DOI: 10.20535/kpisn.2025.2.331279
UDC 519.7

Vadim V. Romanuke*
Vinnytsia Institute of Trade and Economics of State University of Trade and Economics, Vinnytsia, Ukraine
*corresponding author: romanukevadimv@gmail.com

TALL ARRAY METHOD EFFICIENCY IN DATASET DIMENSIONALITY REDUCTION BY PRINCIPAL COMPONENT ANALYSIS

Background. Exploratory data analysis has been extensively growing since the early 2000s. As of 2025, most real-practice datasets are classified as Big Data. The Big Data analytics workflow includes the data preprocessing step, which is the starting point of Big Data computational handling. At this step, the data are tried to get simplified as much as possible. The main paradigm is dimensionality reduction allowing simplifying and visualizing high-dimensional datasets. Principal component analysis (PCA) is a linear dimensionality reduction technique. The PCA can be sped up by applying Tall Arrays, if the data are stored on disk. The Tall Array PCA (TAPCA) computes principal components incrementally using a divide-and-conquer strategy.
Objective. The paper aims to determine when the TAPCA is factually efficient for dimensionality reduction. There are two numeric types to be studied: double and single precision.
Methods. To achieve the said objective, random large datasets are generated as matrices of a specified numeric type. Then computational time of the ordinary MATLAB PCA applied to generated matrices is measured. Next, computational time of converting in-memory arrays (generated matrices) into tall arrays is measured. Computational time of the TAPCA applied to those generated matrices, to which the PCA is applied before, is measured as well.
Results. A comparative analysis of the averaged computational times reveals that computational time complexity of both the PCA and TAPCA is rather polynomial than strictly quadratic or cubic. There is a nearly-hyperbolic margin, which alternatively could be called the TAPCA efficiency threshold, in a plane of the number of dataset observations and the number of dataset features, by which the TAPCA and the ordinary PCA take approximately the same time to compute principal components.
Conclusions. In computing principal components for dimensionality reduction of large datasets stored on disk, the Tall Array method becomes efficient by two parallel processor workers if a dataset has at least 5 to 6 million entries. The Tall Array method is more efficient on datasets with double precision whose efficiency threshold is nearly 6 million entries, whereas the efficiency threshold for datasets with single precision is between 5 to 15.2 million entries.
Keywords: dimensionality reduction; principal component analysis (PCA); Tall Arrays; efficiency threshold; double precision; single precision.

Introduction

Exploratory data analysis has been raised into likely the vastest and deepest domain encompassing various scientific fields of information technology, computer science, and applied mathematics for engineering, econometrics, sociology, bioinformatics, etc. [1, 2]. Meanwhile, datasets to be explored have been extensively or even exponentially growing since early 2000s. As of 2025, most real-practice datasets are classified as Big Data [3, 4]. The Big Data analytics workflow includes the data preprocessing step, which is the starting point of Big Data computational handling [5, 6]. At this point, before the most valuable and decisive Big Data statistics are computed, like mean and standard deviation values, the data are tried to get simplified as much as possible [7, 8]. The main paradigm is dimensionality reduction which allows simplifying and visualizing high-dimensional datasets.

Principal component analysis (PCA) is a linear dimensionality reduction technique, by which the dataset is linearly transformed into a new coordinate system such that the directions (principal components) sorted in descending order capture the largest variation in the data [3, 9]. The principal compo-

Пропозиція для цитування цієї статті: В.В. Романюк, "Ефективність методу Tall Array у зниженні розмірності наборів даних на основі методу головних компонентів", *Наукові вісті КПІ*, № 2, с. 18–29, 2025. doi: 10.20535/kpisn.2025.2.331279

Offer a citation for this article: Vadim Romanuke, "Tall Array method efficiency in dataset dimensionality reduction by principal component analysis", *KPI Science News*, no. 2, pp. 18–29, 2025. doi: 10.20535/kpisn.2025.2.331279

nents constitute an orthonormal basis in which different individual dimensions of the data are linearly uncorrelated [10, 11]. If the dataset is an $M \times N$ matrix $\mathbf{X} = [x_{mn}]_{M \times N}$ representing M observations of N-dimensional objects, where usually $M > N$, the PCA returns N principal components that are unit vectors [9]. The n-th vector, $n = \overline{2, N}$, is the direction of a line that best fits the data while being orthogonal to the first $n - 1$ vectors [11]. A best-fitting line minimizes the average squared perpendicular distance from the points to the line [3, 12].

The first principal component of a dataset with N variables represented by matrix $\mathbf{X} = [x_{mn}]_{M \times N}$ is constructed as a linear combination of the original variables, and it explains the most variance of the data. The first principal component is equivalently defined as a direction that maximizes the variance of the projected data. The second principal component explains the most variance in what is left once the effect of the first principal component is removed. Thus every next principal component explains less variance. Together the N principal components explain all the variance. The n-th principal component, $n = \overline{2, N}$, can be taken as a direction orthogonal to the first $n - 1$ principal components that maximizes the variance of the projected data. Alternatively, the PCA is defined as an orthogonal linear transformation on a real inner product space that transforms the data to a new coordinate system such that the greatest variance by some scalar projection of the data comes to lie on the first coordinate (being the first principal component), the second greatest variance lies on the second coordinate, and so on.

In practice, if the first N^* principal components explain sufficiently high amount of variance (say, typically, 90 % or above), $N^* \in \{\overline{1, N-1}\}$, the remaining $N - N^*$ principal components are ignored. Thus the initial dataset $\mathbf{X}$ is simplified to a dataset represented as an $M \times N^*$ matrix whose m-th row is a vector of values of the first N^* principal components calculated by plugging the entries of the m-th row of matrix $\mathbf{X}$ into the respective linear combinations of the N original variables. Clearly, the simplified dataset is easily visualized if $N^* = 2$ or $N^* = 3$.

If data are frequently updated (e. g., by adding new observations), it is a challenge to compute principal components timely, without delays or excessive memory occupation. The reason is the dataset is either constantly enlarged or updated, and thus the PCA slows down. The PCA can be sped up by applying Tall Arrays, if the data are stored on disk. Tall Arrays (operators and functions using the Tall Array approach) are a feature used in MATLAB for working with datasets that are too large to fit in memory [13, 14]. They allow performing computations on large datasets using familiar MATLAB functions and syntax without needing to manage low-level data processing tasks like chunking, paging, or out-of-core processing [14]. Tall Arrays handle large datasets that exceed the available memory by keeping only a relatively small portion of the data in memory at a time [13, 15]. This is called deferred evaluation, by which operations on an array are not computed immediately but are successively recorded and only executed when the data is explicitly requested [14, 16].

The Tall Array PCA (TAPCA) computes principal components incrementally using a divide-and-conquer strategy [17, 18]. First, the data are standardized by processing it in chunks. Then the covariance matrix is computed and normalized. This is done incrementally by summing outer products of chunks of data. The principal components are finally obtained by performing eigenvalue decomposition of the covariance matrix.

For standardizing each column of matrix $\mathbf{X}$, the mean and standard deviation of the matrix $\mathbf{X}$ column are computed as follows. The tall array (capitalization is used when the Tall Array approach is meant itself) representing the matrix $\mathbf{X}$ column is divided into smaller chunks that can fit into memory. The local sum, count of elements, and local sum of squares (sum of the squares of each element in the chunk) are calculated and stored temporarily for each chunk. And, once all chunks have been processed, the local sums and counts are aggregated to compute the global mean of the column as the ratio of the total sum of the local sums to the total number of elements. Then, in the second pass, for each chunk the local sum of squared differences from the global mean is calculated. These local sums are added up and the resulting global sum of squared differences is divided by the column length (i. e., the total sum of local counts) decreased by 1. The result is the global variance whose square root is the standard deviation. The column mean is subtracted from each entry of the column, whereupon every entry of the centered column is divided by its standard deviation.

