Application of Truncated SVD on Storage of Positional Data

Nathanael Shane Bennet, 13524119
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13524119@mahasiswa.itb.ac.id, shane150806@gmail.com

***Abstract*—SVD, or Singular Value Decomposition is a technique to factorize an $m \times n$ matrix M into the form $U\Sigma V^T$ where U is an $m \times m$ orthogonal matrix, Σ is an $m \times n$ diagonal matrix containing the singular values of M, and V^T is an $n \times n$ orthogonal matrix. Truncated SVD only takes the largest k singular values creating the $m \times k$ matrix U_t, the $k \times k$ matrix Σ_t, and the $k \times n$ matrix V_t^T, which form a low rank approximation of M. Truncated SVD is used in image compression, noise reduction, text feature extraction, and recommendation systems. This paper examines the application of truncated SVD on sets of position and velocity vectors and comparing its accuracy to the application of truncated SVD on image compression. An implementation is provided which uses turtle graphics to compare the low rank approximation with the original matrix.**

***Keywords*—SVD, Linear Algebra, Truncated SVD, low rank approximation**

I. Introduction

Matrices frequently encode complex, high-dimensional data across diverse fields such as signal processing, computer vision, machine learning, and physics simulations. However, their full dimensionality often introduces computational burdens, noise amplification, and overfitting risks. Singular Value Decomposition (SVD), introduced by Eugène Beltrami and James Joseph Sylvester in the late 19th century and later formalized by Carl Eckart and Gale Young in 1936, provides a powerful solution by factorizing an $m \times n$ matrix M into $M = U\Sigma V^T$. Here, U is an $m \times m$ orthogonal matrix whose columns are left singular vectors, Σ is an $m \times n$ diagonal matrix containing non-negative singular values in descending order, and V^T is an $n \times n$ orthogonal matrix with right singular vectors as rows.

Truncated SVD refines this by selecting only the k largest singular values (where $k < \min(m, n)$), forming the low-rank approximation $M \approx U_t\Sigma_t V_t^T$ with U_t as $m \times k$, Σ_t as $k \times k$, and V_t^T as $k \times n$. This dimensionality reduction preserves the matrix's dominant structure while discarding noise-laden minor components, enabling efficient approximations.

SVD's versatility has made it foundational in numerous applications. In image compression, as popularized by NASA's use in the 1990s for space imagery, truncated SVD reduces storage by representing pixel intensity matrices with fewer singular values, achieving high fidelity at low bitrates—often outperforming JPEG for certain images. Noise reduction leverages SVD's separation of signal (large singular values) from noise (small ones), as seen in hyperspectral imaging where it denoises while retaining spectral features. In natural language processing, Latent Semantic Analysis (LSA) applies truncated SVD to term-document matrices for text feature extraction, mitigating sparsity and synonymy to improve search and clustering. Recommendation systems, like those in Netflix and Amazon, use matrix factorization via SVD to predict user preferences from incomplete rating matrices, filling gaps with low-rank assumptions.

Despite these successes, applications to spatiotemporal data like position and velocity vectors which are common in trajectory analysis, robotics, and physics simulations remain underexplored compared to static data like images. This paper examines truncated SVD on such vector sets, benchmarking its reconstruction accuracy against image compression standards using metrics like mean squared error and perceptual fidelity. We provide a Python implementation with Turtle graphics to visually compare low-rank approximations of motion paths against original trajectories and compressed images, demonstrating SVD's adaptability.

II. Theoretical Basis

A. Vectors

In the context of linear algebra, a vector is a physical quantity that cannot be expressed as a single number. Vectors have magnitude and direction and can be represented as an ordered list of numbers (often called components). Vectors can exist in various dimensions, with a vector in n-dimension space being written as (v_1, v_2, …, v_n). Vectors have several key properties, including:

1. Addition

$$u \pm v = \begin{bmatrix} u_1 \pm v_1 \\ u_2 \pm v_2 \\ \vdots \\ u_n \pm v_n \end{bmatrix}$$

2. Scalar Multiplication

$$cv = \begin{bmatrix} cv_1 \\ cv_2 \\ \vdots \\ cv_n \end{bmatrix}$$

3. Norm / Magnitude

$$\|v\| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$$

B. Matrices

A matrix (plural matrices) is a rectangular array of numbers or other mathematical objects arranged in rows and columns. An $m \times n$ matrix has m rows and n columns. Matrices can be used as representations of images, sets of data, or other objects. Key properties of matrices include:

1. Addition ($A_{mn} + B_{mn}$)

$$A + B = \begin{bmatrix} A_{11} + B_{11} & A_{12} + B_{12} & \cdots & A_{1n} + B_{1n} \\ A_{21} + B_{21} & A_{22} + B_{22} & \cdots & A_{2n} + B_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} + B_{m1} & A_{m2} + B_{m2} & \cdots & A_{mn} + B_{mn} \end{bmatrix}$$

2. Scalar Multiplication

$$cA = \begin{bmatrix} cA_{11} & cA_{21} & \cdots & cA_{n1} \\ cA_{21} & cA_{22} & \cdots & cA_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ cA_{m1} & cA_{m2} & \cdots & cA_{mn} \end{bmatrix}$$

3. Matrix Multiplication ($A_{mn} * B_{np} = C_{mp}$)

$$C_{ij} = \sum_{k=1}^{n} A_{ik} \times B_{kj}$$

4. Transpose

$$A^T = \begin{bmatrix} A_{11} & A_{21} & \cdots & A_{m1} \\ A_{12} & A_{22} & \cdots & A_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ A_{1n} & A_{2n} & \cdots & A_{mn} \end{bmatrix}$$

C. Eigenvalues and Eigenvectors

An eigenvector v of a matrix A is a nonzero vector that when multiplied by A is scaled by a constant factor λ (the eigenvalue) ($Av = \lambda v$). The eigenvalue can be calculated by the equation:

$$|\lambda I - A| = 0$$

And the eigenvector v can be calculated by the equation:

$$(\lambda I - A)v = 0$$

D. Singular Value Decomposition

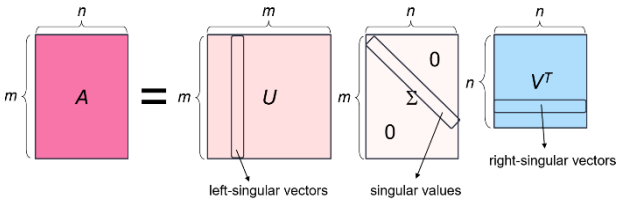


Figure 1. Illustration of SVD

Source: https://towardsdatascience.com/singular-value-decomposition-svd-demystified-57fc44b802a0/

Singular Value Decomposition (SVD) is a technique to factorize an $m \times n$ matrix M into the form $U\Sigma V^T$ where U is an $m \times m$ orthogonal matrix, Σ is an $m \times n$ diagonal matrix containing the singular values of M, and V^T is an $n \times n$ orthogonal matrix.

SVD is done using the following steps:

1. Calculate the eigenvalues and eigenvectors of A^TA.
2. Get the singular values of A by taking the square root of the nonzero eigenvalues from 1. Get Σ by placing the singular values from largest to smallest on an empty $m \times n$ matrix.
3. Calculate V^T by creating a matrix using the normalized eigenvectors from 1. As columns, then transpose the result.
4. Calculate the vectors u through the equation:

$$u_k = \frac{1}{\sigma_k} A v_k$$

The get U by using the normalized forms of u as columns.

SVD is used in various fields, including signal processing, recommendation systems, and text analysis.

E. Truncated SVD

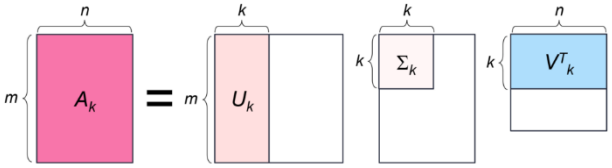


Figure 2. Illustration of truncated SVD

Source: https://towardsdatascience.com/singular-value-decomposition-svd-demystified-57fc44b802a0/

Truncated SVD is a variation of SVD done by only taking the largest k singular values, creating the $m \times k$ matrix U_t, the $k \times k$ matrix Σ_t, and the $k \times n$ matrix V_t^T, which form a low rank approximation of M. Truncated SVD is used mainly in image compression.

III. Implementation

A. Programming Language

For this project, we will use Python as our programming language. We chose Python for its NumPy library, which greatly simplifies matrix calculations, including SVD.

```
import numpy as np
import random
import turtle
```

Figure 3. Libraries used
Source: Own work

For this project we used the libraries NumPy, random, and turtle. As previously said, NumPy is used for its ability to perform matrix calculations out of the box, random is used to generate the data to be used in SVD, and turtle is used to graphically display the data and the reconstruction from truncated SVD.

