Randomized Singular Value Decomposition Algorithm

Ruicheng Ao 1900012179

March 1, 2022

1 Problem description

For a matrix $A \in \mathbb{R}^{m\times n}$, its singular value decomposition (SVD) can be formulated as

$$A = U\Sigma V^\top = \sum_{k=1}^{\min(m,n)} \sigma_k u_k v_k^\top, \tag{1}$$

where $\sigma_1 \geq \sigma_2 \geq \ldots, \sigma_{\min(m,n)} \geq 0$ are so-called singular values of A and U, V have orthogonal columns. In some situations, it is requisited to know the k largest singular values of A. Traditional methods based on numerical algebra can obtain such decomposition with a cost of $\mathcal{O}(mnk)$ [2]. To further reduce the time cost, many randomized procedures are investigated. We mainly focus on two of them, namely, Linear Time SVD [1] and Prototype Randomized SVD [3].

2 Algorithm description

In this section, we give detailed descriptions about the two SVD algorithms mentioned above.

2.1 Linear Time SVD

The linear time SVD approximate $A \in \mathbb{R}^{m\times n}$ with a matrix $C \in \mathbb{R}^{m\times c}$, where c only depends on k. The columns of C are randomly sampled from those of A according to importance. The general framework is given in Algorithm 1.

2.2 Propotype Randomized SVD

The Prototype Randomized SVD method utilizes random sampling to identify a subsapce that captures most characteristics of a matrix. The sampling reduces the computational cost by projection through an orthogonal matrix obtained from the QR decomposition. After that, the reduced matrix is directly manipulated to obtain the decomposition. We demonstrate the framework in Algorithm 2

3 Numerical experiments

In this section, we test the above two algorithms on two datasets to see the efficiency of SVD. Then they are applied to accelerate the matrix completion problem.

Algorithm 1 Linear Time SVD

Input $A \in \mathbb{R}^{m\times n}$, number of samples c

Set importance weight $p_i = \|A_i\|_2^2/\|A\|_F^2$ for sampling
for $l = 1, 2, \ldots, c$ **do**
  Select $i_l \in \{1, 2, \ldots, n\}$ with probability distribution $\mathcal{P}(i_l = j) = p_j$
  Set $C_l = A_{i_l}/\sqrt{cp_{i_l}}$
end for
Compute the $k-$largest sigular values of $C = U_c\Sigma_c V_c^\top$
if Postprocessing **then**
  Set $Y = A^\top U_c$
  Compute QR decomposition $Y = QR$
  Compute SVD for $R^\top = U_r\Sigma_r V_r^\top$
  Set $U_k = U_c U_r, V_k = QV_r$
end if
return The approximate $k-$largest singular values $\Sigma_k = \Sigma_r$ with left and right singular values U_k, V_k.

Algorithm 2 Prototype Randomized SVD

Input $A \in \mathbb{R}^{m\times n}$, preprocessing exponential q, number of additional sampling p

Generate a Gaussian test matrix $\Omega \in \mathbb{R}^{n\times(k+p)}$
Set $Y = (AA^\top)^q A\Omega$
Compute QR decomposition $Y = QR$
Projection $B = Q^\top A$
Compute SVD $B = U_B\Sigma_B V_B T^\top$
Set $U_k = QU_B$
return The $k-$largetst singular values $\Sigma_k = \Sigma_B$ with the left and right vectors $U_k, V_k = V_B$

Algorithm	baseline	Linear time			Prototype randomized		
	-	$c = 2k$	$c = 10k$	$c = 50k$	$q = 0$	$q = 1$	$q = 2$
$k = 5$	$1.19e-02$	$2.04e-02$	$2.92e-02$	$7.15e-02$	$1.24e-01$	$8.52e-02$	$5.07e-02$
$k = 10$	$1.69e-02$	$1.78e-02$	$5.79e-02$	$1.45e-01$	$1.94e-01$	$1.33e-01$	$4.16e-02$
$k = 15$	$1.78e-02$	$2.71e-02$	$1.39e-01$	$1.58e-01$	$1.67e-01$	$1.62e-01$	$6.26e-02$
$k = 20$	$2.02e-02$	$3.79e-02$	$1.81e-01$	$1.75e-01$	$1.92e-01$	$1.78e-01$	$7.03e-02$

Table 1: CPU time for Linear time SVD and Prototype randomized SVD with different parameters and k

