SoftwareX 23 (2023) 101500

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

GPU accelerated singular value thresholding

Xiaotong Li, Karel Adámek, Wes Armour *

University of Oxford, Oxford, UK

ARTICLE INFO

Article history:
Received 31 January 2023
Received in revised form 29 June 2023
Accepted 7 August 2023

Keywords:
Singular value thresholding (SVT)
GPU
Singular value decomposition (SVD)
Matrix completion

ABSTRACT

Matrix completion (MC) is widely used in machine learning and signal processing to fill in the missing data of an incomplete observation matrix. Singular value thresholding (SVT) is one of the most popular algorithms among numerous MC methods. A Python-based GPU-accelerated SVT software is presented in this paper. It is a user-friendly software package to minimise the nuclear norm with high accuracy and high computational efficiency. Its architecture and functionalities are illustrated, followed by a demonstration on how to use this software. Two examples, image inpainting and traffic sensing, are shown to illustrate potential applications of this software. Its impact on scientific and wider audiences is also analysed.

Code metadata

Current code version	v1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-23-00046
Code Ocean compute capsule	/
Legal Code License	CC BY 4.0
Code versioning system used	git
Computing platforms/Operating Systems	Linux
Software code languages, tools, and services used	Python (30.3%), C (25.6%), Cuda (23.4%), C++ (19.7%), Makefile (1.0%)
Compilation requirements, operating environments & dependencies	Please find information on https://github.com/egbdfX/gpuSVT
If available Link to developer documentation/manual	https://github.com/egbdfX/gpuSVT
Support email for questions	wes.armour@oerc.ox.ac.uk (corresponding author)

1. Motivation and significance

Matrix completion (MC) is a very promising method that is used extensively in machine learning to fill in the missing data of an incomplete observation matrix. It has become well-known since its application in the Netflix Prize problem, i.e., the movie-ratings matrix in recommender systems [1]. Currently, MC has been widely applied to many fields [2,3], such as image inpainting [4], magnetic resonance imaging [5], motion estimation [6], video surveillance and face recognition [7], computer vision [8,9], traffic sensing [10], synthetic aperture radar (SAR) imaging [11], Internet of Things (IoT) [12], and integrated radar and communications [13,14].

MC aims to restore a low-rank matrix which well approximates observed entries of the incomplete matrix. There are numerous MC methods [2,15], such as nuclear norm minimisation (e.g., nuclear norm relaxation, robust principal component analysis (RPCA) [16,17]), matrix factorisation [18], minimum rank approximation [19], l_p-norm minimisation [20], adaptive outlier pursuing [21], and so on. Singular value thresholding (SVT) [16], one of the most popular MC methods, is a nuclear norm relaxation algorithm [22].

Software packages, e.g., Templates for First-Order Conic Solvers (TFOCS) [23], could be used to carry out the MC by first-order methods. A nonnegative matrix factorization (NMF) package was proposed to perform nonnegative matrix completion from partial observations [24]. A Fortran-based software ID Version 0.4 [25] computed low-rank approximations by interpolative decompositions (IDs). The SVT can also be implemented by a MATLAB wrapper function *svt* [26]. However, it is known that the SVT process is computationally demanding due to the need to perform singular value decomposition (SVD).

Therefore, we developed a GPU accelerated SVT software written in Python to perform SVT faster. Comparing with other MC methods, the SVT algorithm has very good simplicity, robustness, scalability, and flexibility [15]. Comparing with other SVT

* Corresponding author.
E-mail address: wes.armour@oerc.ox.ac.uk (W. Armour).

https://doi.org/10.1016/j.softx.2023.101500

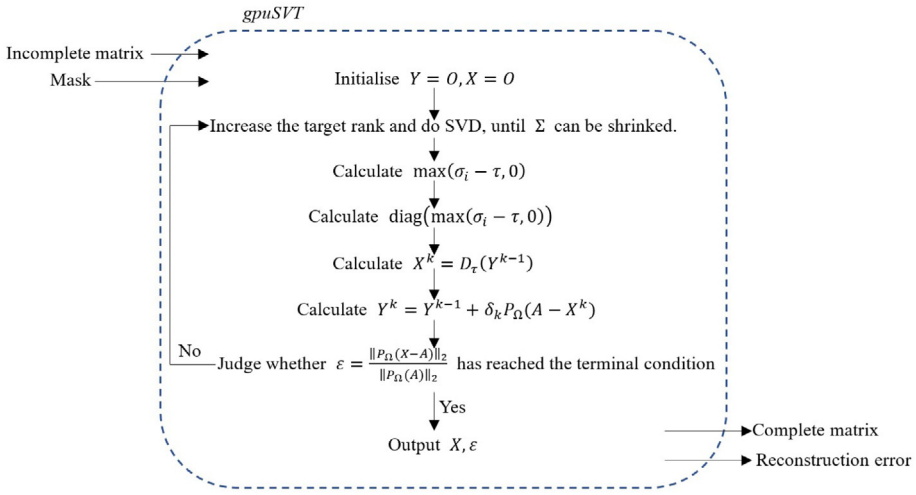

Fig. 1. Diagram of the architecture of **gpuSVT**.

software packages, our software enhances the computational speed. In this software, we wrote a python function *gpuSVT* based on [16] to implement SVT which calls a function *cuRandomSVD* that provides a Python interface for the function *cusolverDnXgesvdr* from the NVIDIA GPU accelerated library cuSOLVER. The function *cusolverDnXgesvdr* implements randomised k-SVD [27] with high accuracy. By using our software, the processing time can be reduced by more than 10 times for large matrices without changing the accuracy of the final results.

