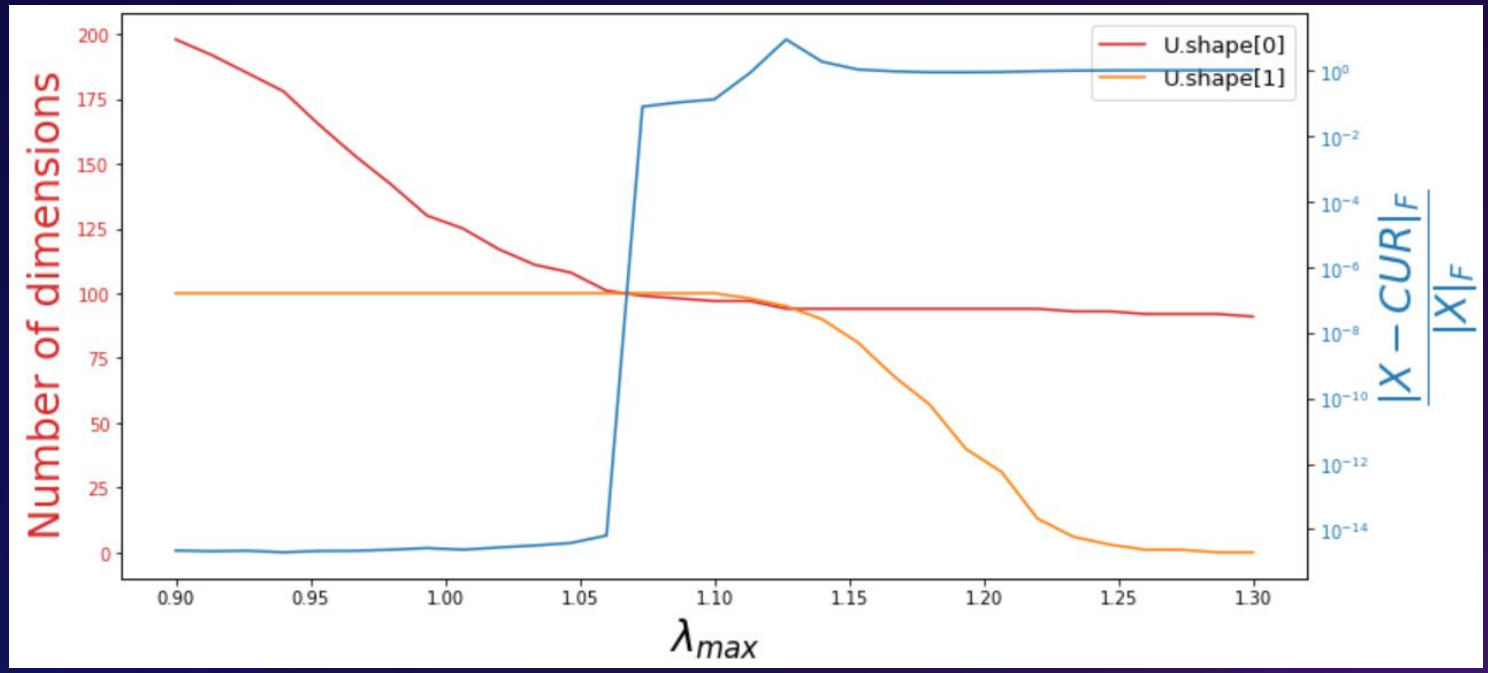


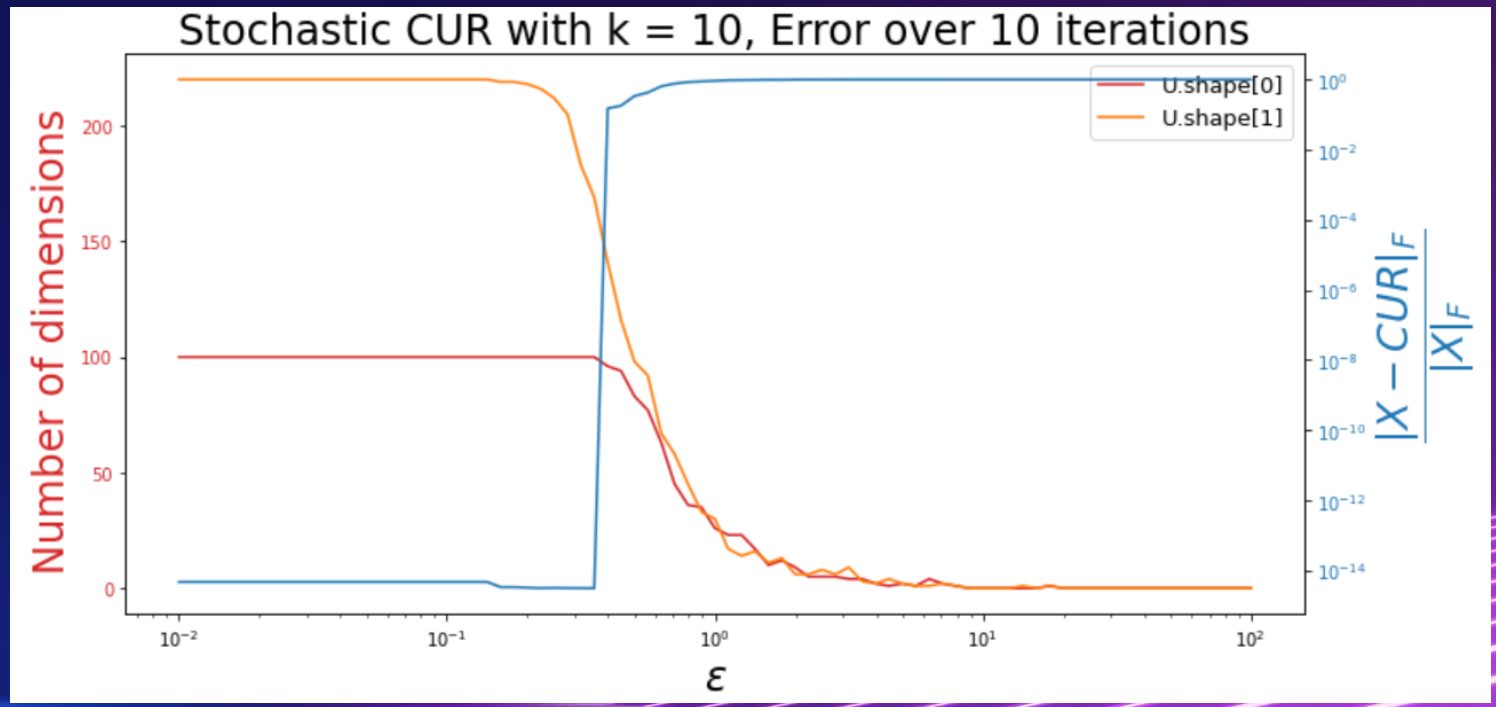


Stochastic vs Deterministic. Random Matrix

```
n, p = (220, 100)
X = np.random.randn(n,p)
```