Algorithm	Linear time			Prototype randomized		
	$c = 2k$	$c = 10k$	$c = 50k$	$q = 0$	$q = 1$	$q = 2$
$k = 5$	$1.13e-01$	$1.05e-01$	$3.87e-02$	$6.52e-02$	$3.89e-02$	$2.35e-02$
$k = 10$	$7.22e-02$	$4.76e-02$	$2.56e-02$	$7.99e-16$	$8.29e-16$	$8.48e-16$
$k = 15$	$3.88e-02$	$2.76e-02$	$1.06e-02$	$8.34e-16$	$8.38e-16$	$9.88e-16$
$k = 20$	$8.49e-16$	$8.40e-16$	$7.33e-16$	$8.84e-16$	$6.73e-16$	$1.09e-15$

Table 2: Relative error of eigenvalue for Linear time SVD and Prototype randomized SVD with different parameters and k on dataset 1

3.1 Comparison of SVD and image restoration

Synthetic datasets The first dataset is generated by the matlab code

$$
\begin{aligned}
&m = 2048;\\
&n = 512;\\
&p = 20;\\
&A = randn(m, p) \star randn(p, n);
\end{aligned}
$$

The second example is our self-chosen image of size 1080×1920. We convert it into grey scale matrix and apply partial SVD with different k on it to see the performance and restoration effact. For the first example, we compare the performance of Linear time with $c = 2k, 10k, 50k$ and Prototype $q = 0, 1, 2$, while for the second example, we mainly focus on the influence of different k on recovery of the image, which is the most important problem in image processing. $c = 10k$ is chosen in example 2.

The results are displayed as below. Figures 1-4 show the magnitude of each eigenvalues compared with the MATLAB default function *svds* on dataset 1. Figures 6-23 show the recovery of image with different k. The tables 1-12 record the relative error of eigenvalues and the corresponding eigen vectors, as well as the CPU time compared with the baseline *svds*.