Owing to the covariance matrix is smaller (its size is $N \times N$), the eigenvalue decomposition can be performed in memory. As for the rest, the TAPCA processes chunks sequentially, keeping memory usage relatively low while still being able to compute the correct coefficients of linear combinations of the original variables to return the principal components. Nevertheless, the sequential data chunking and processing may slow down computation of

principal components. Parallelization partially reduces the slowdown, but it depends on the dataset size and how it is chunked (although Tall Arrays chunk data automatically) [18, 19]. The number of parallel processor workers positively influences the TAPCA efficiency as well – Tall Arrays are generally expected to be more efficient at more parallel processor workers.

Problem Statement

The TAPCA is seemingly a very efficient method of dimensionality reduction, but there are two key aspects that must be taken into account. First, converting an in-memory (ordinary) array into a tall array takes some computational time, which ought to be added to the computational (operation) time taken by the TAPCA itself. Second, it is unclear when an in-memory array is regarded as large enough to apply the TAPCA rather than the (ordinary) PCA [6, 8, 12, 18]. This means that it is unclear when the TAPCA computes principal components faster than the (ordinary MATLAB) PCA does.

Therefore, the objective is to determine when the TAPCA is factually efficient for dimensionality reduction. The two numeric types to be studied are double and single precision. To achieve the objective, the following tasks are to be fulfilled:

1. To generate random large datasets as matrices of a specified numeric type.
2. To measure computational time of the ordinary MATLAB PCA applied to generated matrices.
3. To measure computational time of converting in-memory arrays (generated matrices) into tall arrays (which could be conditionally called tall matrices).
4. To measure computational time of the TAPCA applied to those generated matrices, to which the PCA is applied before.
5. To carry out a comparative analysis of the averaged computational times.
6. To discuss obtained results and findings from the comparative analysis.
7. To conclude on the TAPCA factual efficiency for dimensionality reduction of large datasets stored on disk.
8. To outline open questions and perspectives of further research.

Random Matrices

A random matrix $\mathbf{X} = \left[x_{mn}\right]_{M\times N}$ of a specified numeric type is generated by using the standard normal distribution having zero mean and unit variance [11, 20]. Hence, each entry x_{mn} in the matrix modeling a large dataset is a value of the normally distributed random variable with zero mean and unit variance. The numeric types to be studied are double and single precision for real numbers, so there will be two series of random datasets. For these two types the number of observations M is sequentially set to an element of set

$$\left\{\left\{10^2 \cdot k\right\}_{k=1}^{9}, \left\{10^3 \cdot k\right\}_{k=1}^{9}, \left\{10^4 \cdot k\right\}_{k=1}^{9}, \left\{10^5 \cdot k\right\}_{k=1}^{10}\right\} \quad (1)$$

and the number of initial variables (i. e., original features) N is sequentially set to an element of set

$$\left\{\overline{2, 100}\right\}. \quad (2)$$

At a given couple $\{M, N\}$ 100 random matrices $\mathbf{X} = \left[x_{mn}\right]_{M\times N}$ are generated at 100 different pseudo-random number generator seeds [12, 16, 18]. This is done to obtain statistically consistent and stable operation speed results upon averaging over those 100 generations.

Computational time

The computational time is measured on the dual-core processor Intel Core i5-7200U@2.50GHz in MATLAB R2018a. Firstly, computation of principal components is performed on the random matrix whose size is determined by the couple of integers from (1) and (2). The ordinary PCA computational time is denoted by $t_{\text{PCA}}\left(M, N, i\right)$, where i is the generation number at given $\{M, N\}$. Then, secondly, the random matrix is converted to a tall array, which can be called the random tall array (tall matrix). The time of the conversion is denoted by $\tau_{\text{TA}}\left(M, N, i\right)$. Thirdly, computation of principal components is performed on the random tall array. The TAPCA computational time is denoted by $t_{\text{TAPCA}}\left(M, N, i\right)$.

The averaged computational time of the PCA is

$$\overline{t}_{\text{PCA}}\left(M, N\right) = \frac{1}{100} \cdot \sum_{i=1}^{100} t_{\text{PCA}}\left(M, N, i\right) \quad (3)$$

at given $\{M, N\}$. The averaged computational time of the TAPCA without taking into account the array-to-tall-array conversion is

$$\overline{t}_{\text{TAPCA}}\left(M, N\right) = \frac{1}{100} \cdot \sum_{i=1}^{100} t_{\text{TAPCA}}\left(M, N, i\right). \quad (4)$$

If the conversion is regarded, then the TAPCA total time is calculated as

$$\overline{\theta}_{\mathrm{TAPCA}}\left(M,\,N\right)=\overline{\tau}_{\mathrm{TA}}\left(M,\,N\right)+\overline{t}_{\mathrm{TAPCA}}\left(M,\,N\right)$$
$$=\frac{1}{100}\cdot\sum_{i=1}^{100}\left[\tau_{\mathrm{TA}}\left(M,\,N,\,i\right)+t_{\mathrm{TAPCA}}\left(M,\,N,\,i\right)\right]. \quad (5)$$

Estimations (3)–(5) are fulfilled separately for double precision and single precision.

Analysis

The averaged computational time (3) of the PCA for double precision is shown as a mesh in Fig. 1, where some computational artifacts at

$$M\in\left\{7\cdot10^5,\,8\cdot10^5\right\}\ \text{and}\ N>30 \quad (6)$$

due to aliasing can be spotted. Obviously, the ravine at $M=9\cdot10^5$ cannot be a computational artifact. Computational time (3) is an increasing surface along each of its variables – both the numbers of observations and original variables. It roughly seems that surface (3) increases linearly. Nevertheless, the growth of computational time (3) is nonlinear, being close to quadratic or cubic. Averaged time $\overline{\tau}_{\mathrm{TA}}\left(M,\,N\right)$ of the array-to-tall-array conversion for double precision is shown in Fig. 2, where no significant trends are seen at all. Application of anti-aliasing techniques does not work to see any trend. Despite this, as the number of observations increases, the array-to-tall-array conversion time slowly grows.

Due to the array-to-tall-array conversion time for double precision does not exceed 27 milliseconds even for a million observations, averaged time (4) as a surface looks very resembling to TAPCA averaged time (5) that includes the array-to-tall-array conversion time (Fig. 3). Surface (5) in Fig. 3 has the same computational artifacts at (6) and the ravine at $M=9\cdot10^5$, although they appear to be a little bit softer. Some additional computational artifacts are visible at $M<10^5$ instead. At fewer original variables ($N=2$) and fewer observations ($M=7000$), a huge surge is seen. It is not caused by the array-to-tall-array conversion, though. Whereas the ordinary PCA handles a million-observation double-precision dataset with 100 variables within 9 seconds, the TAPCA takes no longer than just 1.9 seconds.