Section 2 describes the architecture and functionalities of this software. Illustrative examples in image inpainting and traffic sensing are demonstrated in Section 3. The impact of this software is analysed in Section 4. Section 5 presents our conclusions.

2. Software description

This section illustrates the architecture and functionalities of this software. Sample code snippets are also analysed in this section. The software is open source available on GitHub [28].

2.1. Software architecture

In this software, we use Python which is one of the most popular high-level programming languages. CuPy is used for the GPU-accelerated computing. The CUDA Toolkit library cuSOLVER is used for the decomposition of the input matrix.

The diagram of the architecture of our software is shown in Fig. 1. Incomplete matrix/matrices and the corresponding mask matrix/matrices are inputted. The mask matrix is set to distinguish between known and unknown data. The set of indices of the observed entries is denoted by Ω. For a certain position (i, j) in the image, if $(i, j) \in \Omega$, the mask there will be 1, denoting "known"; otherwise, the mask there will be 0, denoting "unknown". If a series of matrices are inputted, the third index can be introduced apart from the 2-dimensional image.

Mathematically, MC can be expressed as

$$\min_{X \in R^{m \times n}} \mathrm{rank}\,(X)\ s.t.\ X_{ij} = A_{ij},\quad (i,j) \in \Omega, \tag{1}$$

where A is the incomplete observed matrix and X is the restored matrix.

The rank minimisation described by Eq. (1) is an NP hard problem. In the SVT approach, the rank minimisation can be achieved by solving the problem expressed by

$$\min \|X\|_*\ s.t.\ P_\Omega\,(X) = P_\Omega\,(A)\,, \tag{2}$$

where $\|\bullet\|_*$ denotes the nuclear norm, and the mask operator P_Ω can be expressed by

$$P_\Omega\,(X)_{ij} = \begin{cases} X_{ij}, & (i,j) \in \Omega \\ 0, & otherwise \end{cases}. \tag{3}$$

The problem can then be solved iteratively [16],

$$\begin{cases} X^k = D_\tau\left(Y^{k-1}\right) \\ Y^k = Y^{k-1} + \delta_k P_\Omega\left(A - X^k\right) \end{cases}, \tag{4}$$

where δ_k $(k \geq 1)$ is the scalar step size and the soft-thresholding operator D_τ at level $\tau > 0$ is expressed by

$$D_\tau\,(Y) := UD_\tau\,(\Sigma)\,V^* = U\,\mathrm{diag}\,(\max\,(\sigma_i - \tau, 0))\,V^* \tag{5}$$

with the r-rank matrix $Y = U\Sigma V^*$ and $\Sigma = \mathrm{diag}\left(\{\sigma_i\}_{i \in [1,r]}\right)$. The matrix X is low-rank and the matrix Y is sparse. The iterative process will end when the reconstruction error ε reaches the terminal condition defined by the user. The complete matrix X and the reconstruction error ε are outputted.

The SVD of the matrix Y is the most time-consuming part of the SVT calculation. To calculate SVD we use the Python function *cuRandomSVD* that uses NVIDIA cuSOLVER library to calculate randomised SVD. There are four inputs in the *cuRandomSVD* function, i.e., the matrix to be decomposed, target rank, oversampling factor (used to guarantee sufficient coverage), and the number of iterations.

2.2. Software functionalities

The main function of this software is

[X, recon_error] = ***gpuSVT***(**A,mask**),

where **X** is the restored matrix, **A** is the incomplete observed matrix, **recon_error** is the reconstruction error ε, and **mask** is the mask matrix indicating known and unknown elements.

The goal of this software is to complete the missing information in the input by doing SVT. An incomplete matrix and the corresponding mask matrix are needed in each pair of inputs, and the output will be the complete matrix. Multiple pairs of inputs can be inputted into the interface of the software at the same time. It will then perform SVT on them separately.

Generally, the function *gpuSVT* can be used to minimise the nuclear norm in applications including, but not limited to, matrix completion and matrix regression. The soft-thresholding level τ, the scalar step size δ, and the terminal conditions (target

Table 1
Structure of the example.

