Tensor Decomposition Based on Sketches

September 2, 2020

Abstract

SketchyCoreTucker/SketchyCoreTT for Tucker/Tensor Train (TT) decomposition.

1 SketchyCoreTucker

1.1 Algorithm

SkechySVD has been extended by [10] to compute the approximate low-rank Tucker decomposition of tensors. We will call this method SketchyTucker. Using the same idea we have for constructing approximation only from samples, we present our SketchyCoreTucker method. Without loss of generality, we assume $\mathcal{A} \in \mathbb{R}^{N_1 \times N_2 \times N_3}$. Its matricizations along the three dimensions are denoted by $\boldsymbol{A}_{(1)} \in \mathbb{R}^{N_1 \times (N_2 N_3)}$, $\boldsymbol{A}_{(2)} \in \mathbb{R}^{N_2 \times (N_3 N_1)}$, and $\boldsymbol{A}_{(3)} \in \mathbb{R}^{N_3 \times (N_1 N_2)}$, respectively. Define

$$\boldsymbol{r} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix}, \quad \boldsymbol{k} = \begin{bmatrix} k_1 \\ k_2 \\ k_3 \end{bmatrix}, \quad \boldsymbol{s} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix}, \quad \boldsymbol{m} = \begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix}, \quad \boldsymbol{n} = \begin{bmatrix} n_1 \\ n_2 \\ n_3 \end{bmatrix},$$

where $\boldsymbol{r} \preceq \boldsymbol{k} \preceq \boldsymbol{s} \preceq \min\{\boldsymbol{m}, \boldsymbol{n}\}$ and "$\preceq$" means "$\leq$" for each entry. Define

$$p_1 = \frac{m_1}{N_2 N_3} \in (0,1), \quad p_2 = \frac{m_2}{N_1 N_3} \in (0,1), \quad p_3 = \frac{m_3}{N_1 N_2} \in (0,1),$$

and

$$q_1 = \frac{n_1}{N_1} \in (0,1), \quad q_2 = \frac{n_2}{N_2} \in (0,1), \quad q_3 = \frac{n_3}{N_3} \in (0,1).$$

The main steps of SketchyCoreTucker are the following.

1. Building sketches.

   - Uniformly sample m_1 columns of $\boldsymbol{A}_{(1)}$. Denote the indices of the sampled columns to be Θ_1. The map $\boldsymbol{\Omega}_1 \in \mathbb{R}^{k_1 \times m_1}$ is applied to $\boldsymbol{A}_{(1)}^{(:,\Theta_1)}$ from the right,

   $$\boldsymbol{Y}_1 = \boldsymbol{A}_{(1)}^{(:,\Theta_1)} \boldsymbol{\Omega}_1^* \in \mathbb{R}^{N_1 \times k_1};$$

   - Uniformly sample m_2 columns of $\boldsymbol{A}_{(2)}$. Denote the indices of the sampled columns to be Θ_2. The map $\boldsymbol{\Omega}_2 \in \mathbb{R}^{k_2 \times m_2}$ is applied to $\boldsymbol{A}_{(2)}^{(:,\Theta_2)}$ from the right,

   $$\boldsymbol{Y}_2 = \boldsymbol{A}_{(2)}^{(:,\Theta_2)} \boldsymbol{\Omega}_2^* \in \mathbb{R}^{N_2 \times k_2};$$

- Uniformly sample m_3 columns of $\boldsymbol{A}_{(3)}$. Denote the indices of the sampled columns to be Θ_3. The map $\boldsymbol{\Omega}_3 \in \mathbb{R}^{k_3 \times m_3}$ is applied to $\boldsymbol{A}_{(3)}^{(:,\Theta_3)}$ from the right,

$$\boldsymbol{Y}_3 = \boldsymbol{A}_{(3)}^{(:,\Theta_3)} \boldsymbol{\Omega}_3^* \in \mathbb{R}^{N_3 \times k_3};$$

- Uniformly sample n_1, n_2, and n_3 rows of $\boldsymbol{A}_{(1)}$, $\boldsymbol{A}_{(2)}$, and $\boldsymbol{A}_{(3)}$. Denote the indices of the sampled rows to be Δ_1, Δ_2, and Δ_3, respectively. Apply maps $\boldsymbol{\Phi}_1 \in \mathbb{R}^{s_1 \times n_1}$, $\boldsymbol{\Phi}_2 \in \mathbb{R}^{s_2 \times n_2}$, and $\boldsymbol{\Phi}_3 \in \mathbb{R}^{s_3 \times n_3}$ to the intersection $\mathcal{A}^{(\Delta_1,\Delta_2,\Delta_3)}$,

$$\mathcal{Z} = \mathcal{A}^{(\Delta_1,\Delta_2,\Delta_3)} \times_1 \boldsymbol{\Phi}_1 \times_2 \boldsymbol{\Phi}_2 \times_3 \boldsymbol{\Phi}_3 \in \mathbb{R}^{s_1 \times s_2 \times s_3};$$

