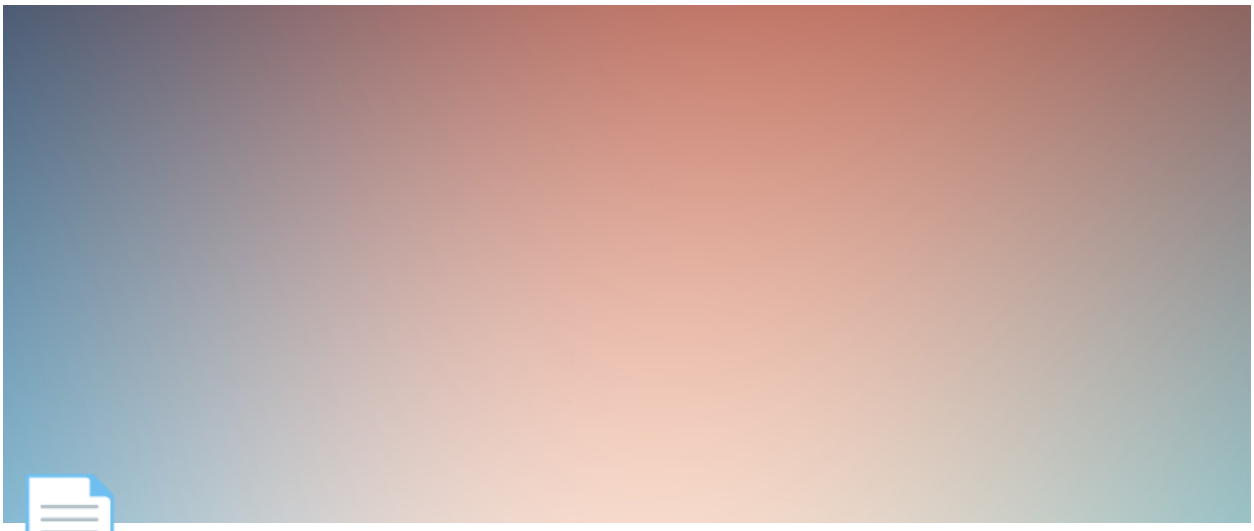


Exploring Matrix Factorization: LU, SVD, and QR Decomposition in Python and MATLAB

Supervisor:

Dr. Tsui-Wei Weng

Posted:

Dec 1, 2022

Course:

DSC 210 FA'23 Numerical Linear Algebra

Team:

- Mansi
  PID: A59020070
  HDSI
- Shobhit
  PID: A59020071
  HDSI
- Jahnavi
  PID: A69027892
  HDSI

1. Introduction

Matrix factorization is a cornerstone in linear algebra, providing essential methods for solving linear systems and understanding matrix structures. This project explores three key factorization techniques: LU, SVD, and QR decomposition, using Python and MATLAB, highlighting their applications and comparing their implementations.

1.1 History/Background:

LU decomposition, pioneered by Alan Turing in the 1940s, became a cornerstone in numerical linear algebra [1]. The factorization breaks down a matrix into the product of a lower triangular matrix (L) and an upper triangular matrix (U). This method found early applications in solving linear systems, and its significance was further emphasized in the influential work "Matrix Computations" by Gene H. Golub and Charles F. Van Loan [2].

SVD, introduced in the 1960s, expresses a matrix as the product of three matrices: U, Σ, and V^T. Golub and Reinsch's 1970 paper, "Singular Value Decomposition and Least Squares Solutions," played a crucial role in formalizing and popularizing SVD [3]. This method is widely used in signal processing, data compression, and dimensionality reduction.

QR decomposition, with roots dating back to the 1920s, represents a matrix as the product of an orthogonal matrix (Q) and an upper triangular matrix (R). Lloyd N. Trefethen and David Bau's "Numerical Linear Algebra" provides valuable insights into the theoretical foundations and practical applications of QR decomposition [4].

1.2 Applications:

Matrix factorization techniques, including LU, SVD, and QR decomposition, have widespread applications across various domains, contributing significantly to solving diverse real-world problems.

LU Decomposition:

Image Processing

LU decomposition is used in image processing for tasks such as image compression and denoising. By decomposing an image matrix into lower and upper triangular matrices, it becomes easier to manipulate and analyze the image data. This

decomposition is employed in various algorithms for efficient storage and processing of images. [5]

Singular Value Decomposition (SVD):

Collaborative Filtering in Recommender Systems

SVD is widely used in collaborative filtering algorithms for recommender systems. In this context, it helps in factorizing the user-item interaction matrix to identify latent features and provide personalized recommendations. SVD is applied to discover patterns and relationships in large datasets, making it a powerful tool for recommendation engines. [6]

QR Decomposition:

Linear Regression in Statistics

QR decomposition is frequently employed in linear regression to solve the least squares problem. It decomposes the design matrix into an orthogonal matrix (Q) and an upper triangular matrix (R). This decomposition aids in solving the linear regression equations efficiently, especially when dealing with over-determined systems of equations. [7]

1.3 State-of-the-art:

In this report, we study three approaches for Matrix Factorizations:

- Randomized Algorithms
- Tensor Decompositions
- Machine Learning Techniques

The focus of this report is the use of (LU, SVD, and QR decomposition as the first approach in matrix factorization. However, these decompositions are only useful when dealing with traditional and linear pattern data. Thus, we next study modern approaches such as Randomized Algorithms, Tensor Decompositions, and Machine Learning Techniques which have remained one of the most popular algorithms used for matrix factorization.

---

2. Problem Formulation

2.1 Relation to numerical linear algebra:

In the context of matrix factorization techniques, the fundamental question is: "How do we represent matrices mathematically, and what insights can we gain from these representations?" We explore this question by focusing on LU, SVD, and QR decomposition and their applications, drawing parallels with numerical linear algebra principles.