The averaged computational time (3) of the PCA for single precision shown as a mesh in Fig. 4 does not have any major computational artifacts. Compared to Fig. 1, this surface is much smoother. The surface increases similarly to that in Fig. 1. The

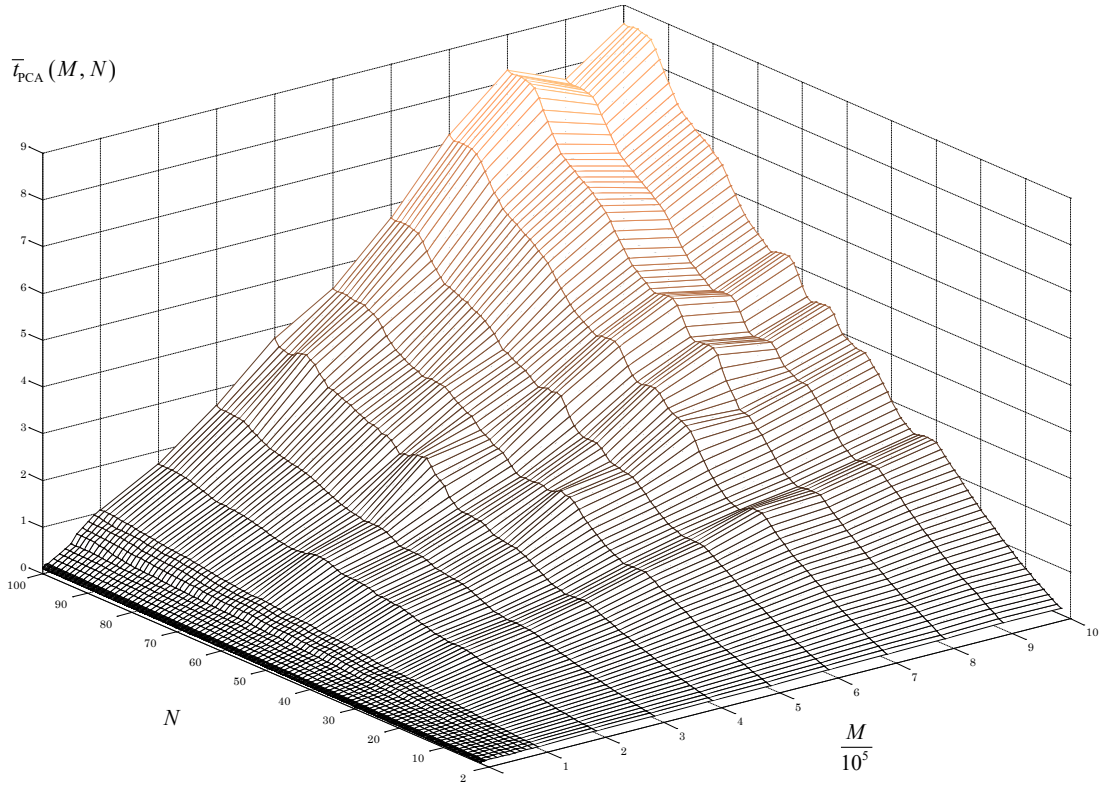

Fig. 1. The averaged computational time (3) of the PCA for double precision

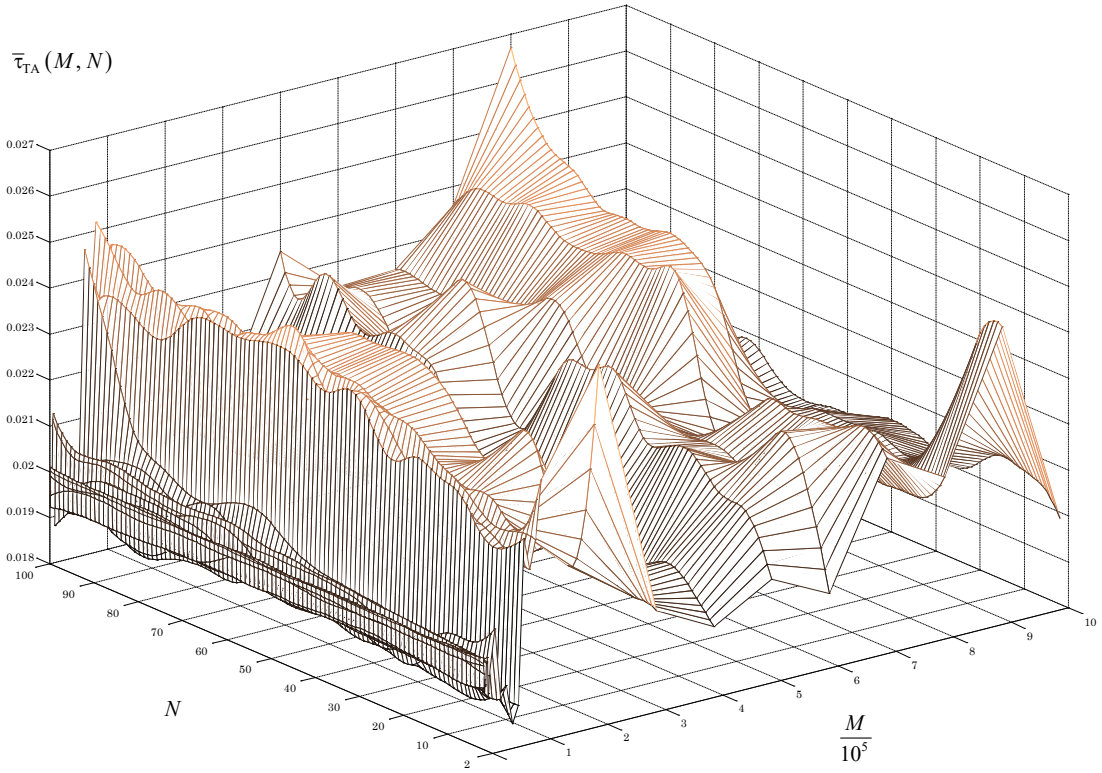

Fig. 2. The averaged time of the array-to-tall-array conversion for double precision