	Item	Description
Input	ini.mat	Accurate value (0 or 1) of the first element of the mask.
	pyinp.mat	Incomplete matrix.
	pymask.mat	Mask matrix.
Script	gpuSVT_example.py	Python code.
Output	nnR_gpuSVT0.mat	Complete matrix of the first image.
	nnR_gpuSVT1.mat	Complete matrix of the second image.
	recon_error_gpuSVT0.mat	Reconstruction error for the completion of the first image.
	recon_error_gpuSVT1.mat	Reconstruction error for the completion of the second image.

reconstruction error or maximum number of iterations) can be set by the users.

The function *cuRandomSVD* can be used to perform randomised SVD with GPU acceleration. The target rank, the oversampling factor, and the number of iterations can be set by the users.

2.3. Sample code snippets analysis

The software **gpuSVT** is open source available on GitHub page [28]. The folder *example* on the GitHub page gives a quick demonstration on how to use this software.

To use our software,[1] the programme can be built with

```
conda create –n SVTexample python=3.9
conda activate SVTexample
cd /path/to/src
make
pip install cupy–cuda
pip install scipy
pip install mat4py
```

Note that the software requires CUDA package and cuSOLVER library installed.

The *example* folder includes a programme file (*gpuSVT_example.py*) and three example input files (*ini.mat*, *pyinp.mat*, *pymask.mat*), as shown in Table 1. The inputs are MATLAB files. The data *pyinp.mat* is the incomplete matrix, where the first dimension is the number of images in the input data cube, the second and the third dimensions are the 2-dimensional incomplete matrix. The data *pymask.mat* is the mask matrix corresponding to the incomplete matrix. It is worth noting that the first element of the mask is set as 0.5 instead of 0 or 1 so that the Python will recognise the data type as double instead of uint8. The data *ini.mat* includes the accurate value (0 or 1) of the first element of the mask, to rectify the mask matrix. In the example, the incomplete input matrix are constructed by images with salt-and-pepper noise. Two pairs of incomplete images and their masks are included in the example inputs, where the number of pairs can be adjusted by users' requirement.

After running the programme,[2] output files with the names prefixed by *nnR* are the complete matrices which are all 2-dimensional. The output files with the names prefixed by *recon_error* are the reconstruction errors varying with increasing iterations. Multiple complete matrices and their corresponding reconstruction errors will be outputted separately.

3. Illustrative examples

To illustrate the use and performance of our software, two applications, i.e, image inpainting and traffic sensing, are demonstrated as examples.

[1] We run the programme on Ubuntu 20.04.4.

[2] The way to use the example is shown on the GitHub page.

3.1. Image inpainting

The software can be applied to restored images with salt-and-pepper noise mask or character mask. It can also be used to remove objects (e.g., human) from scenery.

The incomplete images with salt-and-pepper noise are generated by using MATLAB function *imnoise*, where the noise density is set as 0.05. An example is shown in Fig. 2(a)–(d), including input incomplete image, mask, reconstructed complete image, and reference image. In addition, the incomplete images with floating characters are generated by randomly typing characters onto the images. An example of the result of applying our software to this type of incomplete image is shown in Fig. 2(e)–(h).

Herein, Structural SIMilarity (SSIM) index [30–32] is used to assess the similarity between the reconstructed image and the corresponding reference image. If the SSIM between reference and reconstructed images approaches 1, this indicates that the reconstructed image restores the features in the reference image well. Another way to quantitatively assess the quality of the completion is by comparing the masked part of the reconstructed image and the masked part of the reference image, as shown in Fig. 2(i) and (j). Low-Information Similarity Index (LISI) [33] can be used to compare the similarity between the masked parts, for the reason that the masked parts are low-information with large proportion of missing data. If we use SSIM to assess them, the SSIM will always be close to 1 indicating that the inputs are very similar. It is more convincing to carry out the assessment by LISI, since LISI is intensity-sensitive and especially useful to assess the similarity between the informative parts in low-information images. When both the SSIM and the LISI approach to 1, it demonstrates that the reconstructed image restores the features in the reference image well.

To compare the processing time between the CPU based SVT code and the GPU version, different sizes of incomplete images are used as the inputs. ImageJ [34] is used to resize the input images. The results are shown in Table 2. SVT runs 10,000 iterations for each input. In order to measure the processing time we have averaged and calculated the standard deviation from three runs for every input.

The processing time for image inpainting is plotted in Fig. 3. It can be easily seen from the results that the processing time can be greatly reduced by using our software. For very small images (e.g., 128x72), the processing time of the GPU code will be longer than the CPU code due to the transmission of data to the GPU over the PCIe bus. But for larger images, the advantage of the GPU code is more obvious.

In addition, the GPU-accelerated SVT can also be used to remove certain objects from an image; in other words, it can complete the background of an image. Take the images in the top row of Fig. 4 as examples. The bird/human in these images can be masked by the masks shown in the middle row and then completed by our software to obtain the bottom row of Fig. 4.

