Approximating Element-Wise Functions of Matrix with Improved Streaming Randomized SVD

Yuyang Xie, Xu Feng, Xizhi Zhang, Jiezhong Qiu, Wenjian Yu*
Dept. Computer Science & Tech., BNRist, Tsinghua University, Beijing, China.
{xyy18, fx17, zhangxz18, qiujz16}@mails.tsinghua.edu.cn, yu-wj@tsinghua.edu.cn

***Abstract*—The element-wise functions of a matrix are widely used in machine learning. For the applications with large matrices, efficiently computing the matrix-vector multiplication of matrix element-wise function without explicitly constructed matrix is very desired. In this work, we aim to develop an efficient low-rank approximation of the element-wise function of matrix with the time/memory cost linear to the matrix dimension. We first propose a sparse-sign streaming randomized SVD (ssrSVD) algorithm based on a streaming singular value decomposition (SVD) algorithm and the sparse-sign random projection for the approximation of element-wise function of general asymmetric matrix. For symmetric positive semi-definite (SPSD) matrix, for which the existing Nyström [1] and FastSPSD [2] method do not perform well if the matrix's singular value decays slowly, we propose a theoretically proved shift skill to improve the approximation accuracy. Combining with the ssrSVD, we obtain the sparse-sign streaming SPSD matrix approximation with shift (S3SPSD) algorithm. Experiments are carried out to evaluate the proposed algorithms' performance in approximating element-wise functions of matrix. With the color transfer task based on the Sinkhorn algorithm, the ssrSVD algorithm largely reduces the approximation error (up to $10^5\times$) compared with the state-of-the-art baselines, and results in high-quality color transfer result. For the kernel matrix approximation, the proposed S3SPSD algorithm also consistently outperforms the state-of-the-art baselines. Experimental results finally validate the linear time complexity of the proposed algorithms.**

***Index Terms*—element-wise function of matrix, randomized matrix approximation, sparse-sign random projection, streaming algorithm, shift skill**

I. Introduction

In machine learning applications, the element-wise function of a matrix is widely used. Examples include the ReLU activation [3] in deep neural networks—$\mathrm{ReLU}(\cdot)$, the attention mechanism in Transformer [4]—$\mathrm{softmax}(\cdot)$, and the kernel functions in kernel learning [5]. An element-wise function of matrix can be denoted by $\boldsymbol{A} = f^\circ(\boldsymbol{M}) \in \mathbb{R}^{m\times n}$, where $f^\circ(\cdot)$ is the element-wise function such as $\mathrm{ReLU}(\cdot)$, $\mathrm{softmax}(\cdot)$, and $\exp^\circ(\cdot)$ in radial basis function (RBF) [6], etc. Notice that $\boldsymbol{M}$ and $f^\circ(\boldsymbol{M})$ can be of large dimensions (m and n) so that they are not allowed to be explicitly constructed. Therefore, how to efficiently compute the multiplication of $f^\circ(\boldsymbol{M})$ and vector without the explicitly constructed $\boldsymbol{M}$ and $f^\circ(\boldsymbol{M})$ is of great interest.

The element-wise function of matrix can be a symmetric matrix or an asymmetric matrix. In the kernel learning, such as kernel SVM, the matrix $f^\circ(\boldsymbol{M})$ is a symmetric positive semidefinite (SPSD) matrix. The asymmetric matrix $f^\circ(\boldsymbol{M})$ also exists. An example is the computation of optimal transport distance with the Sinkhorn algorithm [7]–[9], e.g. in the color transfer task [10], where the matrix-vector multiplication of asymmetric matrix $f^\circ(\boldsymbol{M})$ is repeatedly computed.

Over the past few years, there have been some tries to approximate $f^\circ(\boldsymbol{M})$ by finding a low-rank approximation $f^\circ(\boldsymbol{M}) \approx \boldsymbol{U}\boldsymbol{V}^\top$, where $\boldsymbol{U} \in \mathbb{R}^{m\times r}$ and $\boldsymbol{V} \in \mathbb{R}^{n\times r}$. The time/memory costs for the matrix-vector multiplication of $f^\circ(\boldsymbol{M})$ can thus be reduced. The related work about the approximation of the element-wise functions of a matrix can be divided into three categories. The first category is random Fourier feature (RFF) methods, including [11]–[13]. These methods are based on Monte-Carlo sampling. The second category is tensor sketch methods, including [14], [15]. Poly-TensorSketch [15] is one of the recently proposed methods. It only works if $\boldsymbol{M}$ is equal to the multiplication of two low-rank matrices. The third category is matrix factorization methods, mainly designed for the SPSD kernel matrix. Williams and Drineas et al. [1], [16] introduced a Nyström method to approximate the SPSD kernel matrix. Wang et al. [2] proposed an effective matrix approximation method and found it to be a generalized form of the Nyström method. Our work follows the third category.

Randomized matrix computation [17] has gained significant popularity in low-rank approximation of large-scale matrix in the past decade. Halko, Tropp, and Martinsson [17], [18] presented good surveys of techniques for computing a randomized singular value decomposition (SVD)/eigendecomposition. A randomized Nyström approximation method is also proposed in [19]. However, the $O(n^2)$ time/memory costs limit its widespread use. Recently, several randomized SVD designed for streaming data [20], [21] have been studied. These techniques are general and may be beneficial for the low-rank approximation of the element-wise functions of a matrix.

Motivation. For the low-rank approximation of the element-wise functions of an asymmetric matrix $f^\circ(\boldsymbol{M})$, there are few of work, and none is based on the randomized SVD [17]. As for the SPSD kernel matrix approximation, more work has been proposed. Though the Nyström method with uniform column sampling is widely used, it sacrifices accuracy for efficiency. In contrast, Gaussian random projection within randomized SVD [17] is more accurate, but incurs time/memory costs of $O(n^2)$. Is there a technique that balances efficiency and effectiveness between uniform sampling and Gaussian

This work is supported by NSFC under grant No. 61872206.

random projection? As demonstrated in [22], the Nyström method performs better than the RFF method when the input SPSD kernel matrix is low-rank. If the SPSD kernel matrix is not low-rank or with slowly-decayed singular values, how can we improve the accuracy of existing methods?

Contribution. In this paper, we aim to derive an effective low-rank approximation of element-wise functions of a matrix with only $O(m+n)$ time/memory costs. Firstly, we propose a sparse-sign streaming randomized SVD (ssrSVD) algorithm for the element-wise functions of asymmetric matrix. It combines a streaming randomized SVD with sparse-sign random projection [18], which performs similar to Gaussian random projection, but ensures that the time/memory costs are linear to the matrix dimension. Secondly, we propose a shift skill to improve the approximation of the SPSD kernel matrix with slowly decaying singular values. Theoretical analysis is performed to reveal the rationality of the shift skill, with which we further propose a sparse-sign streaming SPSD matrix approximation with shift (S3SPSD) algorithm. With the color transfer task based on the Sinkhorn algorithm, where the related matrix is the element-wise functions of an asymmetric matrix, we show empirically that the ssrSVD is up to $10^5\times$ better than the state-of-the-art baselines in terms of approximation error. An experiment on SPSD kernel matrix approximation with eight datasets shows that the performance of S3SPSD algorithm is significantly better than the baselines. Regarding relative approximation error, it achieves an average of 86.29%, 83.75%, and 51.98% reduction over RFF, Nyström, and FastSPSD for the RBF kernel computation on dataset a9a. The code and test data are publicly available[1].

II. Preliminary

Suppose the element-wise function of matrix is $\boldsymbol{A} = f^{\circ}(\boldsymbol{M})$. $\boldsymbol{M}$ is always computed as $\boldsymbol{M} = \boldsymbol{L}\boldsymbol{R}^{\top}$ or $\boldsymbol{M}(i,j) = \|\boldsymbol{L}(i,:) - \boldsymbol{R}(j,:)\|_2^2$ or other operations between $\boldsymbol{L}$ and $\boldsymbol{R}$, where $\boldsymbol{L} \in \mathbb{R}^{m\times d}$ and $\boldsymbol{R} \in \mathbb{R}^{n\times d}$. The singular value decomposition (SVD) of $f^{\circ}(\boldsymbol{M})$ can be stated as:

$$\boldsymbol{A} = f^{\circ}(\boldsymbol{M}) = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\top}, \tag{1}$$

where $\boldsymbol{\Sigma}$ is the diagonal matrix includes the singular values σ_i of $\boldsymbol{A}$. When $\boldsymbol{A}$ is a SPSD matrix, the eigendecomposition and SVD are the same [23]. In addition, $\boldsymbol{A}$'s Moore−Penrose inverse is given by $\boldsymbol{A}^{\dagger} = \boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{U}^{\top}$. For a dense matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, the time complexity of computing full SVD is $O(\min(m^2n, mn^2))$.