B. Data Generation

The generated data will be in the form of individual objects, with object k having attributes (x_k, y_k, $v_{x,k}$, $v_{y,k}$). 4 objects form a block, with one object per column. The total dataset will be a square matrix composed from n^2 blocks.

```
size = 5
realsize = 4 * size
matrix = np.zeros((realsize, realsize))
for i in range(realsize):
    for j in range(realsize):
        if i % 4 == 0 or i % 4 == 1:
            matrix[i, j] = random.uniform(-300,
300)
        else:
            matrix[i, j] = random.uniform(-40,40)
```

Figure 4. Data Generation section
Source: Own work

C. Truncated SVD

Truncated SVD in NumPy can be done by simply performing svd() on a matrix then truncating the results to k. The k value is selected by choosing the largest k that results in a decomposition which has less total elements than the original matrix. This is done by the equation:

$$n * k + k * k + k * n < n^2$$
$$k^2 + 2nk < n^2$$

Which when plotted with x = n, y = k, and x, y > 0, draws a line approximated by y = 0.4142x.

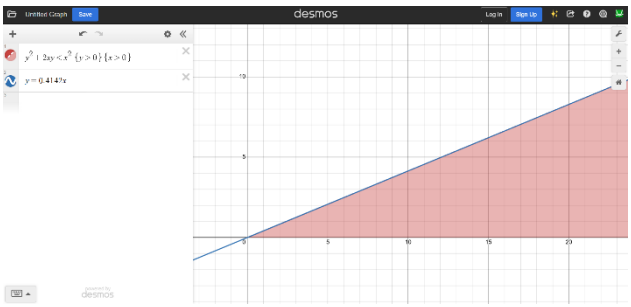


Figure 5. Plot of $k^2 + 2nk < n^2$ on Desmos
Source: Own work

```
U, s, Vt = np.linalg.svd(matrix)
Ur = U[:, :k]
Sr = np.diag(s[:k])
Vtr = Vt[:k, :]
approx = Ur @ Sr @ Vtr
```

Figure 6. Truncated SVD section
Source: Own work

D. Information Display

Various pieces of information will be displayed to help determine the accuracy of the truncation, including the total elements of both the original matrix and the decomposition, k, the singular values, the proportion of singular values captured by k, and the difference between the reconstruction and the original matrix.

```
print(f"original matrix: {matrix.size} elements")
print(f"truncated SVD matrix: {Ur.size + Sr.size
+ Vtr.size} elements")
print(f"smooth = {normalize}")
print(f"k = {k}")
print(s)
print(f"energy = {np.sum(s[:k]) / np.sum(s)}")
print(matrix)
print(approx)
print(np.linalg.norm(matrix - approx))
```

Figure 7. Information Display section
Source: Own work

E. Turtle Display

The program also uses turtle graphics to display the objects from both the original (in black) and the reconstructed matrix (in red), with a blue line connecting them. The position of an object is represented by a dot, and the velocity is represented by a line extending away from the dot. The program uses speed(0) and ht() to speed up the program.

```
pi = 3.141592653589793

def display_object(t, x, y, velx, vely):
    t.seth(0)
    t.teleport(x, y)
    t.dot()
    t.seth(np.arctan(vely / velx) * 180 / pi)
    t.forward(np.sqrt(vely * vely + velx * velx))

def display_line(t, x1, y1, x2, y2):
    t.teleport(x1, y1)
    t.goto(x2, y2)

t = turtle.Turtle()

t.speed(0)
t.ht()
for i in range(4 * size * size):
    t.color("blue")
    display_line(t, matrix[i // realsize * 4, i %
realsize],
    matrix[i // realsize * 4 + 1, i % realsize],
    approx[i // realsize * 4, i % realsize],
    approx[i // realsize * 4 + 1, i % realsize],)
    t.color("black")
    display_object(t, matrix[i // realsize * 4, i
% realsize],
    matrix[i // realsize * 4 + 1, i % realsize],
    matrix[i // realsize * 4 + 2, i % realsize],
    matrix[i // realsize * 4 + 3, i % realsize])
    t.color("red")
    display_object(t, approx[i // realsize * 4, i
% realsize],
    approx[i // realsize * 4 + 1, i % realsize],
    approx[i // realsize * 4 + 2, i % realsize],
    approx[i // realsize * 4 + 3, i % realsize])

turtle.done()
```