2. Computations.

- We compute the QR decomposition of $\boldsymbol{Y}_1$,

$$\boldsymbol{Y}_1 = \boldsymbol{Q}_1 \boldsymbol{R}_1,$$

  where $\boldsymbol{Q}_1 \in \mathbb{R}^{N_1 \times k_1}$, and $\boldsymbol{R}_1 \in \mathbb{R}^{k_1 \times k_1}$;

- We compute the QR decomposition of $\boldsymbol{Y}_2$,

$$\boldsymbol{Y}_2 = \boldsymbol{Q}_2 \boldsymbol{R}_2,$$

  where $\boldsymbol{Q}_2 \in \mathbb{R}^{N_2 \times k_2}$, and $\boldsymbol{R}_2 \in \mathbb{R}^{k_2 \times k_2}$;

- We compute the QR decomposition of $\boldsymbol{Y}_3$,

$$\boldsymbol{Y}_3 = \boldsymbol{Q}_3 \boldsymbol{R}_3,$$

  where $\boldsymbol{Q}_3 \in \mathbb{R}^{N_3 \times k_3}$, and $\boldsymbol{R}_3 \in \mathbb{R}^{k_3 \times k_3}$;

- We compute the core approximation

$$\mathcal{C} = \mathcal{Z} \times_1 (\boldsymbol{\Phi}_1 \boldsymbol{Q}_1^{(\Delta_1,:)})^\dagger \times_2 (\boldsymbol{\Phi}_2 \boldsymbol{Q}_2^{(\Delta_2,:)})^\dagger \times_3 (\boldsymbol{\Phi}_3 \boldsymbol{Q}_3^{(\Delta_3,:)})^\dagger \in \mathbb{R}^{k_1 \times k_2 \times k_3};$$

- The final near-optimal multi-linear $\boldsymbol{r}$ approximation to $\mathcal{A}$, denoted by $[[\mathcal{A}]]_{\boldsymbol{r}}$, is computed by

$$[[\mathcal{C}]]_{\boldsymbol{r}} \times_1 \boldsymbol{Q}_1 \times_2 \boldsymbol{Q}_2 \times_3 \boldsymbol{Q}_3,$$

  where $[[\mathcal{C}]]_{\boldsymbol{r}}$ is the best multi-linear rank $\boldsymbol{r}$ approximation of $\mathcal{C}$. Empirically, $[[\mathcal{C}]]_{\boldsymbol{r}}$ can be computed by algorithm such as HOOI [4].

1.2 Experiments

1.2.1 Datasets

The considered datasets, their dimensions, and proper choices of multilinear ranks, are presented in the following table. Hyperspectral datasets are much more low-rank in mode 3 than in mode 1 and 2. For the Video dataset, the spectrum in mode 1 and 2 decrease very slowly.

Table 1: Datasets, their dimensions, and proper choices of multilinear ranks.

Dataset	Dimensions	Multilinear Ranks
Indian Pines	$145 \times 145 \times 220$	$(20, 20, 16)$
Salinas	$512 \times 217 \times 224$	$(50, 30, 16)$
Pavia University	$610 \times 340 \times 103$	$(100, 80, 9)$
FTIR	$256 \times 256 \times 1506$	$(40, 40, 10)$
Cardiac	$256 \times 176 \times 160$	$(20, 20, 5)$
Navier Stokes	$50 \times 100 \times 200$	$(5, 10, 7)$
Video	$540 \times 960 \times 2200$	$(100, 200, 25)$