The Basic Randomized SVD: We first review the idea of basic randomized SVD [17], [18]. It is based on the fact that random projection, such as Gaussian random projection, can capture the dominant information of the input matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$. Then, a sketch matrix $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{C}$ is obtained, where $\boldsymbol{C} \in \mathbb{R}^{n\times c}$ is a Gaussian random matrix. A matrix $\boldsymbol{Q} = \mathrm{orth}(\boldsymbol{Y})$ is then derived, where $\mathrm{orth}(\cdot)$ is an orthonormalization operation, such as QR factorization. Finally, we can obtain an approximate formula as $\boldsymbol{A} \approx \boldsymbol{Q}\boldsymbol{B}$, where $\boldsymbol{B}$ is a reduced matrix and $\boldsymbol{B} = \boldsymbol{Q}^{\top}\boldsymbol{A}$. We can achieve an approximate SVD of $\boldsymbol{A}$ by performing SVD on $\boldsymbol{B}$. For a dense matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, the basic randomized SVD costs $O(mnc)$ time and the memory cost is $O(mn)$.

The Streaming Randomized SVD: Recently, several streaming randomized SVD algorithms have been proposed [20], [21] for the situation where the input matrix can only be visited once. The input matrix $\boldsymbol{A}$ is only allowed to be visited once during the random projection step and is not allowed to be revisited in the following steps. In contrast, the basic randomized SVD visits $\boldsymbol{A}$ to generate $\boldsymbol{Q}$, but requires revisiting $\boldsymbol{A}$ when constructing the reduced matrix $\boldsymbol{B}$. We describe the streaming randomized SVD algorithm [21] as Alg. 1. In Alg. 1, four Gaussian random matrices $\boldsymbol{C}, \boldsymbol{H}, \boldsymbol{O}$ and $\boldsymbol{S}$ are constructed in Line 1, and three sketch matrices $\boldsymbol{X}, \boldsymbol{Y}$ and $\boldsymbol{Z}$ are drawn in Line 2 by random projection. Then a reduced matrix $\boldsymbol{W}$ is derived in Line 4 based on the three sketch matrices. The time cost of Line 2 is $O(mn(c+s))$. Lines 3-6 cost $O((m+n)(sc+c^2)+mcr)$. Considering that m, n are much larger than c, s, r, the total time cost of Alg. 1 is $O(mn(c+s))$, and the memory cost is $O(mn)$.

Algorithm 1: The Streaming Randomized SVD [21]

```
Input: $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, sketch size parameter $c, s$, rank
    parameter $r$
Output: SVD factors $\boldsymbol{U}\in\mathbb{R}^{m\times r}, \boldsymbol{\Sigma}\in\mathbb{R}^{r\times r}$ and $\boldsymbol{V}\in\mathbb{R}^{n\times r}$
    such that $\boldsymbol{A} \approx \hat{\boldsymbol{A}} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\top}$
1: $\boldsymbol{C} = \mathrm{randn}(n,\ c), \boldsymbol{H} = \mathrm{randn}(m,\ c), \boldsymbol{O} =$
   $\mathrm{randn}(m,\ s), \boldsymbol{S} = \mathrm{randn}(n,\ s)$
2: $\boldsymbol{X} = \boldsymbol{A}^{\top}\boldsymbol{H}, \boldsymbol{Y} = \boldsymbol{A}\boldsymbol{C}, \boldsymbol{Z} = \boldsymbol{O}^{\top}\boldsymbol{A}\boldsymbol{S}$
3: $\boldsymbol{Q} = \mathrm{orth}(\boldsymbol{Y}), \boldsymbol{P} = \mathrm{orth}(\boldsymbol{X})$
4: $\boldsymbol{W} = (\boldsymbol{O}^{\top}\boldsymbol{Q})^{\dagger}\boldsymbol{Z}(\boldsymbol{P}^{\top}\boldsymbol{S})^{\dagger}$
5: $[\hat{\boldsymbol{U}}, \hat{\boldsymbol{\Sigma}}, \hat{\boldsymbol{V}}] = \mathrm{svd}(\boldsymbol{W})$
6: $\boldsymbol{U} = \boldsymbol{Q}\hat{\boldsymbol{U}}(:, 1:r), \boldsymbol{\Sigma} = \hat{\boldsymbol{\Sigma}}(1:r, 1:r), \boldsymbol{V} = \boldsymbol{P}\hat{\boldsymbol{V}}(:, 1:r)$
```

The Nyström Method: The Nyström method [1], [16] is designed for SPSD kernel matrix $\boldsymbol{A} = f^{\circ}(\boldsymbol{M})$. The idea is to uniformly select c ($c \ll n$) columns of $\boldsymbol{A}$ to form a sketch matrix $\boldsymbol{Y}$, i.e., $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{C}$, where each column of $\boldsymbol{C}$ has an element equal to 1, and $\boldsymbol{A}$ is not explicitly constructed. The reduced matrix $\boldsymbol{W}$ is constructed by performing the Moore−Penrose inverse on the corresponding c rows and columns of $\boldsymbol{A}$, i.e., $\boldsymbol{C}^{\top}\boldsymbol{A}\boldsymbol{C}$. Finally, the Nyström approximation is defined as:

$$\boldsymbol{A} \approx \hat{\boldsymbol{A}} = \boldsymbol{Y}\boldsymbol{W}\boldsymbol{Y}^{\top} = (\boldsymbol{A}\boldsymbol{C})(\boldsymbol{C}^{\top}\boldsymbol{A}\boldsymbol{C})^{\dagger}(\boldsymbol{A}\boldsymbol{C})^{\top}. \tag{2}$$

The total time/memory cost of the Nyström method is $O(nc)$.

The FastSPSD Method: The FastSPSD [2] provides an improved matrix approximation formulation for the Nyström method, described as

$$\begin{aligned}\boldsymbol{A} \approx \hat{\boldsymbol{A}} = \boldsymbol{Y}\boldsymbol{W}\boldsymbol{Y}^{\top} &= \boldsymbol{Y}(\boldsymbol{S}^{\top}\boldsymbol{Y})^{\dagger}(\boldsymbol{S}^{\top}\boldsymbol{A}\boldsymbol{S})(\boldsymbol{Y}^{\top}\boldsymbol{S})^{\dagger}\boldsymbol{Y}^{\top}\\ &= (\boldsymbol{A}\boldsymbol{C})(\boldsymbol{S}^{\top}\boldsymbol{A}\boldsymbol{C})^{\dagger}(\boldsymbol{S}^{\top}\boldsymbol{A}\boldsymbol{S})((\boldsymbol{A}\boldsymbol{C})^{\top}\boldsymbol{S})^{\dagger}(\boldsymbol{A}\boldsymbol{C})^{\top}.\end{aligned} \tag{3}$$

[1]https://github.com/xyyphant0m/EWMA

Algorithm 2: FastSPSD Method [2]

```
Input: $\boldsymbol{A} = f^{\circ}(\boldsymbol{M}) \in \mathbb{R}^{n\times n}$, sketch size parameter $c, s$
Output: $\boldsymbol{Y}\in\mathbb{R}^{n\times c}, \boldsymbol{W}\in\mathbb{R}^{c\times c}$ such that $\boldsymbol{A}\approx\hat{\boldsymbol{A}}=\boldsymbol{Y}\boldsymbol{W}\boldsymbol{Y}^\top$
1: generate an uniform column sampling matrix $\boldsymbol{C} \in \mathbb{R}^{n\times c}$
2: $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{C}$
3: generate a leverage score sampling matrix $\boldsymbol{S} \in \mathbb{R}^{n\times s}$
   based on $\boldsymbol{Y}$
4: $\boldsymbol{Z} = \boldsymbol{S}^\top\boldsymbol{A}\boldsymbol{S}$, $\boldsymbol{W} = (\boldsymbol{S}^\top\boldsymbol{Y})^\dagger\boldsymbol{Z}(\boldsymbol{Y}^\top\boldsymbol{S})^\dagger$
```

The algorithm of FastSPSD can be described as Alg. 2. The $\boldsymbol{S} \in \mathbb{R}^{n\times s}$ in Line 3 is a sampling matrix based on leverage score [2], and s should meet that $s \geq c$. If $\boldsymbol{S} = \boldsymbol{C}$, the approximation formulation of the FastSPSD method is equivalent to the Nyström method. The computational complexity of the FastSPSD method is $O(nc^2)$.