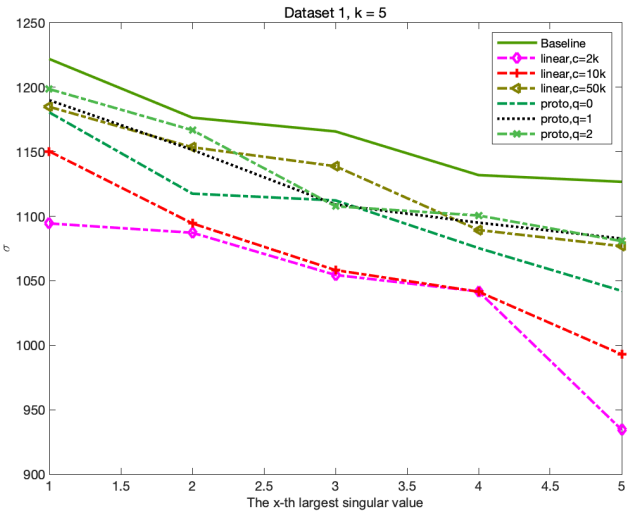

Figure 1: $k = 5$

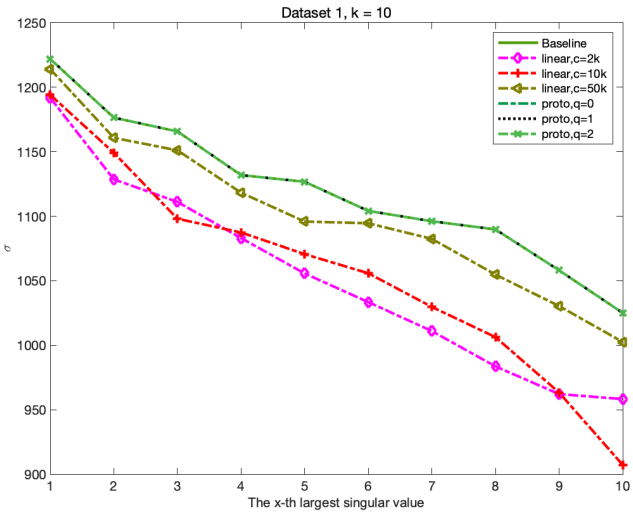

Figure 2: $k = 10$

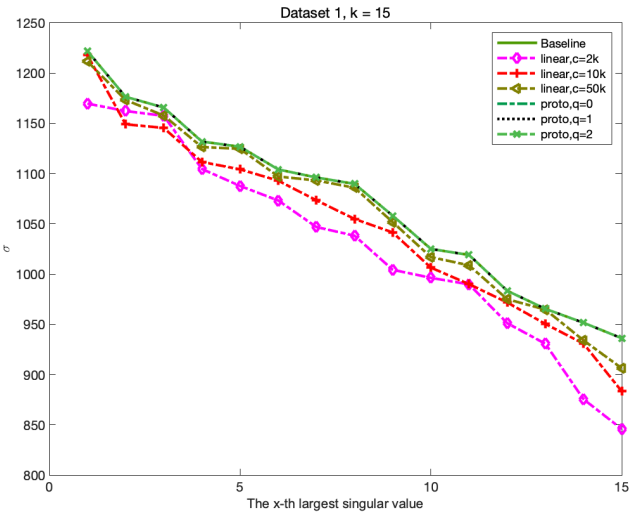

Figure 3: $k = 15$

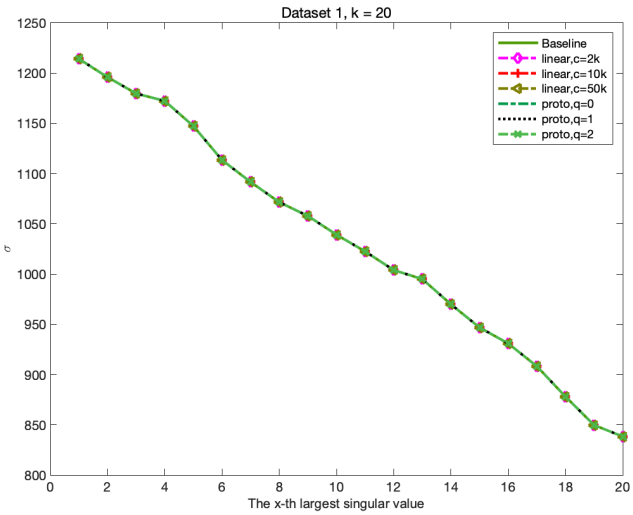

Figure 4: $k = 20$

Results on dataset 1 with different k

Algorithm	Linear time			Prototype randomized		
	$c = 2k$	$c = 10k$	$c = 50k$	$q = 0$	$q = 1$	$q = 2$
$k = 5$	$1.30e-02$	$1.26e-02$	$9.01e-03$	$1.18e-02$	$1.04e-02$	$6.21e-03$
$k = 10$	$1.09e-02$	$1.11e-02$	$7.71e-03$	$1.80e-15$	$3.88e-15$	$4.73e-15$
$k = 15$	$1.10e-02$	$1.03e-02$	$5.99e-03$	$1.47e-15$	$1.15e-15$	$1.93e-15$
$k = 20$	$1.44e-15$	$2.72e-15$	$1.28e-15$	$3.52e-15$	$2.20e-15$	$2.48e-15$

Table 4: Relative $\|\cdot\|_F$ error of V for Linear time SVD and Prototype randomized SVD with different parameters and k on dataset 1

Algorithm	CPU time	Error singular value	Error U	Error V	Error A
Linear time	$6.46e-02$	$2.24e-02$	$1.92e-02$	$2.20e-02$	$2.44e-01$
Prototype	$1.56e-01$	$1.14e-03$	$6.18e-03$	$2.44e-03$	$2.27e-01$
SVDS	$5.10e-02$	$4.66e-16$	$4.49e-16$	$1.67e-16$	$2.27e-01$

Table 5: Comparison of proposed algorithms on realistic image with baseline, $k = 5$

Algorithm	Linear time			Prototype randomized		
	$c = 2k$	$c = 10k$	$c = 50k$	$q = 0$	$q = 1$	$q = 2$
$k = 5$	$1.31e-02$	$1.27e-02$	$9.22e-03$	$1.19e-02$	$1.05e-02$	$6.48e-03$
$k = 10$	$1.11e-02$	$1.12e-02$	$7.89e-03$	$1.79e-15$	$3.87e-15$	$4.74e-15$
$k = 15$	$1.11e-02$	$1.04e-02$	$6.10e-03$	$1.46e-15$	$1.14e-15$	$1.93e-15$
$k = 20$	$1.44e-15$	$2.72e-15$	$1.27e-15$	$3.52e-15$	$2.20e-15$	$2.48e-15$

Table 3: Relative $\|\cdot\|_F$ error of U for Linear time SVD and Prototype randomized SVD with different parameters and k on dataset 1