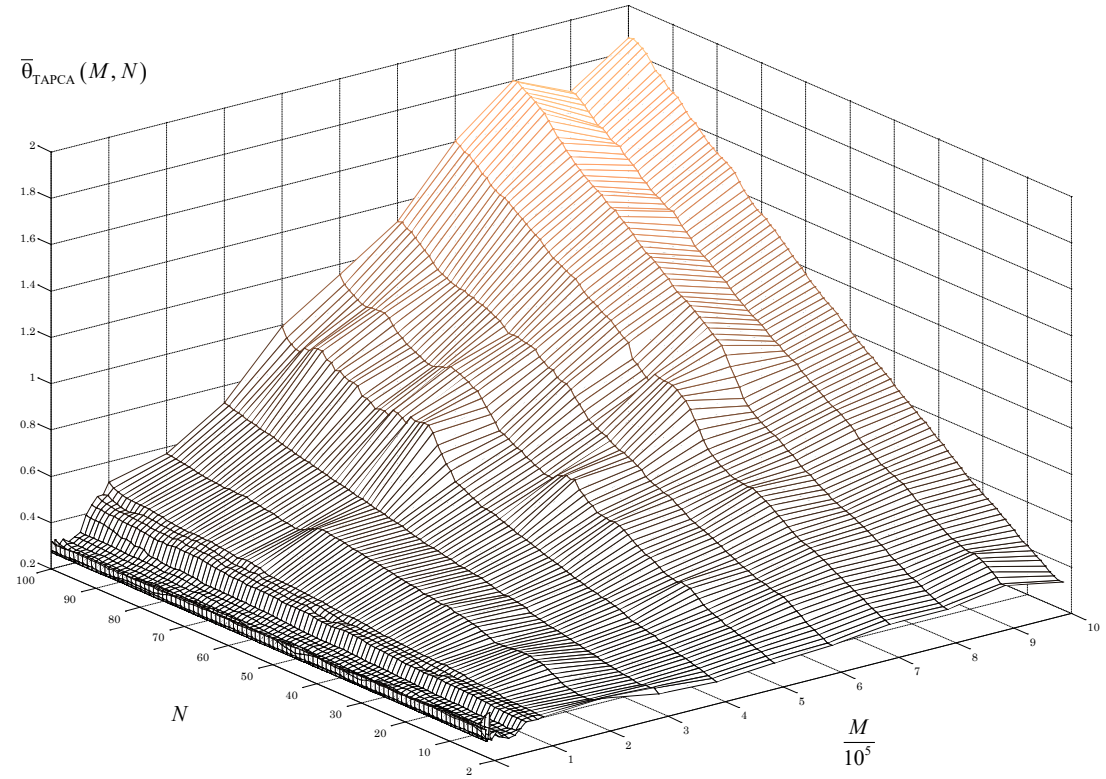

Fig. 3. The averaged computational time (5) of the TAPCA for double precision

ordinary PCA handles a million-observation double-precision dataset with 100 variables within 5.2 seconds. Overall, single-precision PCA is 1.5 to 2.1 times faster than double-precision PCA on average. Averaged time $\overline{\tau}_{\text{TA}}(M, N)$ of the array-to-tall-array conversion for single precision shown in Fig. 5 is faster as well, but the speedup is averagely 5 to 7 %. A single-precision matrix is converted into a tall array within 18 to 25 milliseconds. As the number of observations increases, the array-to-tall-array conversion time in Fig. 5 slowly grows, but the growth is more apparent than that in Fig. 2.

The averaged computational time (5) of the TAPCA for single precision is shown in Fig. 6. Compared to Fig. 3, the mesh is smoother at 10^5 observations and above, but it appears to have more non-linearities. Some computational artifacts are visible at fewer than 10^5 observations. The TAPCA takes no longer than just 1.3 seconds to compute 100 principal components of a million-observation single-precision dataset with 100 variables. Overall, single-precision TAPCA is 1.09 to 1.27 times faster than double-precision TAPCA on average.

Whichever precision or numeric type is, the TAPCA factual efficiency can be explored via ratio

$$\rho_{\text{TAPCA}}(M, N) = \frac{\overline{t}_{\text{PCA}}(M, N)}{\overline{\theta}_{\text{TAPCA}}(M, N)}. \qquad (7)$$

Clearly, the TAPCA is efficient if

$$\rho_{\text{TAPCA}}(M, N) > 1. \qquad (8)$$

The TAPCA efficiency for double precision is visualized in Fig. 7 presented as a plane view on the Cartesian product of sets (1) and (2) for abscissa and ordinate axes, respectively, where light color corresponds to (8), when the TAPCA is efficient; dark color is when the TAPCA is inefficient, i. e. inequality (8) is false. The TAPCA efficiency for single precision in Fig. 8 resembling that in Fig. 7 has a more dark color, i. e. single precision TAPCA has vaster area of inefficiency.

The TAPCA efficiency area exists beyond a nearly-hyperbolic margin (inside the hyperbola), both for double and single precision. Generally speaking, the TAPCA is inefficient at either fewer observations or fewer original variables (features). In more particular terms, it is inefficient at datasets having no more than about 50 000 observations. On the other side, the dataset with fewer than 10