III. Methodology

In this section, we propose two algorithms for approximating general asymmetric matrix $\boldsymbol{A} = f^{\circ}(\boldsymbol{M}) \in \mathbb{R}^{m\times n}$ and the SPSD kernel matrix $\boldsymbol{A} = f^{\circ}(\boldsymbol{M}) \in \mathbb{R}^{n\times n}$, respectively. The proposed algorithms' time and memory cost is linear to $m + n$ or n. In Sec. III-A, we propose an algorithm based on streaming randomized SVD and sparse-sign random projection. A shift skill is presented in Sec. III-B, which improves the SPSD matrix approximation, especially when the SPSD kernel matrix is not low-rank or with slowly-decayed singular values.

A. The ssrSVD Algorithm for Asymmetric Matrix

1) Problems of applying randomized SVD: Considering applying the basic randomized SVD in Sec. II to matrix $\boldsymbol{A}$, the explicit construction of $\boldsymbol{A} = f^{\circ}(\boldsymbol{M}) \in \mathbb{R}^{m\times n}$ will cost $O(mnd)$ time and $O(mn)$ memory for $\boldsymbol{M} = \boldsymbol{L}\boldsymbol{R}^\top$. The computation of $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{C}$, where $\boldsymbol{C}$ is a Gaussian random matrix, will cost $O(mnc)$ time. Therefore, the problem of sketching matrix $\boldsymbol{A}$ without explicitly constructing the matrix is the first problem to solve. Suppose that we have $\boldsymbol{Q}$, the computation of $\boldsymbol{B} = \boldsymbol{Q}^\top\boldsymbol{A}$ still costs $O(mnc)$ time. The second problem is to construct a reduced matrix $\boldsymbol{B}$. Therefore, directly applying the basic randomized SVD to solve the low-rank approximation of the element-wise functions of the asymmetric matrix is unrealistic.

2) The Solution: Borrowing the idea from Nyström and FastSPSD, we can choose the random projection matrix $\boldsymbol{C}$ to be a uniform column sampling matrix and avoid explicitly constructing $\boldsymbol{A}$, which will also reduce the time/memory costs of $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{C}$ to be $O(mc)$. Although the uniform column sampling is widely used in the Nyström method for time efficiency, it shows bad performance for approximation. Fortunately, there is another randomized dimension reduction map with comparable performance to the Gaussian random projection, which we call the sparse-sign random projection matrix [18]. It balances the efficiency and the effectiveness between the uniform column sampling matrix and the Gaussian random matrix. The construction of a sparse-sign random projection matrix is described in Alg. 3. First, a sparsity parameter z in the range $2 \leq z \leq n$ is input. As shown in Lines 3~6 of Alg. 3, each column of the matrix is generated independently at random. We generate z i.i.d random signs and place them in z uniformly random coordinates.

Algorithm 3: Generate Sparse-Sign Random Matrix

```
Input: row number $n$, column number $c$, sparsity parameter $z$
Output: $\boldsymbol{C}\in\mathbb{R}^{n\times c}, \boldsymbol{c}\in\mathbb{R}^{zc\times 1}$
1: $\boldsymbol{C} = \mathrm{zeros}(n,\ c)$
2: $\boldsymbol{c} = \mathrm{zeros}(zc,\ 1)$
3: for $j = 1, 2, ..., c$ do
4:     $\boldsymbol{c}((z(j-1)+1) : zj) = \mathrm{randperm}(n, z)$
5:     $\boldsymbol{C}(\boldsymbol{c}((z(j-1)+1) : zj), j) = \mathrm{sign}(\mathrm{randn}(z, 1))$
6: end for
```

If $\boldsymbol{C}$ is a sparse-sign random matrix, the computation cost of $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{C} \in \mathbb{R}^{m\times c}$ can be largely reduced. There are at most zc nonzeros in $\boldsymbol{C} \in \mathbb{R}^{n\times c}$ (notice $zc \ll n$), which are described by the coordinates with value in $[1, n]$ stored in $\boldsymbol{c}$. Therefore, we can compute $\boldsymbol{Y}$ by performing column sampling of $\boldsymbol{A}$ based on $\boldsymbol{c}$ instead of explicitly constructing $\boldsymbol{A}$ and performing matrix-matrix multiplication. This makes $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{C}$ has a computational cost of $O(mzc)$.

After the random projection step, the remaining problem is how to construct the reduced matrix. We consider adopting the streaming randomized SVD, which computes three sketch matrices and obtains a reduced matrix $\boldsymbol{W}$ based on these sketch matrices. Finally, we describe the sparse-sign streaming randomized SVD as Alg. 4 based on Alg. 1 and Alg. 3.

Algorithm 4: The Sparse-Sign Streaming Randomized SVD (ssrSVD)

```
Input: $\boldsymbol{A} = f^{\circ}(\boldsymbol{M}) \in \mathbb{R}^{m\times n}$, rank parameter $r$, sketch size
       parameter $c, s$, sparsity parameter $z$
Output: SVD factors $\boldsymbol{U}\in\mathbb{R}^{m\times r}, \boldsymbol{\Sigma}\in\mathbb{R}^{r\times r}$ and $\boldsymbol{V}\in\mathbb{R}^{n\times r}$
        such that $\boldsymbol{A}\approx\hat{\boldsymbol{A}}=\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$
1: generate sparse-sign matrices $\boldsymbol{C} \in \mathbb{R}^{n\times c}$, $\boldsymbol{H} \in \mathbb{R}^{m\times c}$,
   $\boldsymbol{O} \in \mathbb{R}^{m\times s}$, $\boldsymbol{S} \in \mathbb{R}^{n\times s}$ and the corresponding coordinate
   vectors $\boldsymbol{c}$, $\boldsymbol{h} \in \mathbb{R}^{zc\times 1}$, $\boldsymbol{o}$, $\boldsymbol{s} \in \mathbb{R}^{zs\times 1}$ using Alg. 3
2: compute $\boldsymbol{X} = \boldsymbol{A}^\top\boldsymbol{H}, \boldsymbol{Y} = \boldsymbol{A}\boldsymbol{C}, \boldsymbol{Z} = \boldsymbol{O}^\top\boldsymbol{A}\boldsymbol{S}$ based on
   $\boldsymbol{c}, \boldsymbol{h}, \boldsymbol{o}$, and $\boldsymbol{s}$, without explicitly stored $\boldsymbol{A}$ or $\boldsymbol{M}$
3: $\boldsymbol{Q} = \mathrm{orth}(\boldsymbol{Y}), \boldsymbol{P} = \mathrm{orth}(\boldsymbol{X})$
4: compute $\boldsymbol{W} = (\boldsymbol{O}^\top\boldsymbol{Q})^\dagger\boldsymbol{Z}(\boldsymbol{P}^\top\boldsymbol{S})^\dagger$ based on $\boldsymbol{o}, \boldsymbol{s}$
5: $[\hat{\boldsymbol{U}}, \hat{\boldsymbol{\Sigma}}, \hat{\boldsymbol{V}}] = \mathrm{svd}(\boldsymbol{W})$
6: $\boldsymbol{U}=\boldsymbol{Q}\hat{\boldsymbol{U}}(:, 1:r), \boldsymbol{\Sigma}=\hat{\boldsymbol{\Sigma}}(1:r, 1:r), \boldsymbol{V}=\boldsymbol{P}\hat{\boldsymbol{V}}(:, 1:r)$
```

Complexity of ssrSVD. Lines 1~2 requires $O((m+n)zc)$ time and $O((m+n)zc)$ memory. As for Lines 3~4, it requires $O(mc^2+nc^2)$ time. The Line 5 requires $O(c^3)$ time and $O(c^2)$ memory. Line 6 demands $O((m+n)cr)$ time. Considering that c, s, z are much smaller than m or n, the total time cost of S3SPSD is $O((m+n)(zc+c^2+cr))$ and the memory cost is $O((m+n)zc)$.

B. Improved Algorithm for SPSD Matrix with a Shift Skill

When applying Alg. 4 to solve the low-rank approximation of SPSD kernel matrix $\boldsymbol{A} = f^{\circ}(\boldsymbol{M}) \in \mathbb{R}^{n\times n}$, Alg. 4 can be simplified when setting $\boldsymbol{H}=\boldsymbol{C}$ and $\boldsymbol{O}=\boldsymbol{S}$. At this point it can be observed that the reduced matrix $\boldsymbol{W}$ in Alg. 4 is the same as that in Alg. 2, which implies that the approximation formula of Alg. 2 is a special case of Alg. 4. Yang et al. [22] give a theoretical and empirical evaluation that Nyström is better than the RFF method, when applied to low-rank matrices. Although the Nyström and FastSPSD methods are widely used for SPSD kernel matrix approximation, their performance may not be good when the SPSD kernel matrix is not low-rank, or with slowly-decayed singular values.