Figure 8. Turtle Display section
Source: Own work

IV. RESULTS

The results of the program with size = 3 are as follows:

```
original matrix: 144 elements
truncated SVD matrix: 112 elements
k = 4
[860.49 792.68 684.10 545.87 365.52 275.08 107.87  62.70  58.70  49.15  16.41   3.77]
energy = 0.7542891439976321
[[ 144.37   97.16  298.86 -291.03  160.67   69.22   63.69   -6.69 -220.63 -127.83  179.77 -260.24]
 [ 162.60  -74.34  -66.34  274.20  212.38 -127.35 -115.81  -33.46  -57.95  109.12  228.24  -27.49]
 [  33.91   25.49  -25.79   39.57  -18.15  -33.55   20.08  -23.57  -25.15  -18.34  -20.46   20.83]
 [  23.54    9.19   38.42    4.07  -27.30  -30.78    4.72  -19.50  -16.94   24.77   37.42   18.66]
 [  67.24  123.40  117.76  290.77  141.46   72.79 -264.86 -186.00  272.44  233.02  288.90  -17.04]
 [  99.99 -137.95 -223.96  169.72 -199.73   66.24  -76.53  265.74 -144.44  297.77 -270.63  224.02]
 [ -30.26  -19.25   20.04    4.81    7.89  -27.94  -10.39   24.18   -5.78    0.37   32.20   -1.55]
 [ -29.91  -20.90   28.53   16.25  -30.89   34.09   38.34    4.37   -3.51   -0.01   -1.64  -30.77]
 [-276.23  -74.15    7.09 -249.14  285.23 -215.64  -74.86  -72.93  164.21  -25.81 -196.64  168.37]
 [  93.85  244.74  -86.14   97.02  297.69 -196.67 -111.33   78.81   87.21 -234.21 -190.25 -185.82]
 [  35.61  -30.84   35.68  -35.34   27.75  -12.86   13.56  -12.71  -32.09   10.20  -21.08   23.22]
 [  32.11   -8.20   10.54  -24.71  -34.55  -23.74   15.29  -32.24   30.52  -25.38   30.66   -6.33]]
[[ 107.69  122.99  245.11 -224.30   70.56   62.93  129.08  -53.85 -153.51 -229.19  200.74 -286.60]
 [ 149.78   80.12  -30.95  285.00  108.61  -19.35 -145.39  -28.75   66.37   85.68  145.02  -66.39]
 [  20.76    6.27  -21.91   24.33   -0.08   -6.47   -4.74   21.83  -12.49   -1.81  -18.79   -5.25]
 [   7.63   -3.21   16.50    6.22  -12.80   18.56    0.91  -12.24   -2.33   14.81   33.96   -4.02]
 [  57.14   53.19   74.58  313.23  156.85   10.68 -221.69 -207.73  234.42  201.42  344.50   -7.22]
 [  64.82 -133.35 -282.14  232.30 -269.46   46.93   -9.77  218.79 -100.61  204.33 -238.40  205.33]
 [  -6.23    0.19    7.05   -2.99    5.84   -0.44   -2.91   -9.81    8.87    1.59    8.90    0.96]
 [   0.25  -10.31   13.36  -16.81  -27.80   20.93   16.78   -0.87  -16.68    5.40   13.48    0.46]
 [-303.56  -20.51  -18.93 -205.87  204.74 -190.08  -43.16  -98.84  237.14  -94.72 -203.38  141.26]
 [  99.78  193.68  -99.17   90.49  336.45 -231.76 -103.51   76.10   41.00 -223.82 -162.05 -169.48]
 [  -6.22   -1.42    7.85  -23.76   -4.52    2.33   12.56    1.06   -8.58  -12.13   -4.82   -3.74]
 [  -3.22    1.25   21.88  -16.94    0.14    8.96    6.69  -14.08   -0.53   -4.32   22.89   -9.15]]
480.60942089535513
```