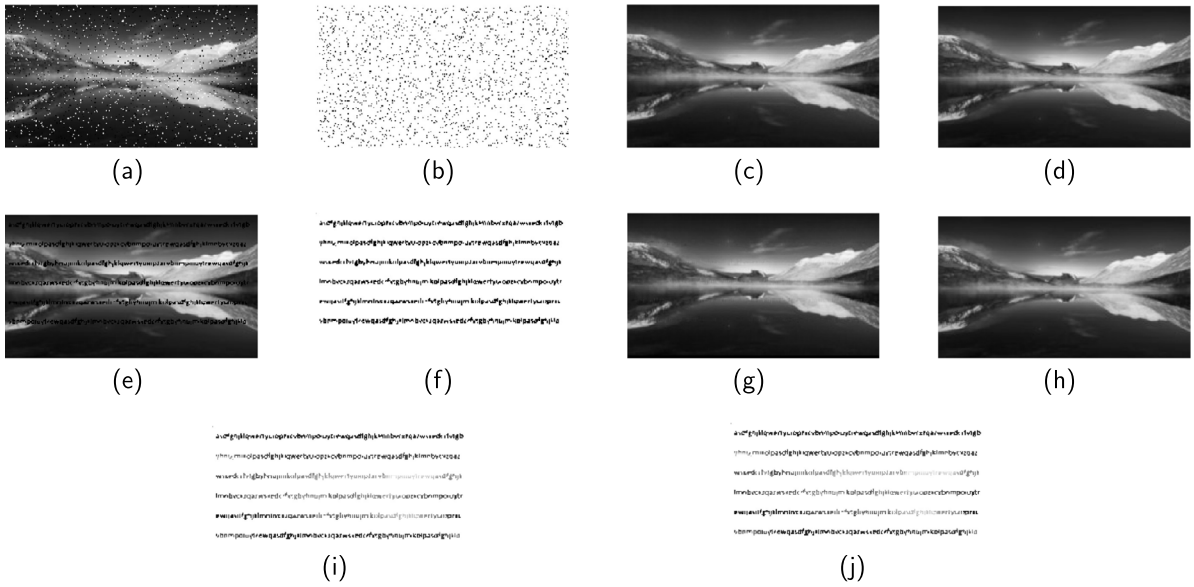

Fig. 2. SVT on (a) an incomplete image with salt-and-pepper noise or (e) an incomplete image floating with characters, where (b) and (f) show the masks of them, respectively. For the salt-and-pepper noise group, the SSIM between (c) the reconstructed complete image and (d) the reference image is 0.9931. For the characters group, the SSIM between (g) the reconstructed complete image and (h) the reference image is 0.9854. To compare the masked parts in (i) the reconstructed image and (j) the reference image, the SSIM between them is 0.9989, whereas the LISI between them is 0.9712. Since both the SSIM and the LISI approach 1, it can be revealed that the reconstructed image restores the features in the reference image well. Note that the reference image is generated by resizing and greying the original image [29]. The incomplete image is generated by adding in salt-and-pepper noise or characters on the reference image.

Table 2
Processing time for image inpainting.

Size	Mask type	SVT processing time (sec)		Speed up
		With CPU	With GPU	
128 × 72	Salt-and-pepper	22.8913 (0.4121)	75.3691 (0.0260)	0.3037
	Character	24.7087 (0.2271)	77.8177 (0.0331)	0.3175
256 × 144	Salt-and-pepper	735.9870 (11.8936)	130.2787 (0.1239)	5.6493
	Character	609.9996 (89.5186)	131.1266 (0.0140)	4.6520
512 × 288	Salt-and-pepper	1794.6431 (125.2505)	130.2413 (0.1896)	13.7794
	Character	1901.4932 (153.3767)	131.2422 (0.1032)	14.4884
1024 × 576	Salt-and-pepper	4463.3578 (99.6356)	324.1538 (0.0762)	13.7693
	Character	4514.2794 (58.7466)	327.1537 (0.2166)	13.7986
1280 × 720	Salt-and-pepper	5662.3376 (157.9425)	346.6708 (0.0575)	16.3335
(HD)	Character	5503.4989 (150.5578)	341.0089 (0.0869)	16.1389
1920 × 1080	Salt-and-pepper	4554.3957 (114.2015)	314.7239 (0.1561)	14.4711
(FHD)	Character	4318.1432 (127.8061)	335.4067 (0.1426)	12.8743
2560 × 1440	Salt-and-pepper	6692.7427 (225.9960)	595.6270 (0.5137)	11.2365
(QHD)	Character	6595.1039 (298.2552)	534.5391 (0.3339)	12.3379
3840 × 2160	Salt-and-pepper	13078.5552 (221.3513)	678.0495 (0.0420)	19.2885
(UHD, 4K)	Character	12905.5984 (103.1753)	697.1325 (0.1761)	18.5124
7680 × 4320	Salt-and-pepper	59546.7128 (138.6502)	2114.2858 (0.3037)	28.1640
(8K)	Character	58619.8915 (78.2195)	2155.1204 (0.3606)	27.2003

Note: The processing time is represented by "Average (Standard Deviation)".