The shift skill [23] is proposed initially to accelerate the convergence of power method for eigenvalue computation [23] by reducing the ratio of the second largest eigenvalue to the largest one. For sketching SPSD kernel matrix $\boldsymbol{A}$, we can combine the shift skill with the random projection step to achieve a more accurate sketch matrix $\boldsymbol{Y}=\boldsymbol{AC}$.

Lemma 1. *[23] Suppose a matrix $\boldsymbol{A}\in\mathbb{R}^{n\times n}$, and a shift $\alpha\in\mathbb{R}$. For any eigenvalue λ of $\boldsymbol{A}$, $\lambda-\alpha$ is an eigenvalue of $\boldsymbol{A}-\alpha\boldsymbol{I}$, where $\boldsymbol{I}$ is the identity matrix. And, the eigenspace of $\boldsymbol{A}$ for λ is the same as the eigenspace of $\boldsymbol{A}-\alpha\boldsymbol{I}$ for $\lambda-\alpha$.*

The singular values of SPSD matrix $\boldsymbol{A}$ are the same as its eigenvalues [23]. Based on Lemma 1, we can see that $\sigma_i(\boldsymbol{A})-\alpha$ is the eigenvalue of $\boldsymbol{A}-\alpha\boldsymbol{I}$. Therefore, $|\sigma_i(\boldsymbol{A})-\alpha|$ is the singular value of $\boldsymbol{A}-\alpha\boldsymbol{I}$. But we notice that $|\sigma_i(\boldsymbol{A})-\alpha|$ is not necessarily the i-th largest singular value. If $\sigma_i(\boldsymbol{A})-\alpha>0$ for any $i\le e$, and they are e largest singular values of $\boldsymbol{A}-\alpha\boldsymbol{I}$, these shifted singular values obviously exhibit faster decay. The following Theorem states when these conditions are satisfied.

Theorem 1. *Suppose $\boldsymbol{A}$ is a SPSD matrix, positive number $\alpha\le\sigma_e(\boldsymbol{A})/2$ and $i\le e$. Then, $\sigma_i(\boldsymbol{A}-\alpha\boldsymbol{I})=\sigma_i(\boldsymbol{A})-\alpha$, where $\sigma_i(\cdot)$ denotes the i-th largest singular value. And, the left singular vector corresponding to the i-th singular value of $\boldsymbol{A}-\alpha\boldsymbol{I}$ is the same as the left singular vector corresponding to the i-th singular value of $\boldsymbol{A}$.*

Proof. For symmetric matrix $\boldsymbol{A}-\alpha\boldsymbol{I}$, the eigenvalue is $\sigma_i(\boldsymbol{A})-\alpha$ and the singular value is $|\sigma_i(\boldsymbol{A})-\alpha|$. When positive number $\alpha\le\sigma_e(\boldsymbol{A})/2$, we can have

$$\alpha-\sigma_i(\boldsymbol{A})\le\alpha\le\sigma_e(\boldsymbol{A})-\alpha. \tag{4}$$

For any $i>e$, we have $\sigma_i(\boldsymbol{A})-\alpha\le\sigma_e(\boldsymbol{A})-\alpha$. After combining it with (4), we can derive (5) as

$$|\sigma_i(\boldsymbol{A})-\alpha|\le\sigma_e(\boldsymbol{A})-\alpha. \tag{5}$$

For any $i\le e$, we will have

$$\sigma_i(\boldsymbol{A})-\alpha\ge\sigma_e(\boldsymbol{A})-\alpha>0. \tag{6}$$

Finally, we can derive $\sigma_i(\boldsymbol{A}-\alpha\boldsymbol{I})=\sigma_i(\boldsymbol{A})-\alpha, i\le e$ by combining (5) and (6). According to Lemma 1, the eigenspace of $\boldsymbol{A}$ for $\sigma_i(\boldsymbol{A})$ is the same as the eigenspace of $\boldsymbol{A}-\alpha\boldsymbol{I}$ for $\sigma_i(\boldsymbol{A})-\alpha$. Therefore, the left singular vector corresponding to the i-th singular value of $\boldsymbol{A}-\alpha\boldsymbol{I}$ is the same as the left singular vector corresponding to the i-th singular value of $\boldsymbol{A}$, for any $i\le e$. Therefore, as long as $\alpha\le\sigma_e(\boldsymbol{A})/2$, the left singular vectors corresponding to the first e singular values of $\boldsymbol{A}-\alpha\boldsymbol{I}$ are the same as those corresponding to $\boldsymbol{A}$. □

According to Theorem 1 and [17], we can have a better sketch matrix $\boldsymbol{Y}$ through the left singular vector of $(\boldsymbol{A}-\alpha\boldsymbol{I})\boldsymbol{C}$. The remaining problem is how to choose shift parameter α.

Considering that the ratio of $\frac{\sigma_i(\boldsymbol{A}-\alpha\boldsymbol{I})}{\sigma_1(\boldsymbol{A}-\alpha\boldsymbol{I})}$, when $i\le c$. As the value of α in Theorem 1 satisfied, we can derive that $\frac{\sigma_i(\boldsymbol{A}-\alpha\boldsymbol{I})}{\sigma_1(\boldsymbol{A}-\alpha\boldsymbol{I})}=\frac{\sigma_i(\boldsymbol{A})-\alpha}{\sigma_1(\boldsymbol{A})-\alpha}\le\frac{\sigma_i(\boldsymbol{A})}{\sigma_1(\boldsymbol{A})}$. It is obvious that the larger the α, the small the ratio $\frac{\sigma_i(\boldsymbol{A}-\alpha\boldsymbol{I})}{\sigma_1(\boldsymbol{A}-\alpha\boldsymbol{I})}$, and the faster decay the singular value. Therefore, a large number of α is preferred to maximize the effect of shift skill on improving the approximation error, while satisfying $\alpha\le\sigma_c(\boldsymbol{A})/2$. Notice that it's impossible to compute the singular value of $\boldsymbol{A}=f^\circ(\boldsymbol{M})\in\mathbb{R}^{n\times n}$ because the matrix $\boldsymbol{A}$ cannot be explicitly constructed. The approach is to use the singular value of $(\boldsymbol{A}-\alpha\boldsymbol{I})\boldsymbol{C}$ to approximate the singular value of $\boldsymbol{A}$, where $\boldsymbol{C}$ is the sparse-sign random matrix. Here we give the Lemma 2 and Theorem 2.

Lemma 2. *[24] Let $\boldsymbol{X},\boldsymbol{Y}$ be two $n\times n$ symmetric matrices. Then for the decreasingly ordered singular values σ of $\boldsymbol{X},\boldsymbol{Y}$ and $\boldsymbol{XY}$, $\sigma_{i+j-1}(\boldsymbol{XY})\le\sigma_i(\boldsymbol{X})\sigma_j(\boldsymbol{Y})$ and $\sigma_{i+j-1}(\boldsymbol{X}+\boldsymbol{Y})\le\sigma_i(\boldsymbol{X})+\sigma_j(\boldsymbol{Y})$ holds for any $1\le i,j\le n$ and $i+j\le n+1$.*

Theorem 2. *Suppose $\boldsymbol{A}\in\mathbb{R}^{n\times n}$, $\boldsymbol{C}\in\mathbb{R}^{n\times c}(c\ll n)$ is an orthonormal matrix and $0\le\alpha\le\sigma_c(\boldsymbol{A})/2$. Then,*

$$\sigma_i((\boldsymbol{A}-\alpha\boldsymbol{I})\boldsymbol{C})+\alpha\le\sigma_i(\boldsymbol{A}),\ i\le c. \tag{7}$$

Proof. First, we need to prove that $\sigma_i((\boldsymbol{A}-\alpha\boldsymbol{I})\boldsymbol{C})\le\sigma_i(\boldsymbol{A}-\alpha\boldsymbol{I})$. Considering $\boldsymbol{C}$ is an orthonormal matrix, we append zero columns to $\boldsymbol{C}$ to get an $n\times n$ matrix $\boldsymbol{B}=[\boldsymbol{C},\boldsymbol{0}]$. Then we have $\sigma_1(\boldsymbol{B})=1$, and we can get

$$\sigma_i((\boldsymbol{A}-\alpha\boldsymbol{I})\boldsymbol{B})\le\sigma_i(\boldsymbol{A}-\alpha\boldsymbol{I})\sigma_1(\boldsymbol{B})=\sigma_i(\boldsymbol{A}-\alpha\boldsymbol{I}) \tag{8}$$

according to Lemma 2. Here we know $(\boldsymbol{A}-\alpha\boldsymbol{I})\boldsymbol{B}=[(\boldsymbol{A}-\alpha\boldsymbol{I})\boldsymbol{C},0]$. For any $i\le c$, $\sigma_i((\boldsymbol{A}-\alpha\boldsymbol{I})\boldsymbol{C})=\sigma_i((\boldsymbol{A}-\alpha\boldsymbol{I})\boldsymbol{B})$. Therefore, we can get the result that $\sigma_i((\boldsymbol{A}-\alpha\boldsymbol{I})\boldsymbol{C})\le\sigma_i(\boldsymbol{A}-\alpha\boldsymbol{I})$ for $i\le c$.