Algorithm	CPU time	Error singular value	Error U	Error V	Error A
Linear time	$6.19e-02$	$1.30e-02$	$3.66e-02$	$3.47e-02$	$1.96e-01$
Prototype	$1.57e-01$	$5.37e-04$	$7.09e-03$	$2.74e-03$	$1.84e-01$
SVDS	$4.03e-02$	$4.20e-16$	$1.12e-13$	$3.74e-14$	$1.83e-01$

Table 6: Comparison of proposed algorithms on realistic image with baseline, $k = 10$

Algorithm	CPU time	Error singular value	Error U	Error V	Error A
Linear time	$7.21e-02$	$8.64e-03$	$3.71e-02$	$3.65e-02$	$1.70e-01$
Prototype	$1.73e-01$	$9.10e-04$	$1.26e-02$	$6.54e-03$	$1.61e-01$
SVDS	$6.69e-02$	$5.19e-16$	$1.20e-15$	$8.35e-16$	$1.61e-01$

Table 7: Comparison of proposed algorithms on realistic image with baseline, $k = 15$

Algorithm	CPU time	Error singular value	Error U	Error V	Error A
Linear time	$8.13e-02$	$9.09e-03$	$7.51e-02$	$6.14e-02$	$1.57e-01$
Prototype	$1.94e-01$	$8.76e-04$	$1.83e-02$	$1.04e-02$	$1.48e-01$
SVDS	$1.06e-01$	$4.26e-16$	$1.45e-15$	$1.01e-15$	$1.47e-01$

Table 8: Comparison of proposed algorithms on realistic image with baseline, $k = 20$

Algorithm	CPU time	Error singular value	Error U	Error V	Error A
Linear time	$7.20e-01$	$6.03e-03$	$2.63e-01$	$2.01e-01$	$9.28e-02$
Prototype	$9.00e-01$	$5.54e-04$	$9.58e-02$	$6.94e-02$	$8.61e-02$
SVDS	$7.46e-01$	$4.70e-16$	$2.02e-14$	$1.52e-14$	$8.57e-02$

Table 9: Comparison of proposed algorithms on realistic image with baseline, $k = 100$

Algorithm	CPU time	Error singular value	Error U	Error V	Error A
Linear time	$1.22e+00$	$3.36e-03$	$3.77e-01$	$2.86e-01$	$6.57e-02$
Prototype	$1.55e+00$	$2.28e-04$	$1.02e-01$	$7.43e-02$	$6.09e-02$
SVDS	$1.60e+00$	$4.71e-16$	$5.50e-14$	$4.14e-14$	$6.07e-02$

Table 10: Comparison of proposed algorithms on realistic image with baseline, $k = 200$

Figure 5: The realistic image

We observe that the Linear time SVD works not very well on dataset 1 when c is small, however, it shows good performance when k or c is large. All algorithms behave well on realistic image, from that the tables show that they can exactly recover the eigenvalues even when $k = 5$. An explanation for better behavior when k is larger is that larger k results in larger sample sets, which leads to better simulation of A. Another interpretation is that the rank of A is exactly 20 in the first dataset, thus when $k \geq 10$, the restoration of the information of A is complete for the Prototype SVD and $k = 20$ for Linear time SVD.

On the other hand, we find that for problems of small scale (rank) like dataset 1, the random SVD has little advantage, while for much larger matrix in the realistic image, the random algorithms outperform the baseline MATLAB SVD, which is corresponding to our intuition that, it is hard to defeat MATLAB when the problem is of small scale, while we can do much better for larger problem.

The relative error in U, V does not mean that the approximate performance is not good, but it may due to space representation, as we see the error in singular value is relatively low. On the other hand, the recovery result of our proposed algorithms is also really good, which looks better than the baseline. This proves its practical use. Additionally, we find that about 200 singular values can recover a realistic image of size 1920×1080.

3.2 Accelerate low-rank matrix recovery

We apply the partial SVD of Linear time algorithm and Prototype algorithm with $k = 5$ on the lrmr problem

$$\min_{X \in R^{m \times n}} \frac{1}{2} \sum_{(i,j) \in \Omega} (X_{ij} - M_{ij})^2 + \mu \|X\|_*. \tag{2}$$

The settings follow those in midterm project. We list the results in Table 11,12.

It should be mentioned that the acceleration is obvious compared with the default SVD in MATLAB. Also, despite more time Linear time algorithm has spent than prototype (though still faster than vanilla SVD), the much better recovery with primitive matrix is obtained. On the other hand, our proposed SVD also maintains low nuclear norm and low error with M, as well as opt value. Therefore, our acceleration by applying SVD techniques mentioned above is truly successful!