3.2. Traffic sensing

In traffic sensing systems, probe vehicles are used to sense the traffic information [10]. However, some data are missing due to uneven distribution of probe vehicles or measurement error. The missing traffic data can be estimated by MC methods. We apply our software to the traffic data of Interstate 205 (Oregon–Washington) Southbound (I-205 SB) [37] to complete the missing data.

Five groups of data are used in this illustration. They are the traffic data for the five weeks commencing 11 Apr 2022, 18 Apr 2022, 25 Apr 2022, 2 May 2022, and 9 May 2022. Table 3 illustrates the processing results by doing SVT with and without GPU acceleration on the traffic data of the five weeks. Both results reach the reconstruction error of 0.01. As shown in the table, the GPU SVT can be quicker or slower than the CPU SVT, due to the fact that the size of the data set in this illustration is very small.

Table 3
Processing time of completing the traffic data.

Week commencing	Percentage of missing data	Processing time (sec) With CPU	With GPU	Speed up
11 Apr 2022	4.09%	0.2471	1.0210	0.2420
18 Apr 2022	3.96%	0.2452	0.9331	0.2628
25 Apr 2022	3.91%	1.9470	0.9327	2.0875
2 May 2022	3.88%	0.4080	0.9810	0.4159
9 May 2022	3.87%	2.8137	0.9897	2.8430

This also aligns with the results of very small images which have been discussed in Section 3.1.

Among the specifications (start time, resolution, detector ID, speed, volume, occupancy, sample size, delay, travel time, vehicle hours travelled, and vehicle miles travelled) of the highway data, we analyse the fields of speed and volume as examples [38]. The

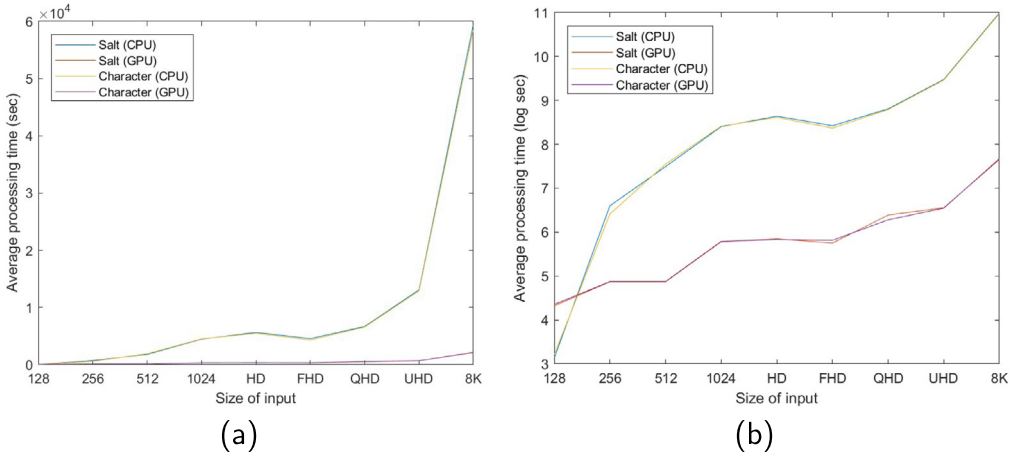

Fig. 3. Average processing time of image inpainting on different sizes of inputs, with (a) linear and (b) logarithmic representations. According to (b), the processing time of the GPU version is approximately exponential with the increase in the size of input, but much quicker than the CPU version, except for inputs with very small sizes.