After combining the fact $\sigma_i(\boldsymbol{A})=\sigma_i(\boldsymbol{A}-\alpha\boldsymbol{I})+\alpha$ in Theorem 1, we have $\sigma_i(\boldsymbol{A})=\sigma_i(\boldsymbol{A}-\alpha\boldsymbol{I})+\alpha\ge\sigma_i((\boldsymbol{A}-\alpha\boldsymbol{I})\boldsymbol{C})+\alpha$ for any $i\le c$. □

We use Alg. 3 to generate random projection matrix $\boldsymbol{C}\in\mathbb{R}^{n\times c}$, which includes $cz\ll n$ nonzero element. Therefore, we can guarantee that the generated $\boldsymbol{C}$ has at most one nonzero element per row and that each column is normalized, which further ensures that $\boldsymbol{C}$ is an orthonormal matrix.

Theorem 3. *Suppose $\boldsymbol{A}\in\mathbb{R}^{n\times n}$, $\boldsymbol{C}\in\mathbb{R}^{n\times c}(c\ll n)$ is an orthonormal matrix and set $\alpha_0=0$, $\alpha_i=(\sigma_c((\boldsymbol{A}-\alpha_{i-1}\boldsymbol{I})\boldsymbol{C})+\alpha_{i-1})/2$. Then the $\alpha_i\ge\alpha_{i-1}$ for any positive integer i.*

Proof. The Theorem is proved using induction.

When $i=1$, $\alpha_1=\sigma_c(\boldsymbol{AC})/2\ge\alpha_0$.

When $i \geq 1$, suppose $\alpha_{i-1} \geq \alpha_{i-2}$. According to Lemma 2, we have $\sigma_c(\boldsymbol{AC} - \alpha_{i-2}\boldsymbol{C}) = \sigma_c(\boldsymbol{AC} - \alpha_{i-1}\boldsymbol{C} + \alpha_{i-1}\boldsymbol{C} - \alpha_{i-2}\boldsymbol{C}) \leq \sigma_c(\boldsymbol{AC} - \alpha_{i-1}\boldsymbol{C}) + \alpha_{i-1} - \alpha_{i-2}$.

Therefore, we have $\alpha_i = (\sigma_c((\boldsymbol{A} - \alpha_{i-1}\boldsymbol{I})\boldsymbol{C}) + \alpha_{i-1})/2 \geq (\sigma_c((\boldsymbol{A} - \alpha_{i-2}\boldsymbol{I})\boldsymbol{C}) + \alpha_{i-2})/2 = \alpha_{i-1}$. $\square$

Based on Theorem 2 and Theorem 3, α_i can be computed according to $\alpha_i = (\sigma_c((\boldsymbol{A} - \alpha_{i-1}\boldsymbol{I})\boldsymbol{C}) + \alpha_{i-1})/2 = (\sigma_c(\boldsymbol{Y} - \alpha_{i-1}\boldsymbol{C}) + \alpha_{i-1})/2$ in an iterative way. Therefore, sequence $\{\alpha_i\}$ is in ascending order with upper bound $\sigma_c(\boldsymbol{A})/2$, which reflects that α_i converges with the increasing of i.

Based on Theorem 3, we may need to compute the singular value of $\boldsymbol{Y} - \alpha\boldsymbol{C}$ several times and it costs $O(nc^2)$ every time. To speed up the computation, we consider to compute the eigenvalue of $(\boldsymbol{Y} - \alpha\boldsymbol{C})^\top(\boldsymbol{Y} - \alpha\boldsymbol{C})$ and obtain the singular value by taking the square root of the corresponding eigenvalue. We expand $(\boldsymbol{Y} - \alpha\boldsymbol{C})^\top(\boldsymbol{Y} - \alpha\boldsymbol{C})$ to get

$$\begin{aligned}(\boldsymbol{Y} - \alpha\boldsymbol{C})^\top(\boldsymbol{Y} - \alpha\boldsymbol{C}) &= \boldsymbol{Y}^\top\boldsymbol{Y} - 2\alpha\boldsymbol{C}^\top\boldsymbol{AC} + \alpha^2\boldsymbol{I}_c \\ &= \boldsymbol{N} - 2\alpha\boldsymbol{T} + \alpha^2\boldsymbol{I}_c,\end{aligned} \tag{9}$$

where $\boldsymbol{N} \in \mathbb{R}^{c\times c}, \boldsymbol{T} \in \mathbb{R}^{c\times c}$, and $\boldsymbol{I}_c \in \mathbb{R}^{c\times c}$ is the identity matrix. After combining the Alg. 4 with Theorem 3 and (9), we can describe the sparse-sign streaming SPSD matrix approximation with shift as Alg. 5. Note that Alg. 5 has been designed and optimized for the SPSD matrix.

Algorithm 5: Sparse-Sign Streaming SPSD Matrix Approximation with Shift (S3SPSD)

```
Input: $\boldsymbol{A} = f^\circ(\boldsymbol{M}) \in \mathbb{R}^{n\times n}$, sketch size parameter $c, s$
Output: $\boldsymbol{Y} \in \mathbb{R}^{n\times c}, \boldsymbol{W} \in \mathbb{R}^{c\times c}$ and a shift value $\alpha$ such that
        $\boldsymbol{A} \approx \hat{\boldsymbol{A}} = \boldsymbol{YWY}^\top + \alpha\boldsymbol{I}_n$
1: generate a sparse-sign matrix $\boldsymbol{C} \in \mathbb{R}^{n\times c}$ and the
   coordinate vectors $\boldsymbol{c} \in \mathbb{R}^{zc\times 1}$ using Alg. 3
2: compute $\boldsymbol{Y} = \boldsymbol{AC}$ based on $\boldsymbol{c}$, without explicit $\boldsymbol{A}$
3: $\boldsymbol{N} = \boldsymbol{Y}^\top\boldsymbol{Y}$
4: compute $\boldsymbol{T} = \boldsymbol{C}^\top\boldsymbol{AC}$ based on $\boldsymbol{c}$, without explicit $\boldsymbol{A}$
5: while $\alpha$ is not convergence do
6:     $[\sim, \boldsymbol{\Lambda}] = \mathrm{eig}(\boldsymbol{N} - 2\alpha\boldsymbol{T} + \alpha^2\boldsymbol{I}_c)$
7:     if $\alpha > \sqrt{\boldsymbol{\Lambda}(c,c)}$ then
8:         break
9:     end if
10:    $\alpha = (\sqrt{\boldsymbol{\Lambda}(c,c)} + \alpha)/2$
11: end while
12: $[\boldsymbol{Y}, \sim, \sim] = \mathrm{svd}(\boldsymbol{Y} - \alpha\boldsymbol{C}, '\mathrm{econ}')$
13: generate a sparse-sign matrix $\boldsymbol{S} \in \mathbb{R}^{n\times s}$ and the
    coordinate vectors $\boldsymbol{s} \in \mathbb{R}^{zs\times 1}$ using Alg. 3
14: compute $\boldsymbol{Z} = \boldsymbol{S}^\top\boldsymbol{AS} - \alpha\boldsymbol{I}_s$ based on $\boldsymbol{s}$
15: compute $\boldsymbol{W} = (\boldsymbol{S}^\top\boldsymbol{Y})^\dagger\boldsymbol{Z}(\boldsymbol{Y}^\top\boldsymbol{S})^\dagger$ based on $\boldsymbol{s}$
```

Complexity of S3SPSD. According to Sec. III-A, the Lines 1~2 of Alg. 5 require $O(ncz)$ time/memory. Lines 3~4, cost $O(nc^2 + c^2z^2)$ time, Lines 5~11 cost $O(c^3)$ time, and Lines 12~15 cost $O(nc^2 + s^2z^2 + scz + cs^2 + c^2s)$ time. Because c, s, z are much smaller than n, the total time cost of S3SPSD algorithm is $O(ncz + nc^2)$ while its memory cost is $O(ncz)$.