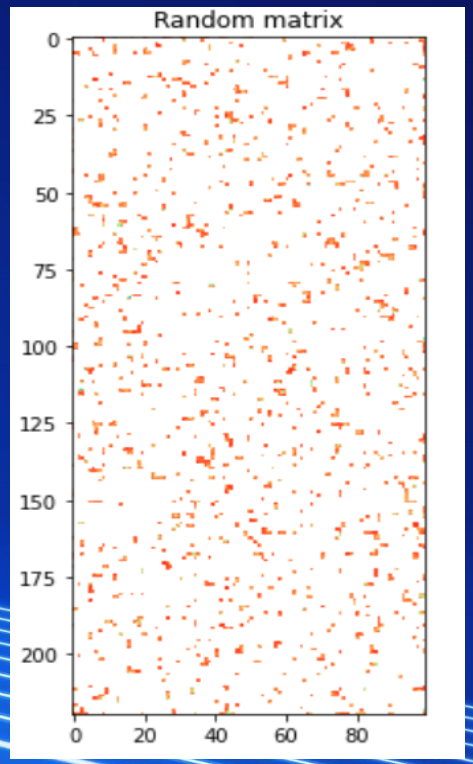


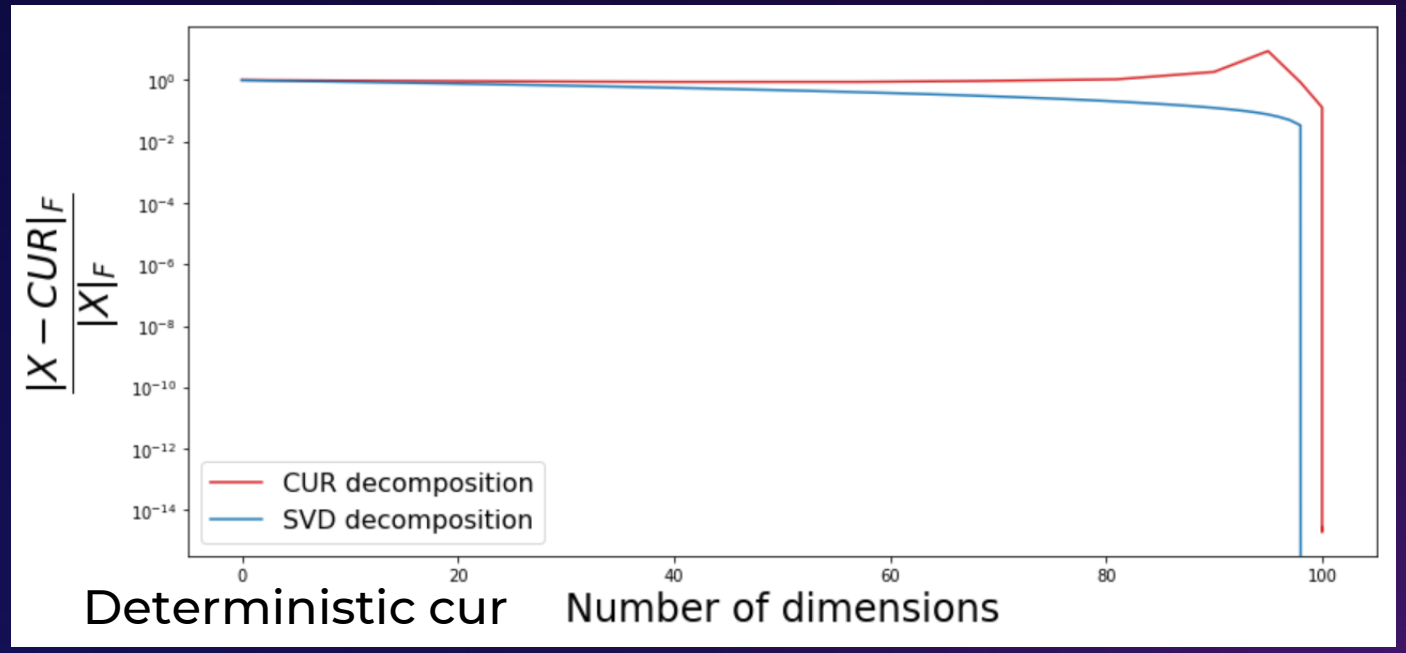


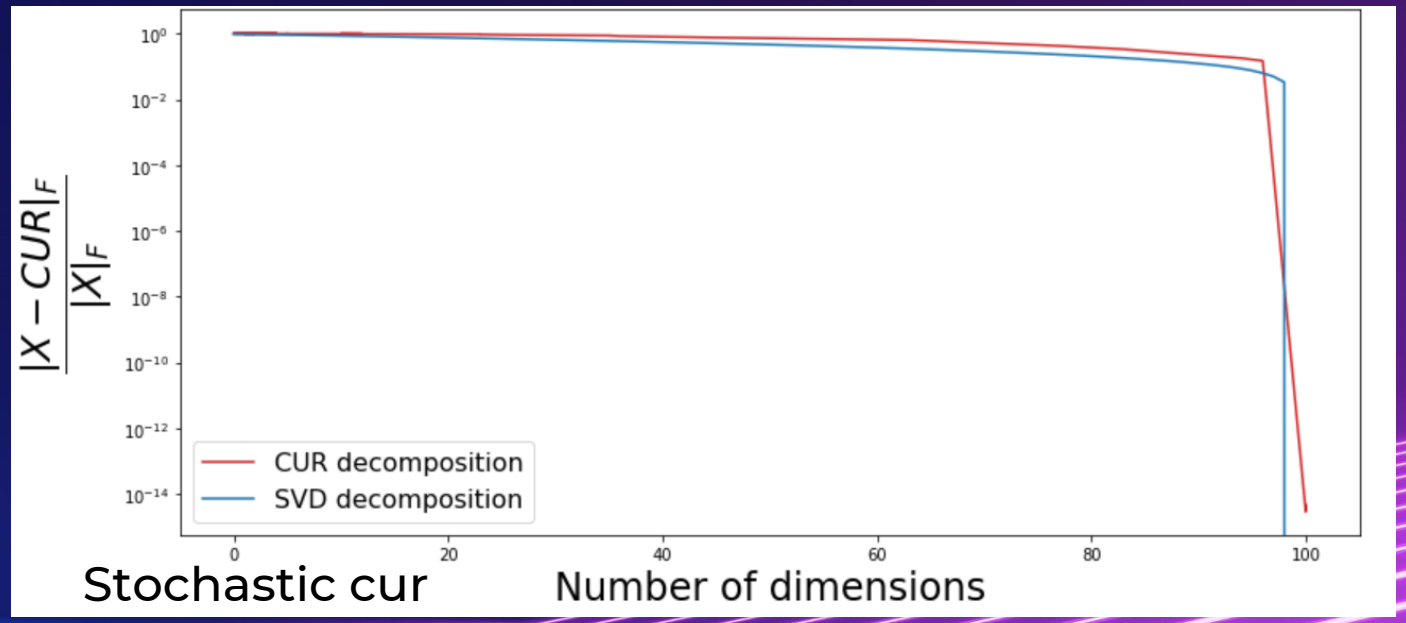