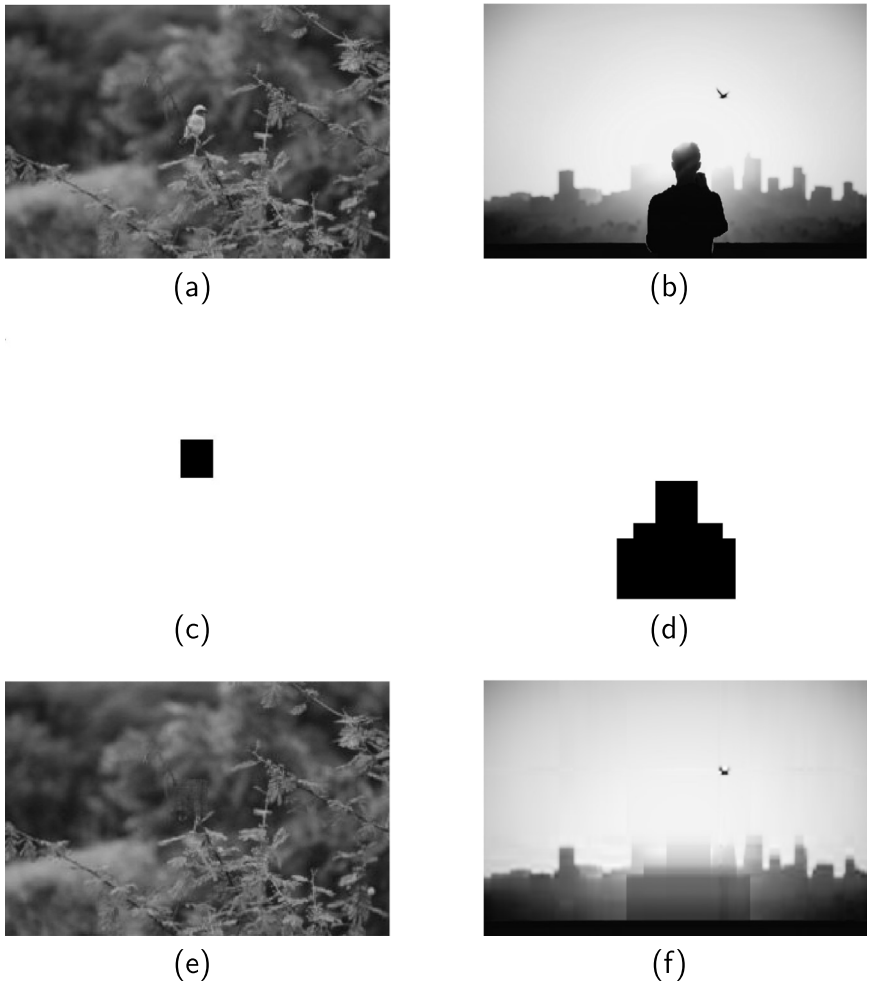

Fig. 4. SVT to remove certain objects from the images. (Top row) The reference images with bird/human, (middle row) their masks, and (bottom row) the corresponding reconstructed images without the bird/human. The reference images are generated by resizing and greying the original images [35,36].

fields "speed" and "volume" show the average speed and the number of vehicles (per hour) passing the detector. Since there is no gold standard data of the real measurement to compare with, to prove the credibility of the completion results, the tolerance level of the amount of unknown data in the input needs to be determined. We can calculate the tolerance level by randomly masking $x\%$ of known data, and then completing it, and finally comparing the reconstructed complete matrix with the reference matrix. The mean and standard deviation of the difference between the reconstructed matrix and the reference matrix for the traffic data of the week commencing 9 May 2022 are shown in Fig. 5 as an example. If the tolerance level is set as 10% difference

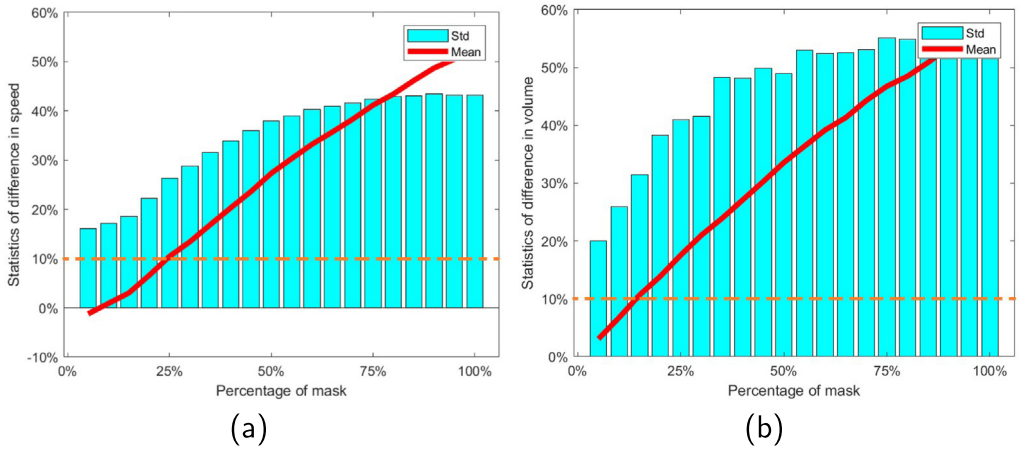

Fig. 5. Statistics (mean and standard deviation) of the difference between the reconstructed matrix and the reference matrix for the randomly masked traffic data of (a) speed and (b) volume in the week commencing 9 May 2022 are shown as examples. This figure aims to find the maximum acceptable amount of unknown data. With the increase in the percentage of mask, the mean of difference between the reconstructed result and reference data increase, with varying in standard deviation of the difference. The orange dash lines show the tolerance level of the difference, which is set as 10%. The percentage of mask corresponding to the cross point of the tolerance level and the mean of difference shows the acceptable amount of unknown data. According to the sub-figures, the minimum acceptable percentage of unknown data is 15%, among the data of speed and volume measured during different time periods (only the data for one week is shown here as an example due to page limitations).