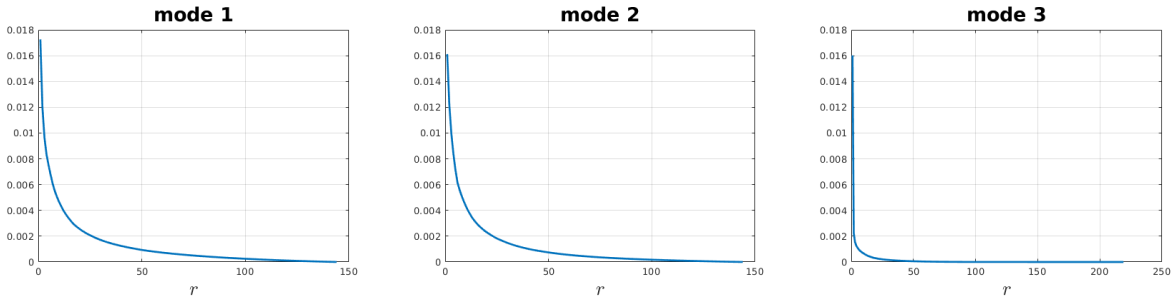

Figure 1: Scree plots for the modes of Indian Pines dataset.

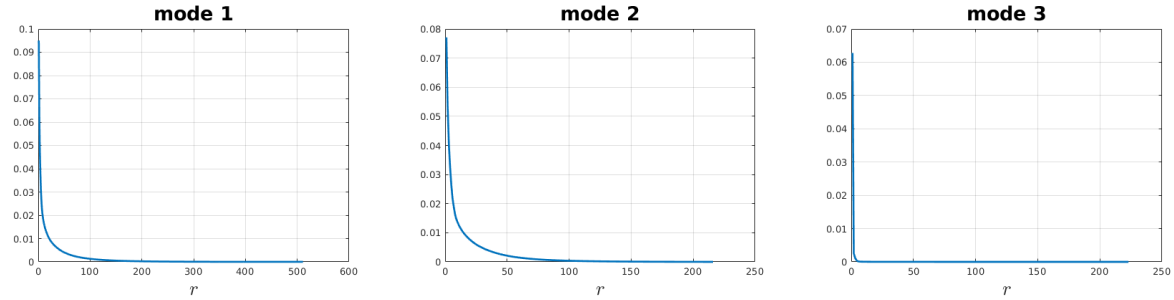

Figure 2: Scree plots for the modes of Salinas dataset.

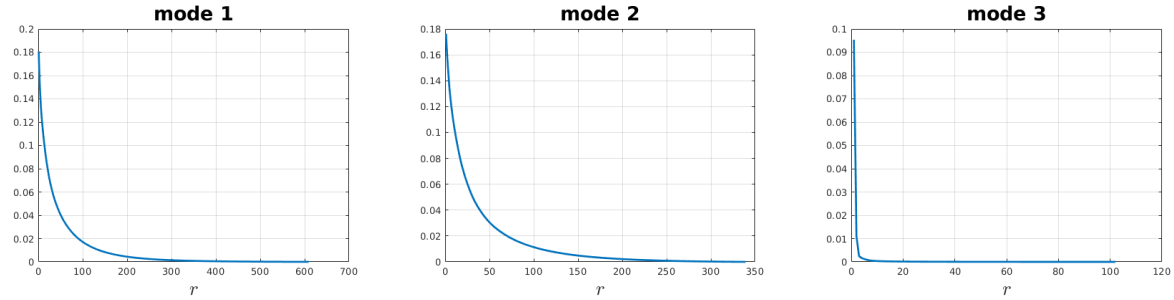

Figure 3: Scree plots for the modes of Pavia University dataset.

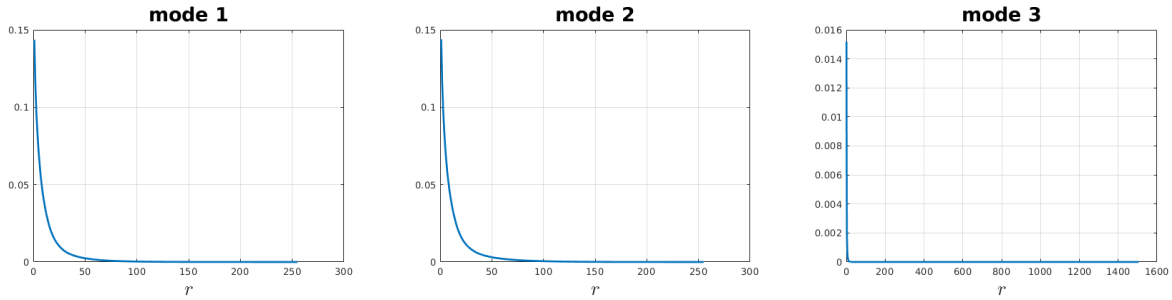

Figure 4: Scree plots for the modes of FTIR dataset.