Stochastic vs Deterministic. Random Matrix

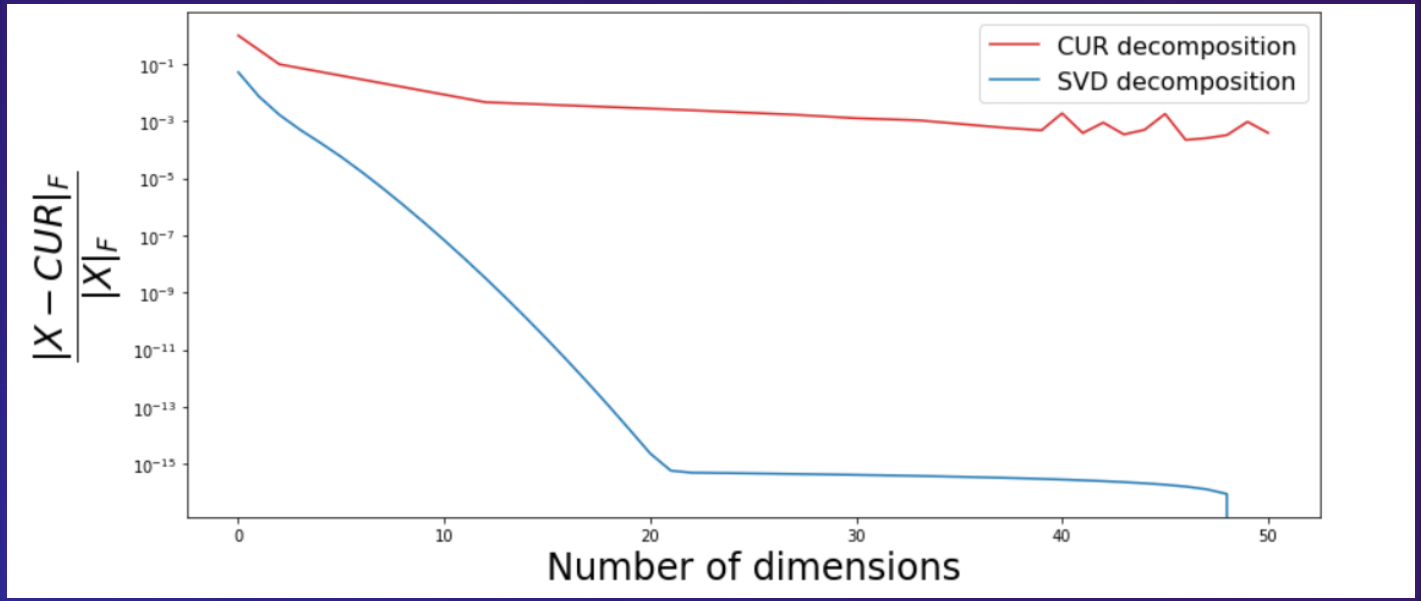
```
n, p = (220, 100)
X = np.random.randn(n,p)
```