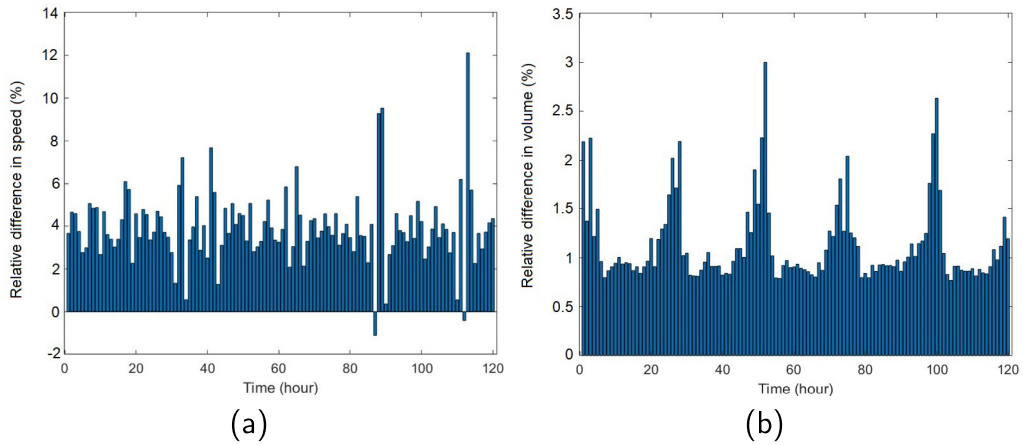

Fig. 6. Relative difference between the reconstructed matrix and the original matrix for the real traffic data of (a) speed and (b) volume in the week started by 00.00 on 9 May 2022 are shown as examples. The time axis is in hours. Only weekdays of each week are plotted and analysed. The relative differences show that regions of rush hour are completed correctly, and regions where data is completely missing cannot be completed because the algorithm will not work with no data.

on average (as shown in the orange dash lines in Fig. 5), the acceptable percentage of unknown data should be no more than 15% to ensure the tolerance. According to Table 3, the percentage of missing data in each week is lower than 15%, which indicates that the reconstructed results of the real data are credible.

Applying our software to the real data of the five weeks, the completed speed and volume are obtained for each week's data. In order to show the contribution of our software in a clearer way, relative differences are calculated to present the difference between the incomplete and the complete matrices.

The relative differences in speed and volume between the reconstructed matrices and the original matrices are shown in Fig. 6. The relative difference is calculated by $(out - in)/in$, where "*out*" is the average value of speed or volume measured by different probe vehicles in each time period (lasting for one hour) in the output of MC, and "*in*" is that average value in the input of MC. Only weekdays of each week are plotted and analysed to reveal the feature of rush hours. If the data of the traffic information is totally lost in a certain time period, the data in that time period cannot be completed even when using MC. Moreover, the characteristics of volume during rush hours become more obvious by doing MC.

4. Impact

The GPU-accelerated SVT software gives an quick and easy access for users to minimise the nuclear norm. In other words, the processing time for large input matrices has been greatly reduced by using this software and the software is easy to use.

Our software has impact on nuclear norm minimisation, which contributes to fields such as matrix completion and matrix regression. We have demonstrated the use of this software in image inpainting and traffic sensing; however, the software can also be applied to problems in fields such as SAR imaging and IoT. In addition, the accelerated SVD module within this software can also be imported by other programmes, which helps to calculate the randomised SVD. Our software greatly increases the computational speed of SVT, as shown in Fig. 3.

This software also has impact on wider audiences. The software is written in Python which is not only popular for scientific computation but is also easily used by broader audiences.

5. Conclusions

A GPU accelerated SVT software package, which is written in Python, is presented in this paper. It can carry out the nuclear norm minimisation with high computational speed and high accuracy. It can be applied to various fields including, but not limited to, matrix completion. Moreover, the GPU accelerated SVD module within this SVT package is also importable in other programmes.

This software package greatly increases the computational efficiency, comparing with previous work, which is useful in the big data era. It has impact on scientific studies, technical research, and wider audiences from different fields.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request

Acknowledgements

This work has received support from STFC Grant (ST/T000570/1).