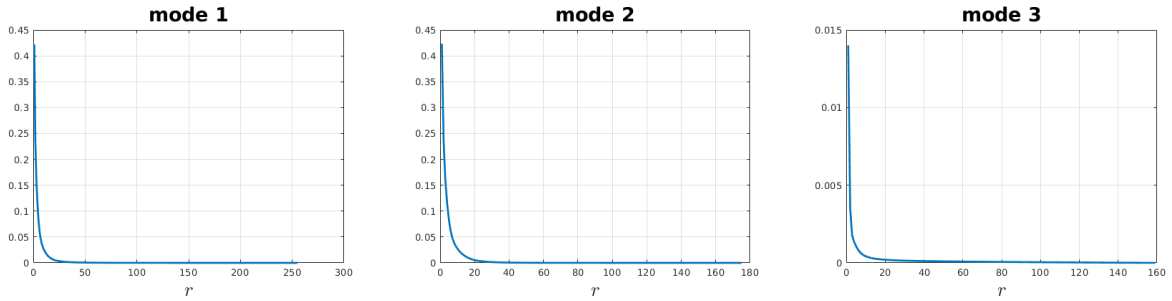

Figure 5: Scree plots for the modes of Cardiac dataset.

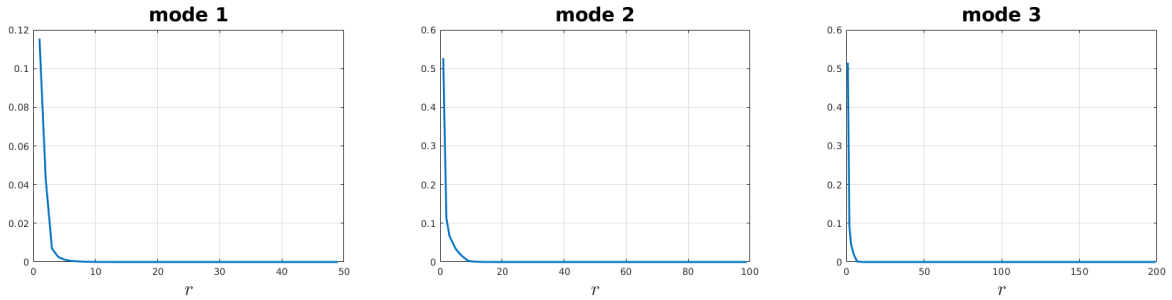

Figure 6: Scree plots for the modes of Navier Stokes dataset.

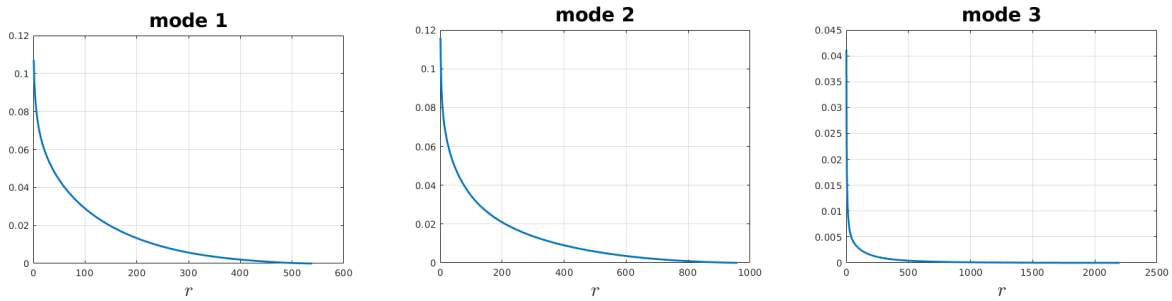

Figure 7: Scree plots for the modes of Video dataset.

1.2.2 Performances

The multilinear rank approximation computed by the higher order orthogonal iteration (HOOI) [4] is treated as ground truth. There are no strong guarantees on the performance of HOOI; however, it is widely believed to produce an approximation that is usually quite close to the best multilinear rank approximation. For SketchyTucker and SketchyCoreTucker, we set $k_1 = 2r_1 + 1$, $k_2 = 2r_2 + 1$, and $k_3 = 4r_3 + 1$. Also, $s_1 = 2k_1 + 1$, $s_2 = 2k_2 + 1$, and $s_3 = 2k_3 + 1$. For SketchyCoreTucker, we set $p_1 = p_2 = p_3 = p$, $q_1 = q_2 = 1$, and $q_3 = q$.