IV. Experiments

We conduct two experiments to verify the effectiveness of the proposed two algorithms. In Sec. IV-A, we apply ssrSVD to approximate asymmetric matrix $\boldsymbol{A} = f^\circ(\boldsymbol{M}) \in \mathbb{R}^{m\times n}$ in the Sinkhorn algorithm. In Sec. IV-B, we evaluate the performance of S3SPSD on SPSD kernel matrix approximation. We have implemented the proposed algorithms and the compared baselines in Python, except PolyTensorSketch (PTS) for which we use the official codes[2] provided in the paper [15]. All experiments are carried out on a server with two Intel® Xeon® Silver 4214 CPUs (at 2.20GHz) and 126GB memory.

A. Application of ssrSVD algorithm in computing the optimal transport distance

The element-wise functions of the asymmetric matrix have many applications in machine learning. We evaluate ssrSVD based on the Sinkhorn algorithm for computing the optimal transport distance [7]–[9]. It requires multiplying an element-wise exponential function of a matrix with a vector at each iteration. And, the Sinkhorn algorithm for optimal transport distance is used for color transfer task [10]. We uniformly randomly sample $\boldsymbol{L} \in \mathbb{R}^{m\times 3}$ from the RGB pixels in the source image matrix $\boldsymbol{E} \in \mathbb{R}^{M\times 3}$ and $\boldsymbol{R} \in \mathbb{R}^{n\times 3}$ from the RGB pixels in the target image matrix $\boldsymbol{F} \in \mathbb{R}^{N\times 3}$, where M and N represent the total RGB pixels in the source and target images, respectively. The Sinkhorn algorithm iteratively computes $\boldsymbol{u} = \boldsymbol{a} \oslash (\exp^\circ(\boldsymbol{M})\boldsymbol{v})$ and $\boldsymbol{v} = \boldsymbol{b} \oslash (\exp^\circ(\boldsymbol{M})^\top\boldsymbol{u})$, where $\boldsymbol{M}(i,j) \triangleq -\frac{1}{\sigma}\|\boldsymbol{L}(i,:) - \boldsymbol{R}(j,:)\|_2^2$, σ is the scaling parameter, $\oslash$ represents the element-wise division, $\boldsymbol{u} \in \mathbb{R}^{m\times 1}, \boldsymbol{v} \in \mathbb{R}^{n\times 1}$ are initialized as vectors of all 1's. Since the pixels are uniformly randomly sampled from the images, $\boldsymbol{a} \in \mathbb{R}^m$ and $\boldsymbol{b} \in \mathbb{R}^n$ are two vectors of all $\frac{1}{m}$'s and $\frac{1}{n}$'s, respectively. The Sinkhorn algorithm finally returns a transfer matrix $\boldsymbol{T} = \mathrm{diag}(\boldsymbol{u})\exp^\circ(\boldsymbol{M})\mathrm{diag}(\boldsymbol{v})$, which is used for color transfer by nearest neighbor interpolation [10]. We evaluate the approximation error of transfer matrix $\boldsymbol{T}$ based on $\|\boldsymbol{T} - \hat{\boldsymbol{T}}\|_2$, where $\boldsymbol{T}$ represents the direct computation of the matrix function and $\hat{\boldsymbol{T}}$ that computed with with a matrix approximation method. Because $\exp^\circ(\boldsymbol{M})$ is not a SPSD matrix, the methods for SPSD kernel matrix approximation cannot be directly applied to this problem. However, a Nyström method for a scalable Sinkhorn algorithm was recently proposed [8]. It is realized by concatenating the matrix $\boldsymbol{L}$ and $\boldsymbol{R}$ into $\boldsymbol{Z} = [\boldsymbol{L}; \boldsymbol{R}] \in \mathbb{R}^{(m+n)\times 3}$, and letting $\boldsymbol{M}(i,j) = -\frac{1}{\sigma}\|\boldsymbol{Z}(i,:) - \boldsymbol{Z}(j,:)\|_2^2$. Therefore, we can compare the proposed ssrSVD algorithm with RFF [11], Nyström [1], FastSPSD [2], and S3SPSD algorithms[3]. We set $\sigma = 0.1$ and run 10 iterations in the Sinkhorn algorithm. For all methods, we set sketch size $c = 100, s = 300$. We additional set $r = c$ for ssrSVD and $z = 4$ for both of ssrSVD and S3SPSD.

We have done the color transfer experiment with some source and target images[4]. One of the color transfer results

[2]https://github.com/insuhan/polytensorsketch

[3]We omit the PolyTensorSketch [15] here, because the provided code by the paper cannot achieve the correct color transfer result.

[4]The images are downloaded from https://github.com/PythonOT/POT

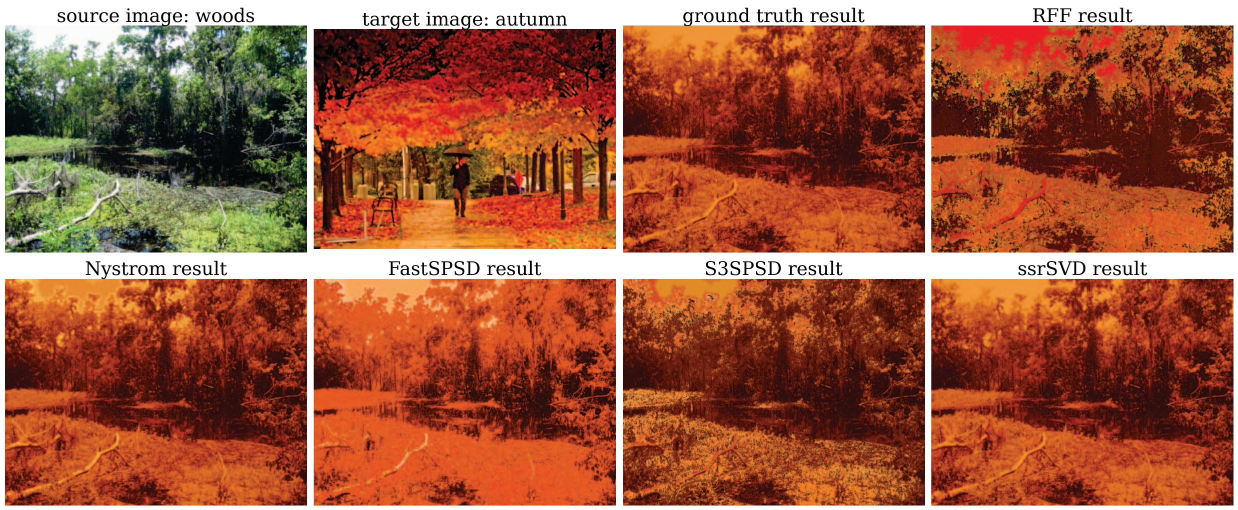

Fig. 1: The color transfer results obtained with different methods.

TABLE I: The comparison of the approximation error $\|\boldsymbol{T}-\hat{\boldsymbol{T}}\|_2$ of the transfer matrix $\boldsymbol{T}$ in the Sinkhorn algorithm. Here m and n represent the numbers of sampled RGB pixels and are the dimensions of the transfer matrix $\boldsymbol{T}$.

Dataset	m	n	RFF	Nyström	FastSPSD	S3SPSD	ssrSVD
source/target image: ocean_day / ocean_sunset	10000	8000	2.38×10^{0}	1.31×10^{-4}	9.53×10^{-3}	4.53×10^{-3}	1.14×10^{-8}
source/target image: ocean_sunset / ocean_day	8000	10000	9.73×10^{-1}	3.25×10^{-4}	1.05×10^{-1}	1.38×10^{-4}	7.39×10^{-9}
source/target image: autumn / woods	10000	10000	2.08×10^{0}	3.16×10^{-2}	1.02×10^{-1}	2.05×10^{-1}	6.16×10^{-6}
source/target image: woods / autumn	10000	10000	5.48×10^{-1}	4.19×10^{-3}	1.78×10^{-1}	1.09×10^{-1}	9.30×10^{-6}
source/target image: fallingwater / woods	8000	10000	2.11×10^{-1}	1.90×10^{-5}	7.55×10^{-2}	3.75×10^{-2}	2.00×10^{-6}
source/target image: woods / fallingwater	10000	8000	5.17×10^{-1}	5.38×10^{-4}	3.89×10^{-1}	3.06×10^{-2}	2.65×10^{-6}

is shown in Fig. 1. As we can see, only ssrSVD achieves an excellent color transfer result, which is indistinguishable from the ground truth result. RFF, FastSPSD, and S3SPSD achieve a lousy performance. Although the Nyström method achieves the second best result, it is still distinguishable. Furthermore, the approximation errors of transfer matrix $\boldsymbol{T}$ are listed in table I. The result shows that ssrSVD is the best one among all the baselines. The approximate error by ssrSVD is orders of magnitude smaller than other methods. To be specific, it is up to $10^8\times$ better in terms of the approximation error when compared to FastSPSD and is up to $10^5\times$ better compared to the Nyström method.