References

[1] Koren Y, Bell R, Volinsky C. Matrix factorization techniques for recommender systems. Computer 2009;42(8):30–7.
[2] Li X, Huang L, So H, Zhao B. A survey on matrix completion: Perspective of signal processing. 2019.
[3] Peng Y, Suo J, Dai Q, Xu W, Lu S. Robust image restoration via reweighted low-rank matrix recovery. In: Proceedings of the 20th anniversary international conference on multimedia modeling (Part I). Springer; 2014, p. 315–26.
[4] Bertalmio M, Sapiro G, Caselles V, Ballester C. Image inpainting. In: Proceedings of the 27th annual conference on computer graphics and interactive techniques. USA: ACM Press; 2000, p. 417–24.
[5] Hu Y, Liu X, Jacob M. A generalized structured low-rank matrix completion algorithm for MR image recovery. IEEE Trans Med Imaging 2019;38:1841–51.
[6] Li K, Dai Q, Xu W, Yang J, Jiang J. Three-dimensional motion estimation via matrix completion. IEEE Trans Syst Man Cybern B Cybern 2012;42(2):539–51.
[7] Candes E, Li X, Ma Y, Wright J. Robust principal component analysis? J ACM 2011;58(3).
[8] Zhao K, Zhang Z. Successively alternate least square for low-rank matrix factorization with bounded missing data. Comput Vis Image Underst 2010;114(10):1084–96.
[9] Lu X, Gong T, Yan P, Yuan Y, Li X. Robust alternative minimization for matrix completion. IEEE Trans Syst Man Cybern B Cybern 2012;42(3):939–49.
[10] Du R, Chen C, Yang B, Guan X. VANET based traffic estimation: A matrix completion approach. In: 2013 IEEE global communications conference. 2013, p. 30–5.
[11] Yang D, Liao G, Zhu S, Yang X, Zhang X. SAR imaging with undersampled data via matrix completion. IEEE Geosci Remote Sens Lett 2014;11(9):1539–43.
[12] Nguyen L, Kim J, Kim S, Shim B. Localization of IoT networks via low-rank matrix completion. IEEE Trans Commun 2019;67(8):5833–47.
[13] Li B, Petropulu A, Trappe W. Optimum co-design for spectrum sharing between matrix completion based MIMO radars and a MIMO communication system. IEEE Trans Signal Process 2016;64(17):4562–75.
[14] Sun S, Petropulu A, Bajwa W. Target estimation in colocated MIMO radar via matrix completion. In: 2013 IEEE international conference on acoustics, speech and signal processing. 2013, p. 4144–8.
[15] Bouwmans T, Aybat N, Zahzah E. Handbook of robust low-rank and sparse matrix decomposition: Applications in image and video processing. CRC Press; 2016.
[16] Cai J, Candes E, Shen Z. A singular value thresholding algorithm for matrix completion. SIAM J Optim 2010;20(4):1956–82.
[17] Lin Z, Chen M, Ma Y. The augmented Lagrange multiplier method for exact recovery of corrupted low-rank matrices. 2013.
[18] Wen Z, Yin W, Zhang Y. Solving a low-rank factorization model for matrix completion by a nonlinear successive over-relaxation algorithm. Math Program Comput 2012;4(4):333–61.
[19] Lee K, Bresler Y. ADMiRA: Atomic decomposition for minimum rank approximation. IEEE Trans Inform Theory 2010;56(9):4402–16.
[20] Zeng W, So H. Outlier-robust matrix completion via l_p-minimization. IEEE Trans Signal Process 2018;66(5):1125–40.
[21] Yan M, Yang Y, Osher S. Exact low-rank matrix completion from sparsely corrupted entries via adaptive outlier pursuit. J Sci Comput 2013;56(3):433–96.
[22] Cai J-F, Wei K. Chapter 2 - exploiting the structure effectively and efficiently in low-rank matrix recovery. In: Kimmel R, Tai X-C, editors. Processing, analyzing and learning of images, shapes, and forms: Part 1. Handbook of numerical analysis, vol. 19, Elsevier; 2018, p. 21–51.
[23] Becker S, Candes E, Grant M. Templates for convex cone problems with applications to sparse signal recovery. Math Program Comput 2011;3:165–218.
[24] Xu Y, Yin W, Wen Z, Zhang Y. An alternating direction algorithm for matrix completion with nonnegative factors. Front Math China 2012;7(2):365–84.
[25] Martinsson P-G, Rokhlin V, Shkolnisky Y, Tygert M. ID: A software package for low-rank approximation of matrices via interpolative decompositions, Version 0.4. 2014, http://tygert.com/id_doc.4.pdf.
[26] Li C, Zhou H. svt: Singular Value Thresholding in MATLAB. J Stat Softw 2017;81.
[27] Halko N, Martinsson P-G, Tropp JA. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. SIAM Rev 2011;53(2):217–88.
[28] Li X, Adamek K, Armour W. gpuSVT. https://github.com/egbdfX/gpuSVT.
[29] Araujo C. Lake 8k ultra HD wallpaper. https://pixabay.com/photos/lake-landscape-nature-5211977/.
[30] Wang Z, Bovik A, Sheikh H, Simoncelli E. Image quality assessment: From error visibility to structural similarity. IEEE Trans Image Process 2004;13(4).
[31] Wang Z, Bovik A, Sheikh H. Structural similarity based image quality assessment. In: Wu H, Rao K, editors. Digital video image quality and perceptual coding, Marcel Dekker series in signal processing and communications. 2005.
[32] Brunet D. A study of the structural similarity image quality measure with applications to image processing [Ph.D. thesis], University of Waterloo; 2012.
[33] Li X, Armour W. Intensity-sensitive similarity indexes for image quality assessment. In: 2022 26th International conference on pattern recognition. 2022, p. 1975–81.
[34] Fiji - ImageJ Wiki. https://imagej.net/software/fiji/.
[35] More N. File:Baya weaver bird Dehu (22) 16.jpg. https://commons.wikimedia.org/wiki/File:Baya_Weaver_bird_Dehu_(22)_16.jpg.
[36] Dumlao N. Using people as a subject seen through the sun. https://unsplash.com/photos/EwaV3HjwmUo.
[37] Data of highways on PORTAL. https://new.portal.its.pdx.edu/highways/.
[38] PORTAL: User documentation. https://adus.github.io/portal-documentation/documents/downloads/.