From the tables, we see that SketchyTucker is faster than HOOI, but the approximation error is in general 2 or 3 times larger except for the Navier Stokes dataset, where the approximation error is near optimal. We also find SkethcyCoreTucker is able to reduce the computation cost without sacrificing approximation accuracy of SkethcyTucker.

Table 2: Approximation errors of HOOI and SketchyTucker.

	Indian Pines	Salinas	Pavia University	FTIR	Cardiac	Navier Stokers	Video
HOOI	0.0032	0.0054	0.0219	0.0058	0.0076	0.0033	0.0344
SketchyTucker	0.0084	0.0138	0.0508	0.0126	0.0153	0.0036	0.0762

Table 3: Computation time of HOOI and SketchyTucker.

	Indian Pines	Salinas	Pavia University	FTIR	Cardiac	Navier Stokers	Video
HOOI	0.2012	2.6336	2.6712	11.3116	0.4544	0.0335	89.5288
SketchyTucker	0.1396	0.6959	1.0373	1.7271	0.1272	0.0268	26.8015

Table 4: Performances of SketchyCoreTucker on FTIR dataset.

$q = 0.06$	$p = 0.002$	$p = 0.004$	$p = 0.006$
err	0.0130	0.0129	0.0127
time	0.6372	0.6425	0.6510

Table 5: Performances of SketchyCoreTucker on Video dataset.

$q = 0.2$	$p = 0.06$	$p = 0.08$	$p = 0.1$
err	0.0776	0.0773	0.0771
time	9.9597	10.0304	10.3523

Remark. *SketchyTucker [10] claims that [7] is the only one-pass Tucker approximation algorithm before itself. SketchyCoreTucker seems to further improve upon SketchyTucker. However, it is unclear what could be the contribution of SketchyCoreTucker, given the vast body of literature on Tucker approximation [1].*

2 SketchyCoreTT

2.1 Algorithms

2.1.1 Tensor Train Decomposition

Tensor Train decomposition [9] is especially suitable for computing the decomposition of higher order tensors. Without loss of generality, we describe the algorithm for third order tensors.

Algorithm 1 Tensor Train Decomposition

Input: $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$.

Output: Tensor $\mathcal{B}$ in TT format with cores $\boldsymbol{G}_1 \in \mathbb{R}^{n_1 \times r_1}$, $\mathcal{G}_2 \in \mathbb{R}^{r_1 \times n_2 \times r_2}$, $\boldsymbol{G}_3 \in \mathbb{R}^{r_2 \times n_3}$.

1: Initialize $\mathcal{C} = \mathcal{A}$.
2: Set $\boldsymbol{C} = \text{reshape}(\mathcal{C}, n_1, n_2 n_3)$;
3: Compute truncated SVD of $\boldsymbol{C}$ with rank r_1, denoted by $\boldsymbol{U}_1 \boldsymbol{\Sigma}_1 \boldsymbol{V}_1^*$;
4: Set $\boldsymbol{G}_1 = \text{reshape}(\boldsymbol{U}_1, n_1, r_1)$;
5: Update $\boldsymbol{C} = \boldsymbol{\Sigma}_1 \boldsymbol{V}_1^*$;
6: Set $\boldsymbol{C} = \text{reshape}(\boldsymbol{C}, r_1 n_2, n_3)$;
7: Compute truncated SVD of $\boldsymbol{C}$ with rank r_2, denoted by $\boldsymbol{U}_2 \boldsymbol{\Sigma}_2 \boldsymbol{V}_2^*$;
8: Set $\mathcal{G}_2 = \text{reshape}(\boldsymbol{U}_2, r_1, n_2, r_2)$;
9: Set $\boldsymbol{G}_3 = \boldsymbol{\Sigma}_2 \boldsymbol{V}_2^*$.

2.1.2 SketchyTT

SketchyTT computes the truncated SVDs in Step 3 and Step 7 using SketchySVD [11], a randomized SVD algorithm based on building sketches. There are extra parameters (k_1, s_1, k_2, s_2). Usually, $s_1 = 2k_1 + 1$ and $s_2 = 2k_2 + 1$. Whenever possible, we choose $k_1 = 4r_1 + 1$ and $k_2 = 4r_2 + 1$.

2.1.3 SketchyCoreTT

SketchyCoreTT computes the truncated SVDs in Step 3 and Step 7 using SketchyCoreSVD [2], an algorithm combines SketchySVD with random sampling. Besides (k_1, s_1, k_2, s_2), there are two extra parameters (p_1, p_2) about sampling ratios.