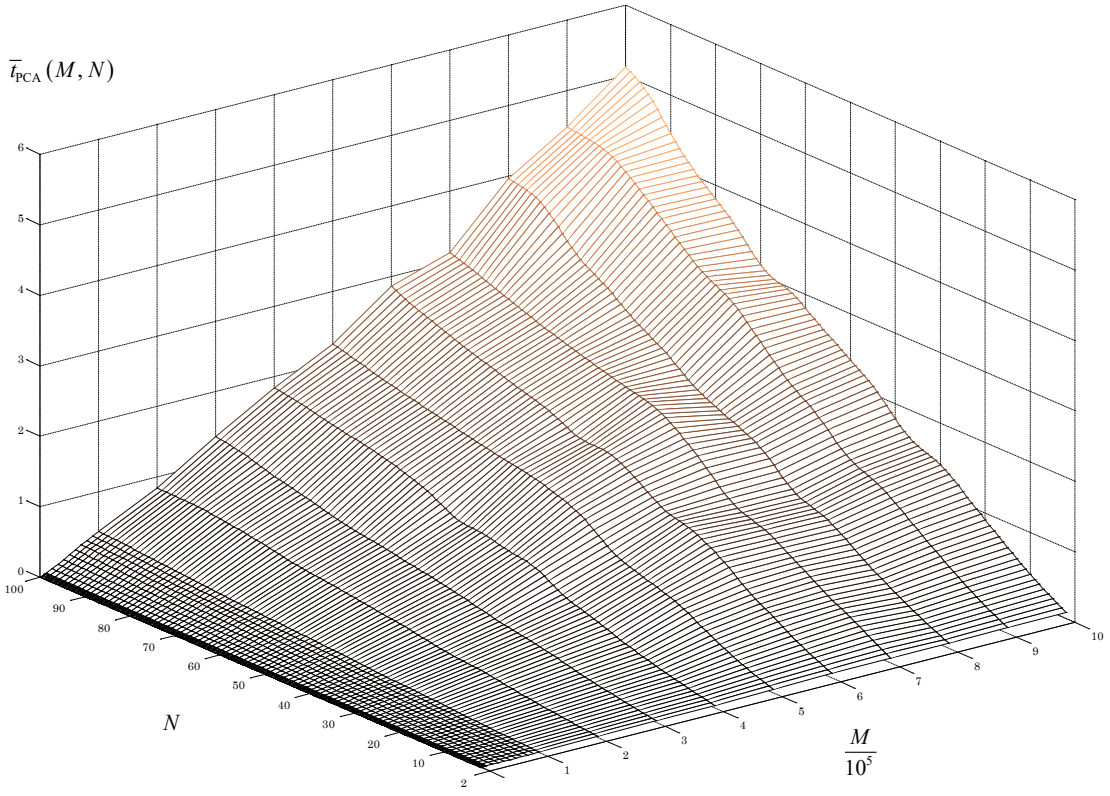

Fig. 4. The averaged computational time (3) of the PCA for single precision

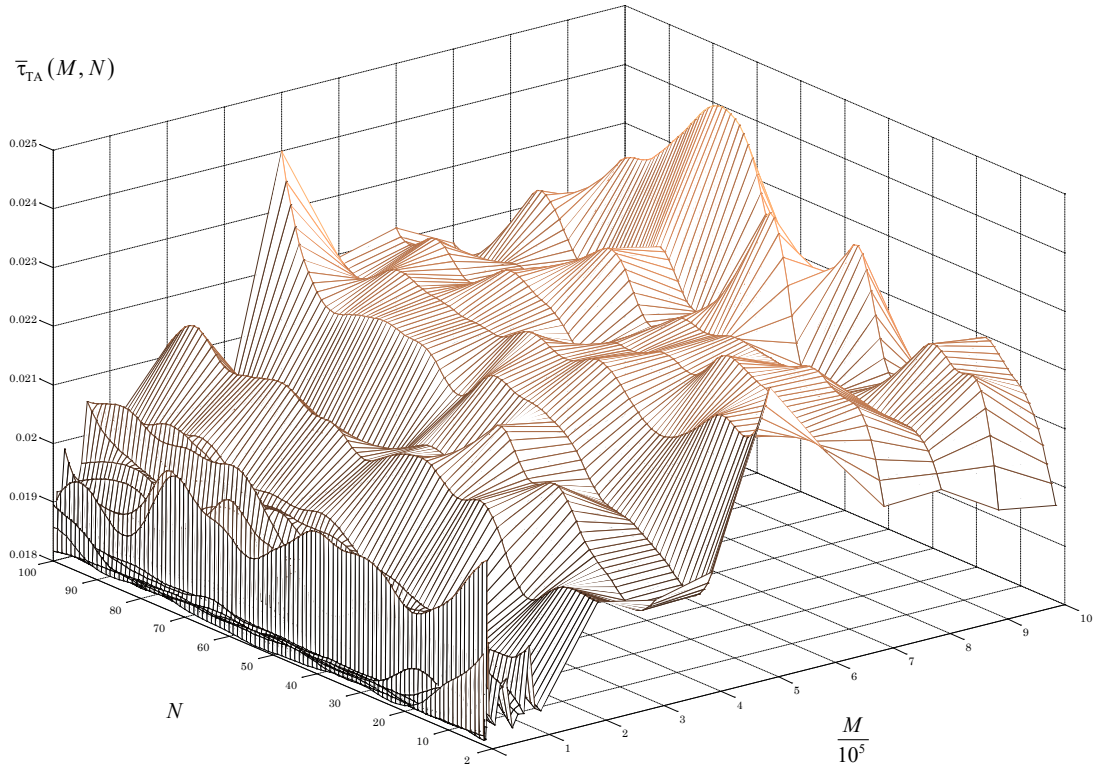

Fig. 5. The averaged time of the array-to-tall-array conversion for single precision

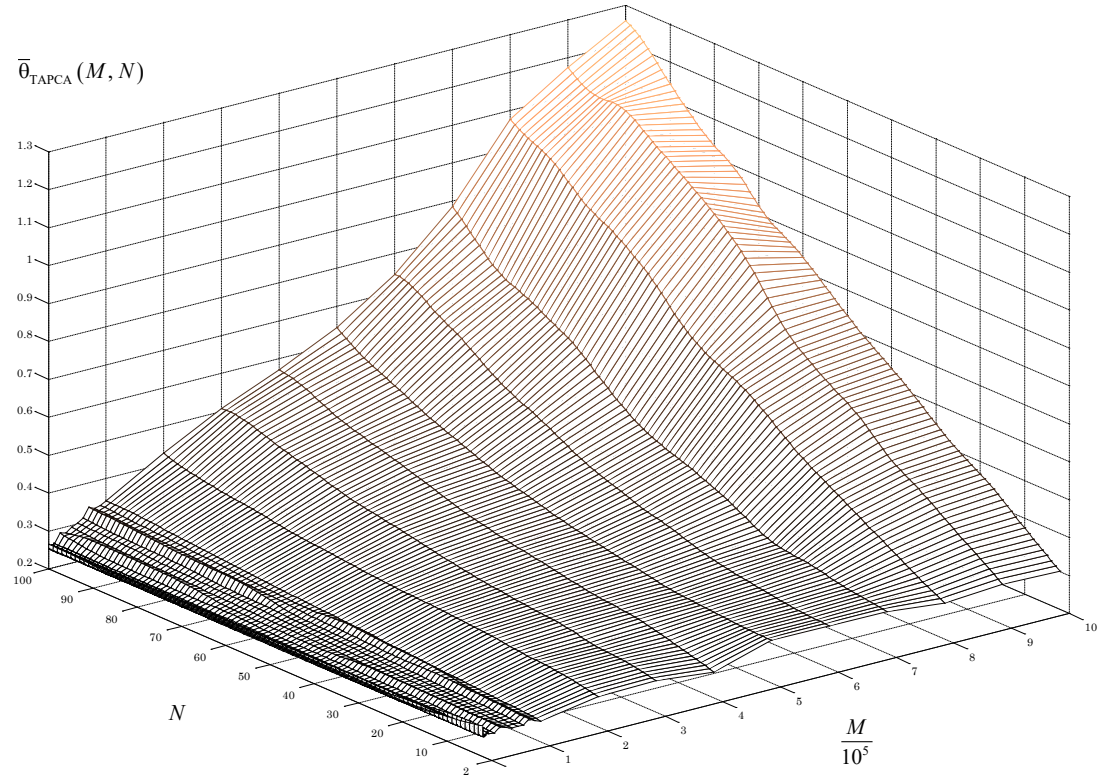

Fig. 6. The averaged computational time (5) of the TAPCA for single precision

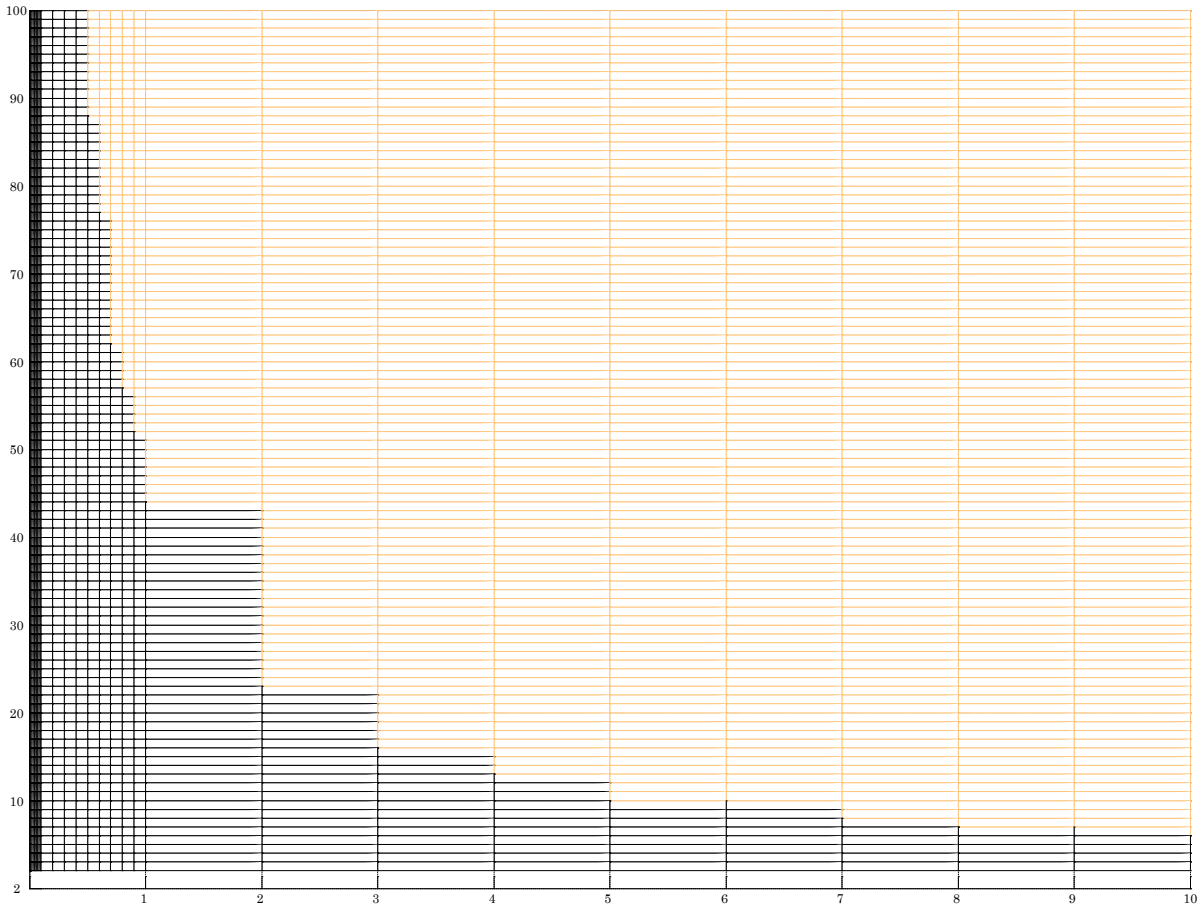

Fig. 7. Efficiency of the TAPCA for double precision on the Cartesian product of (1) and (2)