B. Comparison of S3SPSD algorithm and other SPSD matrix approximation methods

We compare the proposed S3SPSD algorithm with RFF, PTS, Nyström, FastSPSD, and ssrSVD. The ssrSVD takes the tested SPSD matrix as general asymmetric matrix. For FastSPSD method, ssrSVD and S3SPSD, we set $s = 5c$, which is shown to be effective in [2], [21]. For ssrSVD and S3SPSD, we additionally set $z = 4$ and we set $r = c$ for ssrSVD. Table II summarizes the statistics of the eight datasets used in our experiment. $\boldsymbol{G} \in \mathbb{R}^{n\times d}$ denotes the dataset matrix. The datasets satimage, cpu_small_scale, usps, mushroom, letter, and a9a are obtained from the LIBSVM[5]. The letter only contains training data,while the testing data is stored as a dataset named letter_large. The abalone dataset is from UCI machine learning repository[6]. We construct $\boldsymbol{A} = f^{\circ}(\boldsymbol{M})$ based on the dataset matrix $\boldsymbol{G}$. The RBF kernel function and the compactly supported RBF (csRBF) kernel function [6], [25] are considered, respectively. The RBF kernel matrix $\boldsymbol{A}_{RBF}$ is defined as

$$\boldsymbol{A}_{\mathrm{RBF}}(i,j) = \exp^{\circ}(-\frac{1}{\sigma^2}\|\boldsymbol{G}(i,:)-\boldsymbol{G}(j,:)\|_2^2), \tag{10}$$

and the csRBF kernel matrix is

$$\boldsymbol{A}_{\mathrm{csRBF}}(i,j)=\boldsymbol{A}_{\mathrm{RBF}}(i,j)[(1-\frac{\|\boldsymbol{G}(i,:)-\boldsymbol{G}(j,:)\|_2}{\theta})^{v}]_{+}, \tag{11}$$

where $\sigma > 0$ is the scaling parameter, $\theta > 0$ is the cutting-off point, $v > (d+1)/2$ and function $[a]_+ \triangleq max(a,0)$. For csRBF kernel, we choose $\theta = 3\sigma$ and $v = \lceil (d+1)/2 \rceil$ according to [25]. We focus on the kernel matrices with slowly decaying singular values, which the existing methods cannot well approximate. So, relatively small values of σ are set. For RBF kernel, we set $\sigma^2 = 0.2$ for satimage, usps, letter and

[5]https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/
[6]https://archive.ics.uci.edu/ml/index.php

TABLE II: Statistics of datasets for kernel approximation.

Dataset	satimage	abalone	cpu_small_scale	usps	mushroom	letter	letter_large	a9a
$n \times d$	4435×36	4177×8	8192×12	9298×256	8124×112	15000×16	20000×16	32561×123

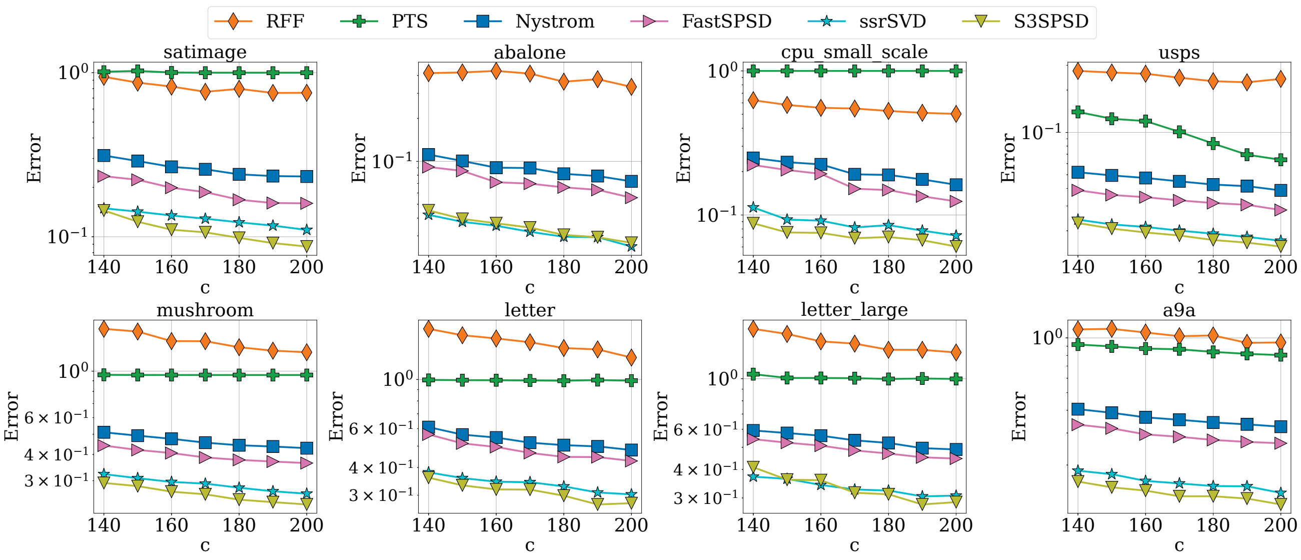

Fig. 2: The relative approximation error $\frac{\|A-\hat{A}\|_2}{\|A\|_2}$ w.r.t sketch size c for approximating RBF kernel matrices.

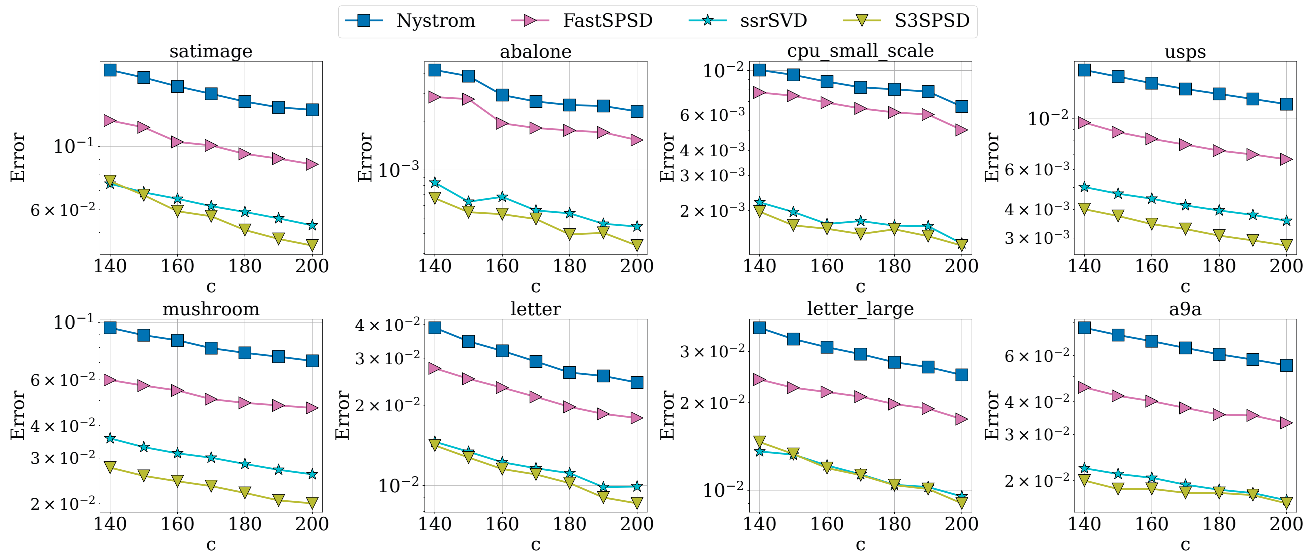

Fig. 3: The relative approximation error $\frac{\|A-\hat{A}\|_2}{\|A\|_2}$ w.r.t sketch size c for approximating csRBF kernel matrices. The RFF method does not work due to the lack of a corresponding non-negative measure. The PTS method cannot handle this problem, either.

letter_large, $\sigma^2 = 0.02$ for abalone and cpusmall_scale, and $\sigma^2 = 2.5$ for mushroom and a9a. The csRBF kernel matrix is sparse, which always has slowly decaying singular values. For csRBF kernel, we set $\sigma^2 = 1000$ for usps, mushroom and a9a, and $\sigma^2 = 10$ for the remaining datasets. As for the evaluation metric, we use the relative approximation error $\frac{\|A-\hat{A}\|_2}{\|A\|_2}$, where $\hat{A}$ is the computed low-rank approximation of A. Since all the methods are randomized, we run each algorithm ten times and report the average relative approximation error.