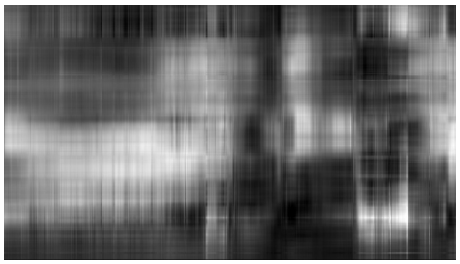

Figure 6: Linear time

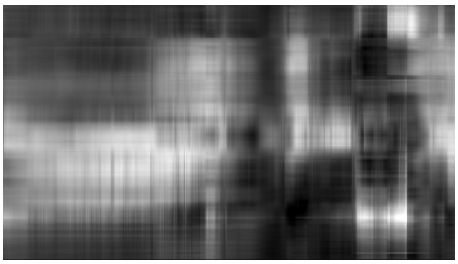

Figure 7: Prototype

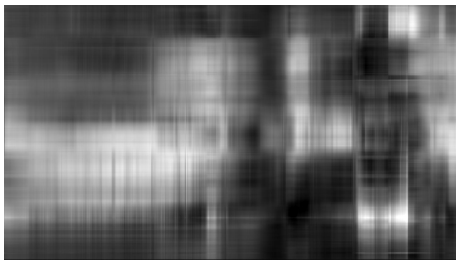

Figure 8: SVDS

Results on realistic image, $k = 5$

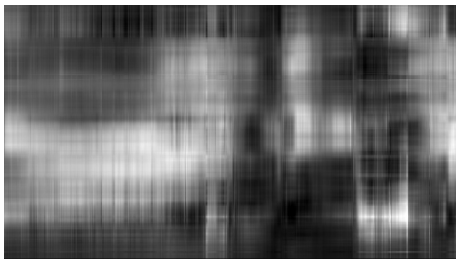

Figure 9: Linear time

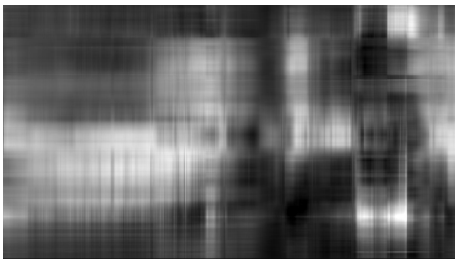

Figure 10: Prototype

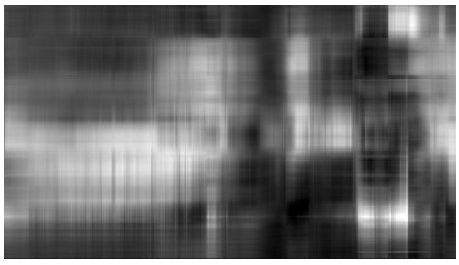

Figure 11: SVDS

Results on realistic image, $k = 10$

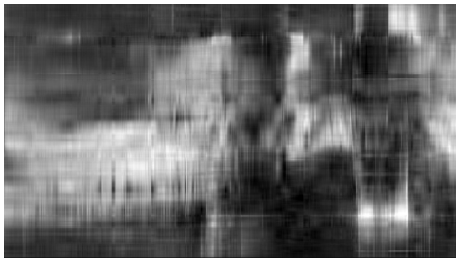

Figure 12: Linear time

Figure 13: Prototype

Figure 14: SVDS

Results on realistic image, $k = 15$

Figure 15: Linear time

Figure 16: Prototype

Figure 17: SVDS

Results on realistic image, $k = 20$

Figure 18: Linear time

Figure 19: Prototype

Figure 20: SVDS

Results on realistic image, $k = 100$

Figure 21: Linear time

Figure 22: Prototype

Figure 23: SVDS

Results on realistic image, $k = 200$

Algorithm	μ =0.1				
	CPU time	Opt value	Error with M in norm $\|\cdot\|_{\mathcal{F}}$	Error with A in norm $\|\cdot\|_{\mathcal{F}}$	$\|x\|_*$
Mosek	$1.67e+00$	$1.19e+01$	$5.43e-03$	$1.58e-01$	$1.19e+02$
ADMM	$2.51e-01$	$1.19e+01$	$5.47e-03$	$1.73e-01$	$1.19e+02$
ADMM linear time	$2.19e-01$	$1.24e+01$	$1.73e-02$	$8.04e-02$	$1.19e+02$
ADMM prototype	$2.02e-01$	$1.19e+01$	$5.83e-03$	$1.42e-01$	$1.19e+02$
Algorithm	μ =0.01				
	CPU time	Opt value	Error with M in norm $\|\cdot\|_{\mathcal{F}}$	Error with A in norm $\|\cdot\|_{\mathcal{F}}$	$\|x\|_*$
Mosek	$2.09e+00$	$1.20e+00$	$5.76e-04$	$1.56e-01$	$1.20e+02$
ADMM	$2.09e-01$	$1.20e+00$	$5.79e-04$	$2.02e-01$	$1.20e+02$
ADMM linear time	$1.72e-01$	$1.80e+00$	$1.87e-02$	$6.60e-02$	$1.21e+02$
ADMM prototype	$1.43e-01$	$1.20e+00$	$8.82e-04$	$1.52e-01$	$1.20e+02$
Algorithm	μ =0.001				
	CPU time	Opt value	Error with M in norm $\|\cdot\|_{\mathcal{F}}$	Error with A in norm $\|\cdot\|_{\mathcal{F}}$	$\|x\|_*$
Mosek	$1.11e+00$	$1.20e-01$	$5.77e-05$	$1.56e-01$	$1.20e+02$