features is more efficiently handled by the ordinary PCA. The double precision TAPCA efficiency is approximately margined with hyperbola

$$N = \frac{4178585.7041}{M} + 1.661$$

$$\text{for } M \in \left[4 \cdot 10^4; 10^6\right]. \quad (9)$$

The single precision TAPCA efficiency is approximately margined with hyperbola

$$N = \frac{4919317.7503}{M} + 10.2581$$

$$\text{for } M \in \left[4 \cdot 10^4; 10^6\right]. \quad (10)$$

It is worth noting that the margin by (9) is about six times as more accurate than the margin by (10). It is also visible from Fig. 7 whose nearly-hyperbolic staircase margin is far less contorted than that in Fig. 8. An efficiency threshold can be deduced from (9) as the double-precision dataset size (the number of its entries, i. e. $M \cdot N$), at which applying the TAPCA is equivalent to applying the PCA, whereas principal components for larger datasets are better to compute by the Tall Array method. Thus, the double-precision dataset threshold is nearly 4.5 to 5.9 million entries. There is a similar finding for single-precision datasets whose threshold is nearly 5 to 15.2 million entries. The nearly-hyperbolic margins in Fig. 7 and 8 also allow concluding that the thresholds are likely to become lower for too “thin” datasets (having no more than 10 features or so) and datasets having 10^5 observations or so.

Discussion

The obtained results visualized in Fig. 1, 3, 4, 6–8 reveal what Tall Arrays are factually capable of and when they are efficient in computing principal components for dimensionality reduction of large datasets stored on disk. Computational time complexity of both the PCA and TAPCA is rather polynomial than strictly quadratic or cubic, although it

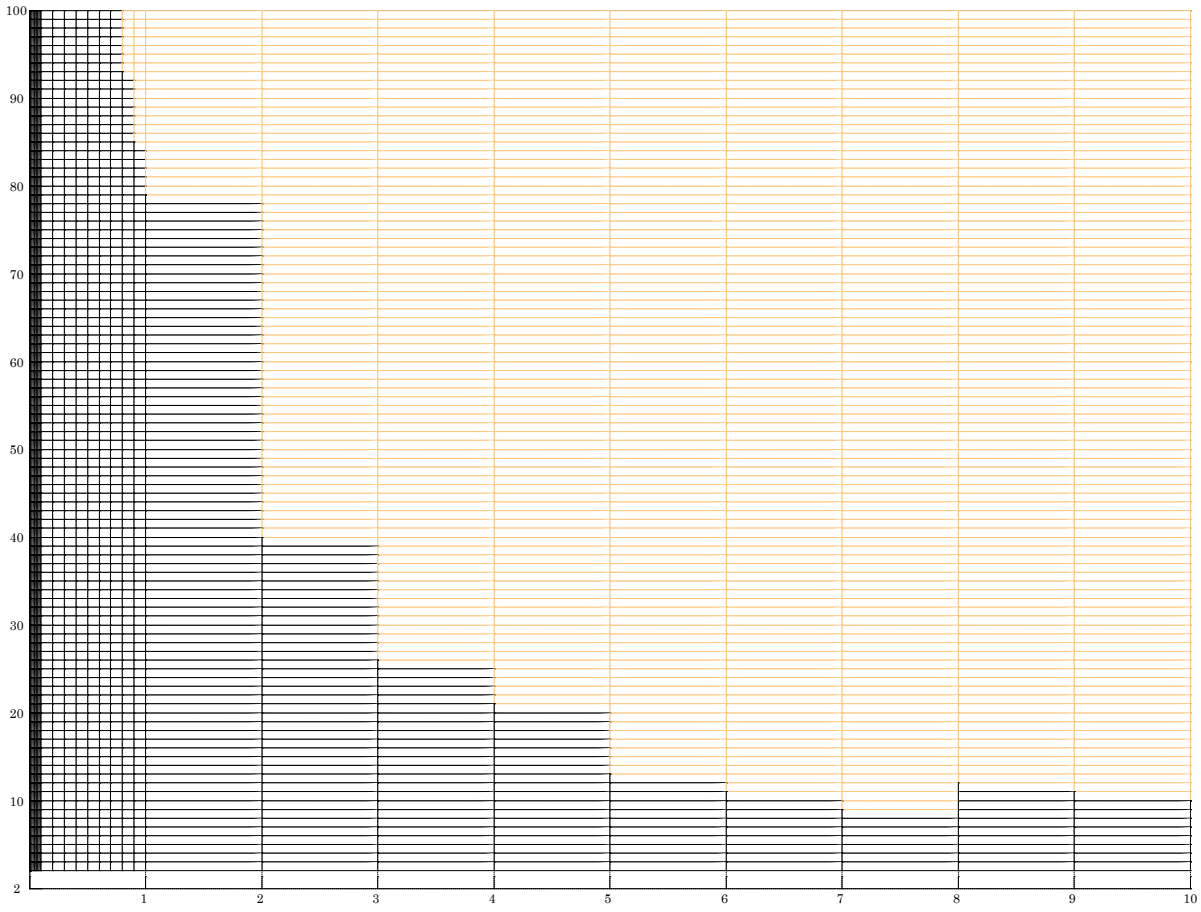

Fig. 8. Efficiency of the TAPCA for single precision on the Cartesian product of (1) and (2)

seems to be quasilinear. Nevertheless, computational time spans have been registered for the TAPCA by two parallel processor workers, so it is naturally expected that the TAPCA will be more efficient by more parallel processor workers. However, as the dataset size increases, whether in its number of observations or features, or both, the growth of ratio (7) reaches its saturation (see it in Fig. 9 for double precision) scarcely exceeding 5. Ratio (7) for the TAPCA factual efficiency for single precision looks similarly.