2.1.4 CUR Based TT

Tensor train can be used to compute tensor DMD [6]. Recently, [8] combines EDMD with TT, which is computed using sequential CUR adopting a maximum-volume principle when selecting the row/column indices. Thus we would like to compare methods based on sketching and methods based on sampling to compute the TT. For simplicity, we first compare with the following two variants of CUR.

- The first method is based on row/column lengths when selecting the indices, an extension of [5] to the asymmetric case. **Like SketchySVD, it needs to access all the data.** The resulting algorithm is called TT-CUR$_1$;
- The second method [3] samples the rows/columns without replacement. **Like SketchyCoreSVD, it does not need to access all the data.** The resulting algorithm is called TT-CUR$_2$.

2.2 Experiments

The approximation error is defined as

$$err = \frac{\|\mathcal{B} - \mathcal{A}\|_F^2}{\|\mathcal{A}\|_F^2}.$$

The recorded times are in seconds. The times reported are averaged over 10 random trails. The algorithms in Section 2.1 are first compared on the Video dataset, which is a video of 2200 frames, and of size 540×960 in each frame. We reshape the video such that $\mathcal{A} \in \mathbb{R}^{2200 \times 540 \times 960}$. We choose $r_1 = 50$ and $r_2 = 100$. We use the same (s_1, s_2) for the number of sampled rows/columns for the two methods based on CUR.

Table 6: Performance comparisons for Video dataset.

	TT	TT-CUR$_1$	TT-CUR$_2$	SketchyTT	SketchyCoreTT			
(p_1, p_2)	-	-	-	-	$(0.2, 0.85)$	$(0.25, 0.85)$	$(0.2, 0.9)$	$(0.25, 0.9)$
err	0.0356	0.0487	0.0705	0.0655	0.0742	0.0722	0.0723	0.0704
time	169.46	5.6114	1.0806	18.310	9.8882	10.957	9.9811	10.885

Based on the table, there is really no advantages of the methods based on sketches. TT-CUR$_2$ is a really simple algorithm, and it can yield the same accuracy of SketchyCoreTT with much less time.

Remark. *We cannot say CUR based approaches are always better, but I am not sure if it's meaningful to proceed with SketchyCoreTT.*

References

[1] Salman Ahmadi-Asl, Andrzej Cichocki, Anh Huy Phan, Ivan Oseledets, Stanislav Abukhovich, and Toshihisa Tanaka. Randomized algorithms for computation of Tucker decomposition and Higher Order SVD (HOSVD). *arXiv preprint arXiv:2001.07124*, 2020.

[2] Chandrajit Bajaj, Yi Wang, and Tianming Wang. SketchyCoreSVD: SketchySVD from random subsampling of the data matrix. *arXiv preprint arXiv:1907.13634*, 2019.

[3] Jiawei Chiu and Laurent Demanet. Sublinear randomized algorithms for skeleton decompositions. *SIAM Journal on Matrix Analysis and Applications*, 34(3):1361–1383, 2013.

[4] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. A multilinear singular value decomposition. *SIAM journal on Matrix Analysis and Applications*, 21(4):1253–1278, 2000.

[5] Petros Drineas and Michael W Mahoney. On the Nyström method for approximating a Gram matrix for improved kernel-based learning. *journal of machine learning research*, 6(Dec):2153–2175, 2005.

[6] Stefan Klus, Patrick Gelß, Sebastian Peitz, and Christof Schütte. Tensor-based dynamic mode decomposition. *Nonlinearity*, 31(7):3359, 2018.

[7] Osman Asif Malik and Stephen Becker. Low-rank Tucker decomposition of large tensors using TensorSketch. In *Advances in Neural Information Processing Systems*, pages 10096–10106, 2018.

[8] Feliks Nüske, Patrick Gelß, Stefan Klus, and Cecilia Clementi. Tensor-based EDMD for the Koopman analysis of high-dimensional systems. *arXiv preprint arXiv:1908.04741*, 2019.

[9] Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

[10] Yiming Sun, Yang Guo, Charlene Luo, Joel Tropp, and Madeleine Udell. Low-rank Tucker approximation of a tensor from streaming data. *arXiv preprint arXiv:1904.10951*, 2019.

[11] Joel A Tropp, Alp Yurtsever, Madeleine Udell, and Volkan Cevher. Streaming low-rank matrix approximation with an application to scientific simulation. *arXiv preprint arXiv:1902.08651*, 2019.