How does it relate to SVD?



For RBF matrix

Same:

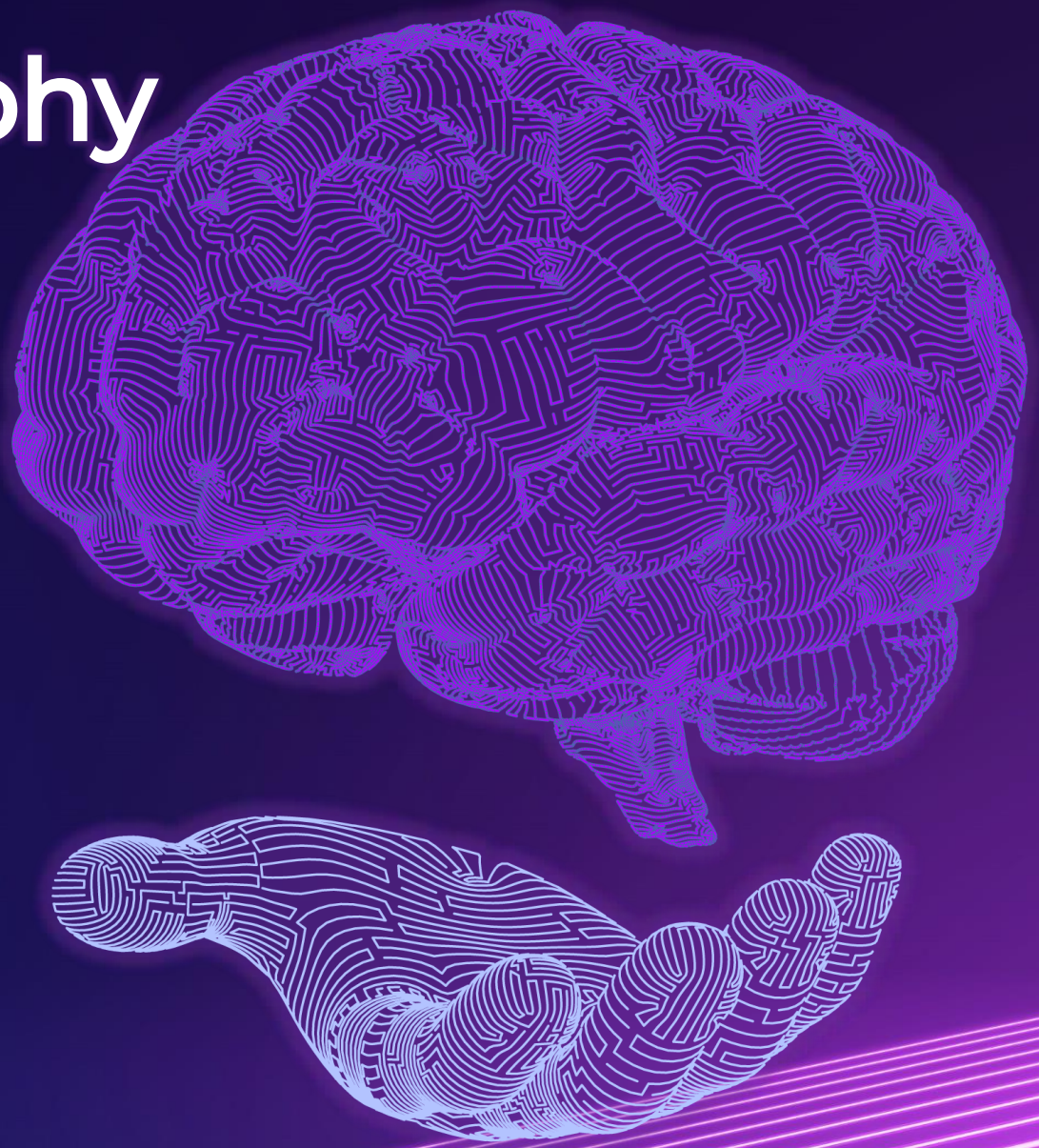
- Used both of them as low-rank approximation