Figure 9. Results of size = 3
Source: Own work

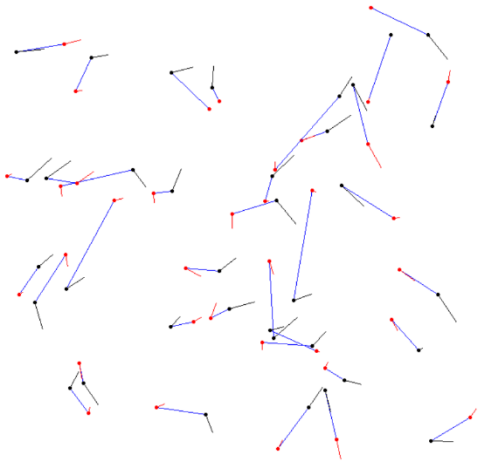


Figure 10. Turtle results of size = 3

The lack of accuracy in the results can be explained by looking at the singular values, where it can be seen that k = 4 only captures 75% of the total singular values. While this is mostly sufficient for images, this reconstruction is not good enough for coordinate data like the one generated. To test this theory, we created a function to create an image from both the original and reconstructed matrix.

```
def display_dot(t, x, y, color):
    color = min(color, 1)
    color = max(color, 0)
    t.teleport(x, y)
    t.color(color, color, color)
    t.dot()

for index, value in np.ndenumerate(matrix):
    row, col = index
    if row % 4 <= 1:
        display_dot(t, col * 4, row * 4, (value +
300) / 600)
    else:
        display_dot(t, col * 4, row * 4, (value +
40) / 80)

for index, value in np.ndenumerate(approx):
    row, col = index
    if row % 4 <= 1:
        display_dot(t, col * 4 - realsize * 4 -
40, row * 4, (value + 300) / 600)
    else:
        display_dot(t, col * 4 - realsize * 4 -
40, row * 4, (value + 40) / 80)
```

Figure 11. Image Display section
Source: Own work

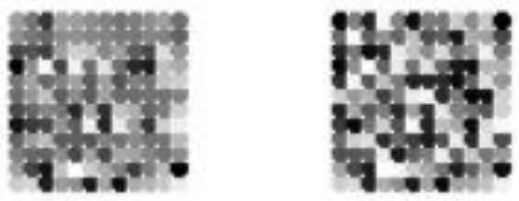


Figure 12. Image Display results for size = 3 and k = 4
Source: Own work

As can be seen from Fig. 12, the reconstructed image is still readable, while the reconstructed coordinate data is not very accurate. It can also be seen that the rows containing velocity data (rows where row % 4 >= 2) are significantly less accurate than rows containing positional data. This is due to the larger absolute values of the positional data saturating the singular values, meaning that the largest singular values mostly contain positional data, while the velocity data is contained within the smaller singular values. This can be seen in the case where size = 3 and k = 6, where while the positional data is correct, the velocity data still contains inaccuracies.

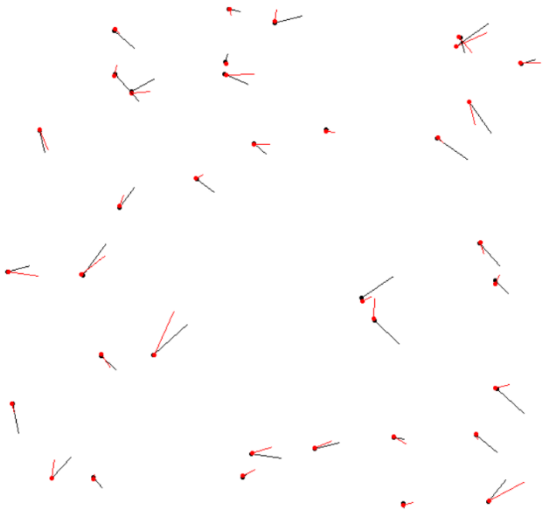


Figure 13. Turtle Results for k = 6
Source: Own work

We can solve this by normalizing (subtracting by the mean and dividing by the standard deviation) the data by row before SVD and denormalizing the reconstruction. This prevents the larger positional data from saturating the smaller velocity data, though at the cost of further accuracy.

```
normalize = True

rowmeans = np.zeros(realsize)
rowstd = np.zeros(realsize)

for i in range(realsize):
    rowmeans[i] = np.mean(matrix[i])
    rowstd[i] = np.std(matrix[i])

if normalize == True:
    matrix_norm = (matrix - rowmeans[:, None]) /
(rowstd[:, None] + 1e-8)
    U, s, Vt = np.linalg.svd(matrix_norm)
else:
    U, s, Vt = np.linalg.svd(matrix)

if normalize == True:
    approx = approx * (rowstd[:, None] + 1e-8) +
rowmeans[:, None]
```