ADMM	$2.06e-01$	$1.20e-01$	$5.91e-05$	$2.50e-01$	$1.20e+02$
ADMM linear time	$1.49e-01$	$1.84e-01$	$6.06e-03$	$5.49e-02$	$1.21e+02$
ADMM prototype	$1.31e-01$	$1.20e-01$	$9.84e-05$	$1.59e-01$	$1.20e+02$

Table 11: Numerical result on example 1 with different μ

Algorithm	μ =0.1				
	CPU time	Opt value	Error with M in norm $\|\cdot\|_{\mathcal{F}}$	Error with A in norm $\|\cdot\|_{\mathcal{F}}$	$\|x\|_*$
Mosek	$9.04e+00$	$1.99e+01$	$1.86e-03$	$2.33e-03$	$1.99e+02$
ADMM	$8.25e-01$	$1.99e+01$	$1.86e-03$	$2.33e-03$	$1.99e+02$
ADMM linear time	$8.45e-01$	$2.00e+01$	$2.55e-03$	$2.95e-03$	$1.99e+02$
ADMM prototype	$2.91e-01$	$1.99e+01$	$1.86e-03$	$2.33e-03$	$1.99e+02$
Algorithm	μ =0.01				
	CPU time	Opt value	Error with M in norm $\|\cdot\|_{\mathcal{F}}$	Error with A in norm $\|\cdot\|_{\mathcal{F}}$	$\|x\|_*$
Mosek	$1.09e+01$	$2.00e+00$	$1.86e-04$	$2.33e-04$	$2.00e+02$
ADMM	$6.68e-01$	$2.00e+00$	$1.86e-04$	$2.33e-04$	$2.00e+02$
ADMM linear time	$6.65e-01$	$2.00e+00$	$2.50e-04$	$2.82e-04$	$2.00e+02$
ADMM prototype	$3.11e-01$	$2.00e+00$	$1.86e-04$	$2.33e-04$	$2.00e+02$
Algorithm	μ =0.001				
	CPU time	Opt value	Error with M in norm $\|\cdot\|_{\mathcal{F}}$	Error with A in norm $\|\cdot\|_{\mathcal{F}}$	$\|x\|_*$
Mosek	$1.19e+01$	$2.00e-01$	$1.86e-05$	$2.37e-05$	$2.00e+02$
ADMM	$1.31e+00$	$2.00e-01$	$2.99e-05$	$1.49e-02$	$2.00e+02$
ADMM linear time	$6.91e-01$	$2.00e-01$	$2.50e-05$	$2.91e-05$	$2.00e+02$
ADMM prototype	$2.56e-01$	$2.00e-01$	$1.86e-05$	$2.33e-05$	$2.00e+02$

Table 12: Numerical result on example 2 with different μ

References

[1] Petros Drineas, Ravi Kannan, and Michael W Mahoney. Fast monte carlo algorithms for matrices ii: Computing a low-rank approximation to a matrix. *SIAM Journal on computing*, 36(1):158–183, 2006.

[2] Gene H Golub and Charles F Van Loan. *Matrix computations*. JHU press, 2013.

[3] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.