The nearly-hyperbolic margin, which alternatively could be called the TAPCA efficiency threshold, implies the size of a dataset (or the size of a Big Data instance) as matrix $\mathbf{X} = [x_{mn}]_{M \times N}$, by which the TAPCA and the ordinary PCA take approximately the same time to compute N principal components. For datasets whose size is above the threshold, the TAPCA is faster. Here, it is necessary to remember that starting parallel processor workers takes some time, so applying the TAPCA to smaller datasets, even if their size is above the threshold (but the size is close to the margin inside the hyperbola), is reasonable only for multiple times. Applying the TAPCA just once is reasonable if the dataset (say, of 10 features at most) cannot be loaded into the workspace. For instance, 10 principal components of a dataset as matrix $\mathbf{X} = [x_{mn}]_{550000 \times 10}$ of 550 000 observations and 10 features are computed by the PCA within 370.4 milliseconds, whereas the TAPCA takes about 375.5 milliseconds (i. e., it is 1.3789 % slower) if to count time spent on converting this matrix into a tall array. Without taking into account the array-to-tall-array conversion, the TAPCA takes about 353.1 milliseconds (which is 4.6733 % faster). So, the conversion taking here 22.4 milliseconds does not ruin the TAPCA efficiency (and thus it may be conditionally neglected) only if matrix $\mathbf{X}$ is converted into a tall array once and then the TAPCA is applied at least twice – to the tall array and its modification (such a modification is expected to be a subdataset of the initial dataset, rather than an expansion of the dataset).

A large dataset (a Big Data instance) is usually stored on disk by some privacy and security reasons.

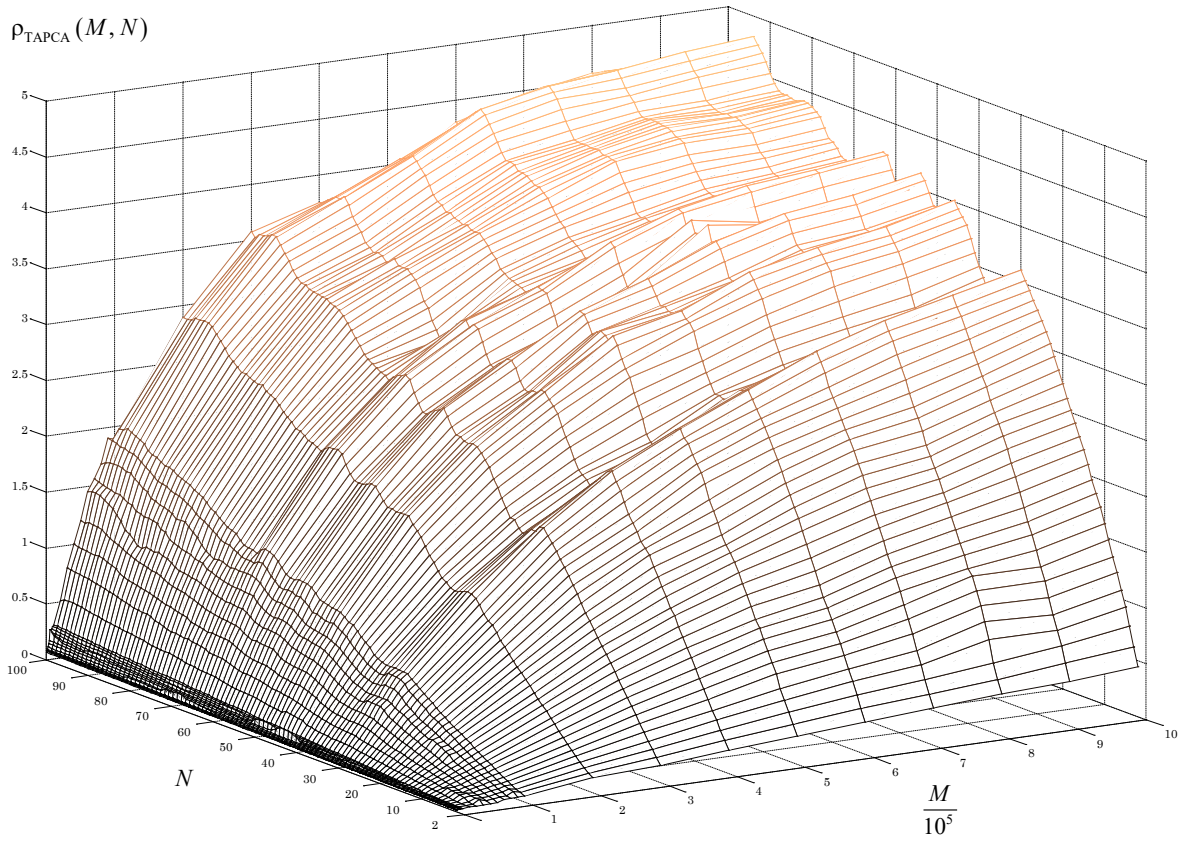

Fig. 9. Ratio (7) showing the TAPCA factual efficiency for double precision

This is when Tall Arrays become useful as they do not disclose factual data. In this comprehension, the array-to-tall-array conversion can be considered as a tradeoff (payment) for privacy and security along with simplified interoperability and manageability owing to dimensionality reduction by principal components.

Conclusions

In computing principal components for dimensionality reduction of large datasets stored on disk, the Tall Array method becomes efficient by two parallel processor workers if a dataset has at least 5 to 6 million entries. The Tall Array method is more efficient on datasets with double precision whose efficiency threshold is nearly 6 million entries, whereas the efficiency threshold for datasets with single precision is between 5 to 15.2 million entries. Both the thresholds may become lower as the number of dataset features is dropped below 10 or the number of observations does not exceed 10^5.

The presented research is nearly the worst-case scenario, in which only two parallel processor workers are deployed. As the number of workers increases, the TAPCA factual efficiency threshold is expected to drop further. The computational time complexity of both the PCA and TAPCA is polynomial, so the drop will be significant even for four parallel processor workers (it is also a commonly widespread computational architecture), let alone batch computation of principal components on computer clusters by using the Tall Array approach.

An open question is about the reasonability or efficiency of storing a large dataset on disk (prior to dimensionality reduction), when Big Data clusters like Hadoop or cloud-based solutions are available [21, 22]. The second open question, lying nearly in parallel to the mentioned one, is about efficiently using the MapReduce technique for dimensionality reduction by principal component analysis. Another open question is how to further speed up the PCA by implementing it (or TAPCA) on graphic processing units (GPUs). These questions are the perspective for further research. Furthermore, a general methodology of efficient computation of principal components for dimensionality reduction of large datasets should be formulated and justified.

References

[1] Ü. Demirbaga *et al.*, Big Data Analytics. Theory, Techniques, Platforms, and Applications, Springer, Cham, 2024. Retrieved from: doi: 10.1007/978-3-031-55639-5.

[2] A. Jamarani *et al.*, "Big data and predictive analytics: A systematic review of applications", *Artificial Intelligence Review*, 2024, Vol. 57, no. 176. Retrieved from: doi: 10.1007/s10462-024-10811-5

[3] I. Si-ahmed *et al.*, "Principal component analysis of multivariate spatial functional data", in *Big Data Research*, 2025, vol. 39, Art. no. 100504. Retrieved from: doi: 10.1016/j.bdr.2024.100504

[4] A. Meepaganithage *et al.*, "Enhanced Maritime Safety Through Deep Learning and Feature Selection", *Advances in Visual Computing. ISVC 2024. Lecture Notes in Computer Science*, Vol. 15047, Springer, Cham, 2025, pp. 309–321. Retrieved from: doi: 10.1007/978-3-031-77389-1_24

[5] W.J. Ewens and K. Brumberg, Introductory Statistics for Data Analysis, Springer, Cham, 2023. Retrieved from: doi: 10.1007/978-3-031-28189-1

[6] S. Akter and S.F. Wamba, Handbook of Big Data Research Methods, Edward Elgar Publishing, 2023. Retrieved from: doi: 10.4337/9781800888555