Figure 14. Modifications to code for normalization
Source: Own work

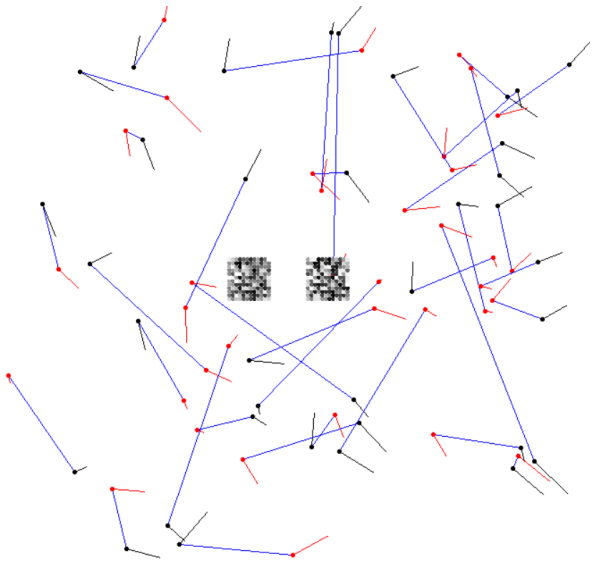


Figure 15. Turtle and Image Display results with k = 4 and normalization
Source: Own work

This image also lets us examine why the reconstructed image result looks much better than the reconstructed coordinate image. Looking at the top-right corner of the images, two of the pixels are somewhat darker in the reconstruction than the original. While in the image this would be interpreted as slight blurring or artifacting, in the coordinate data this would be interpreted as a significant difference in the y position coordinate and the x velocity coordinate of an object. This illustrates that a low rank approximation for coordinate data requires more accuracy than what the unmodified truncated SVD algorithm currently used can provide.

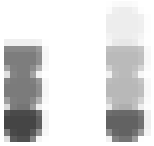


Figure 16. Top right corners of Image Display results
Source: Own work

V. Conclusion

This study explored the application of truncated SVD to sets of position and velocity vectors, comparing reconstruction quality against image-compression scenarios and highlighting several domain-specific insights. Overall, truncated SVD can effectively capture the dominant structure in spatial data, but its efficacy on coordinate data (positions and velocities) is more sensitive to scale, heterogeneity across data components, and the relative magnitudes of the features. Further avenues for research include exploration of tensor SVD or other low-rank approximations, randomized SVD for larger datasets, and better metrics through plots of singular vectors and reconstruction errors by feature type.

VI. Appendix

Github: https://github.com/kalkabena/turtleTSVDtest.git

Video: https://drive.google.com/file/d/1c-gzN_ehE2bdhGuxamsCizyTOQ1LNFop/view?usp=sharing

VII. Acknowledgment

The author expresses heartfelt gratitude to The Lord Almighty for the strength, resolve, and chance to finish this paper. They also sincerely thank Arrival Dwi Sentosa, S.Kom., M.T., the lecturer of IF2123 Linear Algebra and Geometry, for his devoted guidance and support, which have inspired them throughout their teaching experience with the students.

References

[1] Munir, Rinaldi, "Ruang vektor umum (Bagian 1)", 2025, [Online], Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/Algeo-15-Ruang-vektor-umum-Bagian1-2025.pdf. [Accessed: Dec. 25, 2025]

[2] Munir, Rinaldi, "Review matriks", 2025, [Online], Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/Algeo-01-Review-Matriks-2025.pdf. [Accessed: Dec. 25, 2025]

[3] Munir, Rinaldi, "Nilai Eigen dan Vektor Eigen (Bagian 1)", 2025, [Online], Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/Algeo-19-Nilai-Eigen-dan-Vektor-Eigen-Bagian1-2025.pdf. [Accessed: Dec. 25, 2025]

[4] Munir, Rinaldi, "Singular Value Decomposition (Bagian 1)", 2025, [Online], Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/Algeo-21-Singular-value-decomposition-Bagian1-2025.pdf. [Accessed: Dec. 25, 2025]

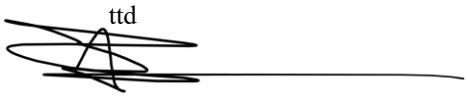
[5] Munir, Rinaldi, "Singular Value Decomposition (Bagian 2)", 2025, [Online], Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/Algeo-22-Singular-value-decomposition-Bagian2-2025.pdf [Accessed: Dec. 25, 2025]

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 24 Desember 2025

ttd



Nama dan NIM