Difference:

- CUR is highly interpretable
- C and R are not unitary
- U in CUR decomposition is a pseudoinverse matrix on the intersection
- Σ a real diagonal $k \times k$ matrix.

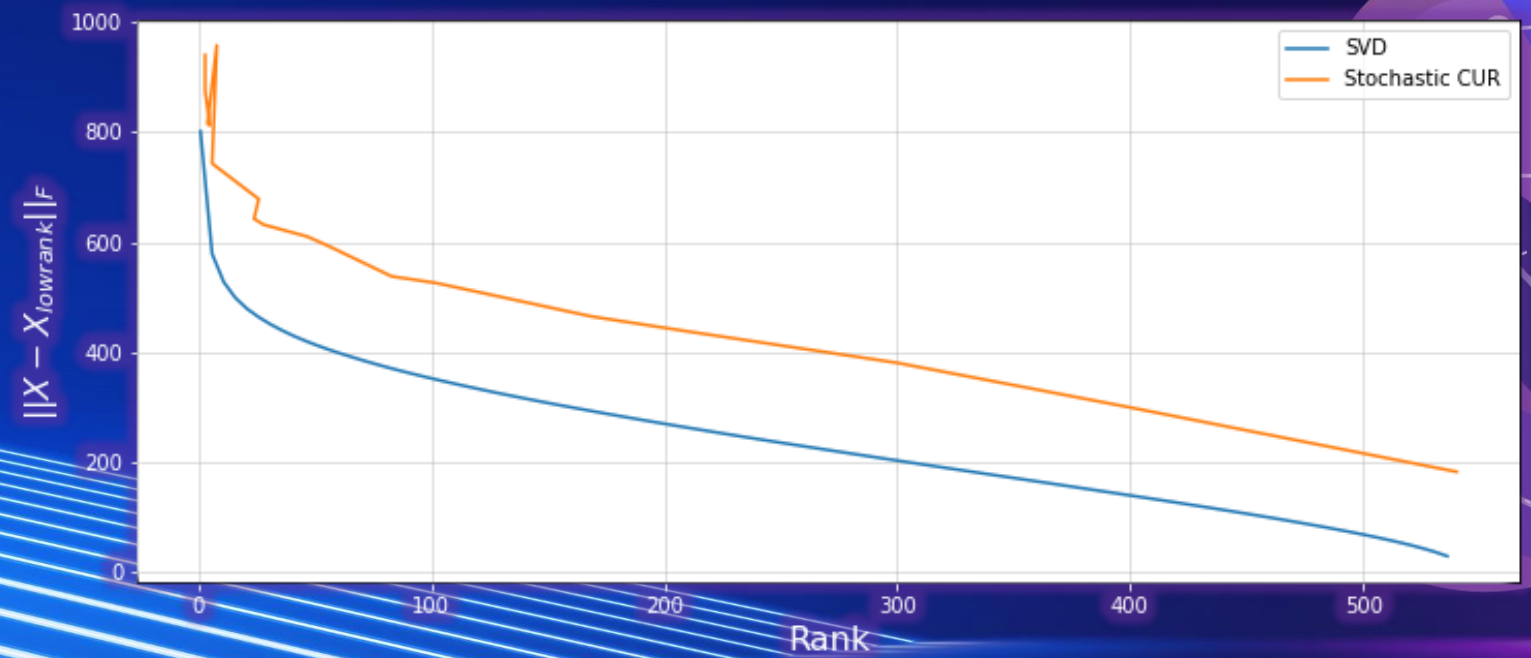
Application. Electroencephalography

- Some channels in EEG data have noise and disturbances, which will interfere with the interpretability of results
- We should pre-pre-process data using CUR
- Therefore, we can get rid of uncorrelated channels with noise