We summarize the results of the SPSD kernel approximation of $\boldsymbol{A}_{RBF}$ and $\boldsymbol{A}_{csRBF}$ in Fig. 2 and Fig. 3, respectively. From them, we see that S3SPSD always shows the best result for all the datasets, and ssrSVD is the second best method. They both show significantly better performance than the other methods. As shown in Fig. 2, concerning the relative approximation error $\frac{\|\boldsymbol{A}-\hat{\boldsymbol{A}}\|_2}{\|\boldsymbol{A}\|_2}$, S3SPSD achieves averagely 86.29% and 83.75% relative better performance gain than RFF and PTS in a9a dataset, respectively. Compared with Nyström, FastSPSD, and ssrSVD, it still achieves an average relative improvement of 61.12%, 51.98%, and 13.56%. The PTS approximates the RBF kernel based on the fact that $\boldsymbol{A}_{\text{RBF}} = \boldsymbol{D}\exp^{\circ}(\frac{1}{\sigma^2}\boldsymbol{G}\boldsymbol{G}^{\top})\boldsymbol{D}$, where $\boldsymbol{D}$ is a diagonal matrix with diagonal elements computed as $\boldsymbol{D}(i,i) = \exp(-\frac{1}{2\sigma^2}\|\boldsymbol{G}(i,:)\|_2^2)$. If the scaling parameter σ is small, the element of matrix $\exp^{\circ}(\frac{1}{\sigma^2}\boldsymbol{G}\boldsymbol{G}^{\top})$ can be of huge value, which makes PTS hard to approximate this matrix. For the abalone dataset, matrix $\exp^{\circ}(\frac{1}{\sigma^2}\boldsymbol{G}\boldsymbol{G}^{\top})$ actually includes value `INF` when $\sigma^2 = 0.02$. This makes PTS does not work for the abalone dataset, and thus its results are not available in Fig. 2. The RFF and PTS methods are not able to approximate the csRBF kernel matrix. So in Fig. 3, we do not show their results. The figure reveals that, regarding the relative approximation error, the propoesd S3SPSD algorithm outperforms Nyström, FastSPSD and ssrSVD by 71.83%, 52.68%, and 6.57% on average respectively, for the a9a dataset. The both figures also show that the S3SPSD method obtains more accurate results than the ssrSVD method, which attributes to the shift skill proposed in Sect. III.B.

C. *Scalability*

We randomly generate some synthetic matrices $\boldsymbol{R} \in \mathbb{R}^{n\times d}$ with different values of n, and test the proposed ssrSVD and S3SPSD algorithms for approximating the corresponding RBF kernel matrices. We set $d = 20, c = r = 50, s = 250, z = 4$. The running time of the both algorithms are shown in Fig. 4. From the figure we see that their runtime increases nearly linearly as the matrix dimension n grows.

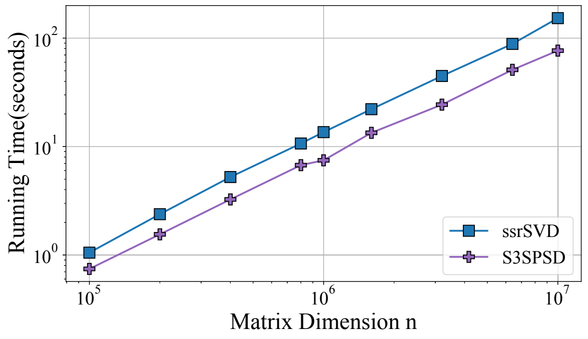

Fig. 4: The increasing trends of proposed algorithms' runtime.

V. Conclusion

In this work, we propose two linear-complexity algorithms for the low-rank approximation of matrix element-wise function $\boldsymbol{A} = f^{\circ}(\boldsymbol{M}) \in \mathbb{R}^{m\times n}$. Firstly, for the asymmetric matrix $\boldsymbol{A}$, a sparse-sign streaming randomized SVD (ssrSVD) algorithm based on sparse-sign random projection and a streaming SVD algorithm is proposed. Secondly, we propose a shift skill to improve the approximation accuracy of SPSD kernel matrix $\boldsymbol{A}$, which is effective for matrix with slowly-decayed singular values, and further obtain the sparse-sign streaming SPSD matrix approximation with shift (S3SPSD) algorithm. The experiment on color transfer based on the Sinkhorn algorithm shows that the ssrSVD algorithm achieves orders of magnitude better accuracy than the existing methods for approximating asymmetric matrix element-wise function, and produces high-quality color transfer result. The experiment on SPSD kernel matrix approximation shows that the S3SPSD algorithm is better than the ssrSVD algorithm, and both of them are significantly better than the other baselines.

References

[1] P. Drineas, M. W. Mahoney, and N. Cristianini, "On the nyström method for approximating a gram matrix for improved kernel-based learning." *The Journal of Machine Learning Research*, vol. 6, no. 12, 2005.

[2] S. Wang, Z. Zhang, and T. Zhang, "Towards more efficient spsd matrix approximation and cur matrix decomposition," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 7329–7377, 2016.

[3] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 315–323.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *NeurIPS*, vol. 30, 2017.

[5] T. Hofmann, B. Schölkopf, and A. J. Smola, "Kernel methods in machine learning," *The annals of statistics*, vol. 36, no. 3, pp. 1171–1220, 2008.

[6] M. G. Genton, "Classes of kernels for machine learning: A statistics perspective," *the Journal of Machine Learning Research*, vol. 2, no. 2, pp. 299–312, 2002.

[7] M. Cuturi, "Sinkhorn distances: Lightspeed computation of optimal transport," *NeurIPS*, vol. 26, 2013.

[8] J. Altschuler, F. Bach, A. Rudi, and J. Niles-Weed, "Massively scalable sinkhorn distances via the nyström method," *NeurIPS*, vol. 32, 2019.

[9] G. Peyré, M. Cuturi *et al.*, "Computational optimal transport: With applications to data science," *Foundations and Trends® in Machine Learning*, vol. 11, no. 5-6, pp. 355–607, 2019.

[10] S. Ferradans, N. Papadakis, G. Peyré, and J.-F. Aujol, "Regularized discrete optimal transport," *SIAM Journal on Imaging Sciences*, vol. 7, no. 3, pp. 1853–1882, 2014.

[11] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," *NeurIPS*, vol. 20, 2007.

[12] ——, "Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning," *NeurIPS*, vol. 21, 2008.

[13] Q. Le, T. Sarlós, and A. Smola, "Fastfood-computing hilbert space expansions in loglinear time," in *ICML*, 2013, pp. 244–252.

[14] N. Pham and R. Pagh, "Fast and scalable polynomial kernels via explicit feature maps," in *SIGKDD*, 2013, pp. 239–247.

[15] I. Han, H. Avron, and J. Shin, "Polynomial tensor sketch for element-wise function of low-rank matrix," in *ICML*, 2020, pp. 3984–3993.

[16] C. Williams and M. Seeger, "Using the nyström method to speed up kernel machines," *NeurIPS*, vol. 13, 2000.

[17] N. Halko, P.-G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM review*, vol. 53, no. 2, pp. 217–288, 2011.

[18] P. Martinsson and J. Tropp, "Randomized numerical linear algebra: foundations & algorithms (2020)," *arXiv preprint arXiv:2002.01387*.

[19] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher, "Fixed-rank approximation of a positive-semidefinite matrix from streaming data," *NeurIPS*, vol. 30, 2017.

[20] W. Yu, Y. Gu, and J. Li, "Single-pass PCA of large high-dimensional data," in *IJCAI*, 2017, pp. 3350–3356.

[21] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher, "Streaming low-rank matrix approximation with an application to scientific simulation," *SIAM Journal on Scientific Computing*, vol. 41, no. 4, pp. A2430–A2463, 2019.

[22] T. Yang, Y.-F. Li, M. Mahdavi, R. Jin, and Z.-H. Zhou, “Nyström method vs random fourier features: A theoretical and empirical comparison,” *NeurIPS*, vol. 25, 2012.

[23] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU press, 2013.

[24] R. A. Horn and C. R. Johnson, *Topics in Matrix Analysis*. Cambridge University Press, 1991, p. 134–238.

[25] A. Gittens and M. W. Mahoney, “Revisiting the nyström method for improved large-scale machine learning,” *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 3977–4041, 2016.