[7] V.V. Romanuke, "Fast Kemeny consensus by searching over standard matrices distanced to the averaged expert ranking by minimal difference", *Research Bulletin of NTUU "Kyiv Polytechnic Institute"*, 2016, no. 1, pp. 58–65. Retrieved from: doi: 10.20535/1810-0546.2016.1.59784

[8] J. Cao, "Data Collection in the Era of Big Data", *E-Commerce Big Data Mining and Analytics. Advanced Studies in E-Commerce*, Springer, Singapore, 2023, pp. 19–28. Retrieved from: doi: 10.1007/978-981-99-3588-8_2

[9] W.K. Ндrdle *et al.*, "Principal Component Analysis", *Applied Multivariate Statistical Analysis*, Springer, Cham, 2024, pp. 309–345. Retrieved from: doi: 10.1007/978-3-031-63833-6_11

[10] I.T. Jolliffe and J. Cadima, "Principal component analysis: a review and recent developments", *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 2016, Vol. 374, Iss. 2065, no. 20150202. Retrieved from: doi: 10.1098/rsta.2015.0202

[11] A.C. Olivieri, *Principal Component Analysis*, *Introduction to Multivariate Calibration*, Springer, Cham, 2024, pp. 71–87. Retrieved from: doi: 110.1007/978-3-031-64144-2_4

[12] V.V. Romanuke, "Speedup of the *k*-means algorithm for partitioning large datasets of flat points by a preliminary partition and selecting initial centroids", in *Applied Computer Systems*, 2023, Vol. 28, no. 1, pp. 1–12. Retrieved from: doi: 110.2478/acss-2023-0001

[13] S. Ekici *et al.*, "Electricity Consumption Analysis with Matlab Tall Arrays", 1*st International Engineering and Technology Symposium (1st IETS)*, May 2018, Batman University, Batman, Turkey, 2018. Retrieved from: https://www.researchgate.net/publication/327987720_Electricity_Consumption_Analysis_with_Matlab_Tall_Arrays

[14] M. Paluszek and S. Thomas, "Data for Machine Learning in MATLAB", *MATLAB Machine Learning Recipes*, Apress, Berkeley, CA, 2024, pp. 21–48. Retrieved from: doi: 10.1007/978-1-4842-9846-6_2

[15] V.V. Romanuke, "Limitation of effectiveness in using MATLAB gpuArray method for calculating products of transpose-symmetrically sized matrices", *Herald of Khmelnytskyi national university. Technical sciences*, 2015, no. 5, pp. 243–248. Retrieved from: https://elar.khmnu.edu.ua/handle/123456789/4611

[16] V.V. Romanuke, "Maximum-versus-mean absolute error in selecting criteria of time series forecasting quality", *Bionics of intelligence*, 2021, no. 1, pp. 3–9. Retrieved from: doi: 10.30837/bi.2021.1(96).01

[17] C.L. Valenzuela and A. J. Jones, "Evolutionary divide and conquer (I): A novel genetic approach to the TSP", *Evolutionary Computation*, 1993, Vol. 1, Iss. 4, pp. 313–333. Retrieved from: doi: 10.1162/evco.1993.1.4.313

[18] V.V. Romanuke, "Deep clustering of the traveling salesman problem to parallelize its solution", in *Computers & Operations Research*, 2024, Vol. 165, no. 106548. Retrieved from: doi: 10.1016/j.cor.2024.106548

[19] T. Weinzierl, Principles of Parallel Scientific Computing: A First Guide to Numerical Concepts and Programming Methods, Springer, Cham, 2021. Retrieved from: doi: 10.1007/978-3-030-76194-3

[20] V.V. Romanuke, "Optimal construction of the pattern matrix for probabilistic neural networks in technical diagnostics based on expert estimations", *Information, Computing and Intelligent Systems*, 2021, no. 2, pp. 19–25. Retrieved from: doi: 10.20535/2708-4930.2.2021.244186

[21] R. Han and Y. Wang, "The Advance and Performance Analysis of MapReduce", *Proceedings of 2nd International Conference on Artificial Intelligence, Robotics, and Communication. ICAIRC 2022. Lecture Notes in Electrical Engineering*, Vol. 1063, Springer, Singapore, 2023, pp. 205–213. Retrieved from: doi: 10.1007/978-981-99-4554-2_20

[22] S. Hedayati *et al.*, "MapReduce scheduling algorithms in Hadoop: a systematic study", *Journal of Cloud Computing*, 2023, Vol. 12, no. 143. Retrieved from: doi: 10.1186/s13677-023-00520-9.

В.В. Романюк

ЕФЕКТИВНІСТЬ МЕТОДУ TALL ARRAY У ЗНИЖЕННІ РОЗМІРНОСТІ НАБОРІВ ДАНИХ НА ОСНОВІ МЕТОДУ ГОЛОВНИХ КОМПОНЕНТІВ

Проблематика. Розвідковий аналіз даних швидко розвивається ще з початку 2000-х років. На 2025 рік більшість реальних наборів даних класифікують як великі дані. Робочий процес аналітики великих даних включає етап попередньої обробки даних, який є початковою точкою обробки великих даних. На цьому кроці дані намагаються максимально спростити. Основною парадигмою є зниження розмірності, що дозволяє спростити та візуалізувати масиви даних великої розмірності. Метод головних компонентів (PCA) є лінійним методом зниження розмірності. PCA можна прискорити, застосувавши Tall Arrays, якщо дані зберігаються на диску. Tall Array PCA (TAPCA) обчислює головні компоненти поступово, використовуючи стратегію divide-and-conquer.

Мета дослідження. Мета полягає у тому, щоб визначити, коли TAPCA фактично ефективний для зниження розмірності. Вивчаються два типи чисел – з подвійною та одинарною точністю.

Методика реалізації. Для досягнення зазначеної мети генеруються випадкові великі набори даних у вигляді матриць певного числового типу. Потім вимірюється час обчислень звичайного PCA у середовищі MATLAB, застосованого до згенерованих матриць. Далі вимірюється обчислювальний час перетворення масивів (згенерованих матриць) у пам'яті у tall-масиви. Також вимірюється час обчислення TAPCA, застосованого до тих згенерованих матриць, до яких PCA застосовувався раніше.

Результати дослідження. Порівняльний аналіз усередненого часу обчислень показує, що часова складність обчислень як PCA, так і TAPCA є радше поліноміальною, ніж строго квадратичною чи кубічною. Існує майже гіперболічна границя, яку альтернативно можна назвати порогом ефективності TAPCA, у площині кількості спостережень набору даних і кількості ознак набору даних, за якою TAPCA та звичайний PCA потребують приблизно однакового часу для обчислення головних компонентів.

Висновки. В обчисленні головних компонентів для зниження розмірності великих наборів даних, що зберігаються на диску, метод Tall Array стає ефективним за двох паралельних процесорів, якщо набір даних містить принаймні 5-6 мільйонів записів. Метод Tall Array більш ефективний для наборів даних з подвійною точністю, де його поріг ефективності становить майже 6 мільйонів записів, тоді як поріг ефективності для наборів даних з одинарною точністю становить від 5 до 15,2 мільйонів записів.

Ключові слова: зниження розмірності; метод головних компонентів (PCA); Tall Arrays; поріг ефективності; подвійна точність; одинарна точність.

Рекомендована Радою
факультету прикладної математики
КПІ ім. Ігоря Сікорського

Надійшла до редакції
30 січня 2025 року

Прийнята до публікації
30 червня 2025 року