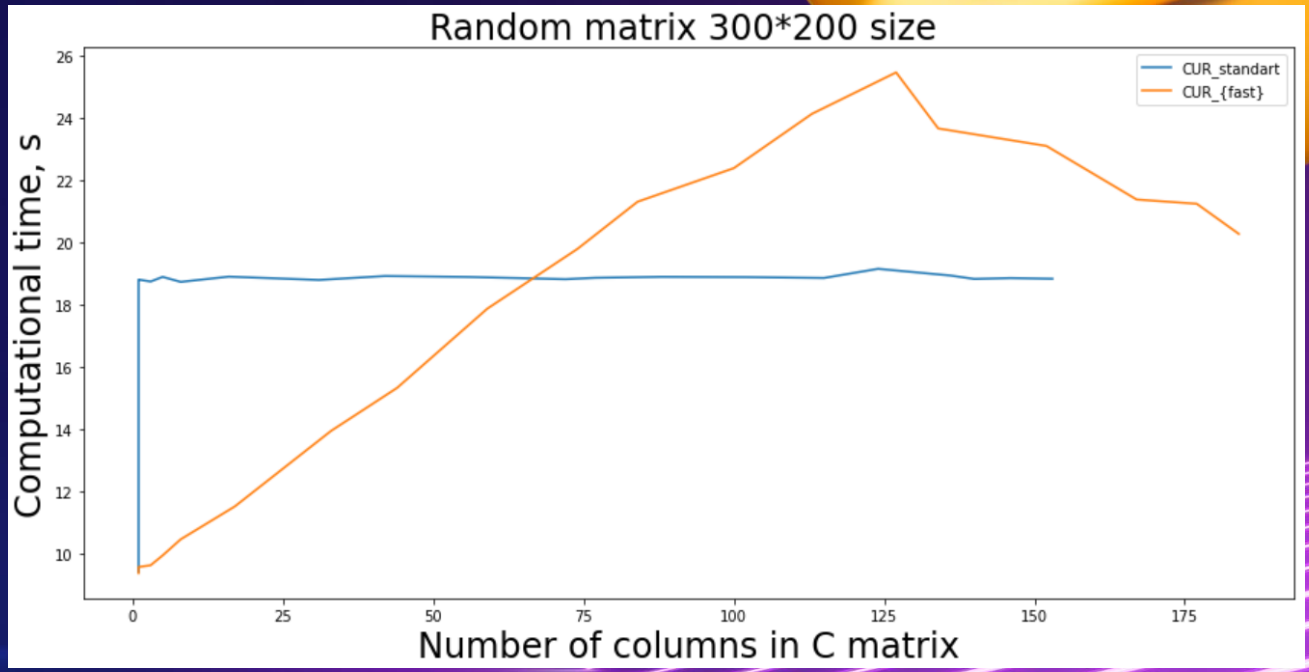
Application. Genes

- Dataset of Gene Expression for Kidney and Colon Cells
- 546 samples x 10’935 genes



So, should we use it? When?

- To sum up, practically Deterministic CUR should be used with caution, because of its high computational cost.
- Fast version of the algorithm improves the situation, but for particular conditions:
  - Target rank is sufficiently small
  - Matrix is *tall*
- It is fairly complicated to search for appropriate parameters for both Stochastic and Deterministic CUR



References

1. https://proceedings.icml.cc/paper/2020/file/fe70c36866add1572a8e2b96bfede7bf-Paper.pdf
2. https://link.springer.com/chapter/10.1007/978-981-10-6463-0_37
3. https://www.jstor.org/stable/40254734?seq=1
4. https://www.pnas.org/content/pnas/106/3/697.full.pdf
5. https://en.wikipedia.org/wiki/Singular_value_decomposition
6. https://en.wikipedia.org/wiki/CUR_matrix_approximation