LU Decomposition:

When representing matrices for LU decomposition, we encounter challenges such as division by zero can occur if the pivot element in the Gaussian elimination step is zero.

Singular Value Decomposition (SVD):

When representing matrices for LU decomposition, we had to make sure that the matrix is of full rank for an exact decomposition. Rank-deficient matrices may cause issues.

QR Decomposition:

For QR decomposition, we make sure to use a non-singular matrix. As QR decomposition can fail for singular matrices.

2.2 Approach description:

Our project, "Exploring Matrix Factorization: LU, SVD, and QR Decomposition in Python and MATLAB," follows a systematic approach to understanding and implementing the matrix factorization techniques, emphasizing their applications and comparative analysis.

LU Decomposition:

1. **Matrix Representation:**
   Create a matrix A, which must be a square matrix. Initialize L and U matrices. L will be a lower triangular matrix, and U will be an upper triangular matrix.
2. **Decomposition Process:**
   Apply LU decomposition to factorize the A into lower (L) and upper (U) triangular matrices.

Singular Value Decomposition (SVD):

1. **Matrix Representation:**
   Create a matrix A which must be a square matrix. Initialize U, S, and V^T matrices. U will be a left singular vectors matrix., S will be the singular values in a 1-D array and V^T will be the transpose of the right singular vectors matrix.
2. **SVD Decomposition:**
   Apply SVD decomposition to factorize the A to find the final singular matrix.

QR Decomposition:

1. **Matrix Representation:**
   Create a matrix A, which must be a square matrix. Initialize Q and R matrices. Q an orthogonal matrix, and R will be an upper triangular matrix.
2. **QR Decomposition:**
   Apply QR decomposition to factorize the A into Q and R matrices. .

2.3 SOTA Approach description:

In this project, we have considered three modern approaches to solve problems that traditional NLA fails to solve.

1. **Randomized Algorithms:**
   Randomized algorithms use random numbers to make decisions during computation. They are often employed to approximate solutions to mathematical problems with significant computational savings. In the context of linear algebra, randomized algorithms can be used for tasks like matrix approximation and factorization. They provide a trade-off between computational efficiency and accuracy. One example is the use of random sampling techniques in matrix sketching or randomized SVD (Singular Value Decomposition). [9]
2. **Tensor Decomposition:**
   Tensor decomposition, also known as tensor factorization, is a technique used to break down a higher-order tensor into a sum of simpler tensors. It is an extension of matrix factorization to higher dimensions. Tensor decomposition is widely used in various fields, including machine learning, signal processing, and neuroscience. Common methods include Canonical Polyadic (CP) decomposition, Tucker decomposition, and PARAFAC (parallel factor analysis). [10]

3. **Machine Learning Techniques:**
   Matrix factorization is a popular technique in machine learning for collaborative filtering, dimensionality reduction, and recommendation systems. It involves factorizing a matrix into the product of two or more lower-rank matrices. Singular Value Decomposition (SVD), Alternating Least Squares (ALS), and Stochastic Gradient Descent (SGD) are common algorithms for matrix factorization. [11]

2.4 Evaluation:

For the evaluation of our project, we plot each matrix factorization approach in both Python and MATLAB. Along with that, we have timed each approach decomposition process of matrix so that we can know how much time LU, SVD, and QR decomposition takes in both Python and MATLAB

---

3. Experiments

3.1 Setup and logistics:

- We use Python and MatLab as the programming language for this project. We will use Jupyter Notebook to run Python code and use Matlab for coding Matlab program and all files are saved on GitHub for reference.
- For Python, we're using NumPy for matrix operations, SciPy for advanced linear algebra functions, and Matplotlib for visualizations. In MATLAB, we'll utilize its built-in functions for all three decompositions. These tools were chosen for their widespread use and robustness in mathematical computations.
- Finally, we use Notion to write the project report.

3.2 Dataset and preprocessing:

We have created a 50x50 matrix using Numpy Library.

Also, we have created a 200x200, matrix using Numpy Library.

3.3 Implementation:

LU Decomposition:

For Python, we will use lu from the scipy-linalg library to perform LU decomposition on a square matrix.

```
# Start timer
start_time = time.time()

# Perform LU Decomposition
P, L, U = lu(A)

# End timer
end_time = time.time()

# Calculate execution time
execution_time = end_time - start_time
print(f"LU Decomposition in Python took: {execution_time} secon
```

For Matlab, we will use lu which is a built-in method for calculating LU decomposition of a matrix.

```
% Start timer
tic;

% Perform LU Decomposition
[L, U, P] = lu(A);

% End timer and print execution time
toc;
```

Singular Value Decomposition (SVD):

For Python, we will use svd from the scipy-linalg library to perform SVD on a square matrix.

```
# Start timer
start_time = time.time()
```

```
# Perform SVD
U, S, VT = np.linalg.svd(A)

# End timer
end_time = time.time()

# Calculate execution time
execution_time = end_time - start_time
print(f"SVD in Python took: {execution_time} seconds")
```

For Matlab, we will use svd which is a built-in method for calculating SVD of a matrix.

```
% Start timer
tic;

% Perform SVD
[U, S, VT] = svd(A);

% End timer and print execution time
elapsedTime = toc;
fprintf('SVD in MATLAB took: %f seconds\n', elapsedTime);
```

QR Decomposition:

For Python, we will use qr from the scipy-linalg library to perform QR decomposition on a square matrix.

```
# Start timer
start_time = time.time()

# Perform QR Decomposition
Q, R = np.linalg.qr(A)

# End timer
end_time = time.time()
```

```
# Calculate execution time
execution_time = end_time - start_time
print(f"QR Decomposition in Python took: {execution_time} secon
```

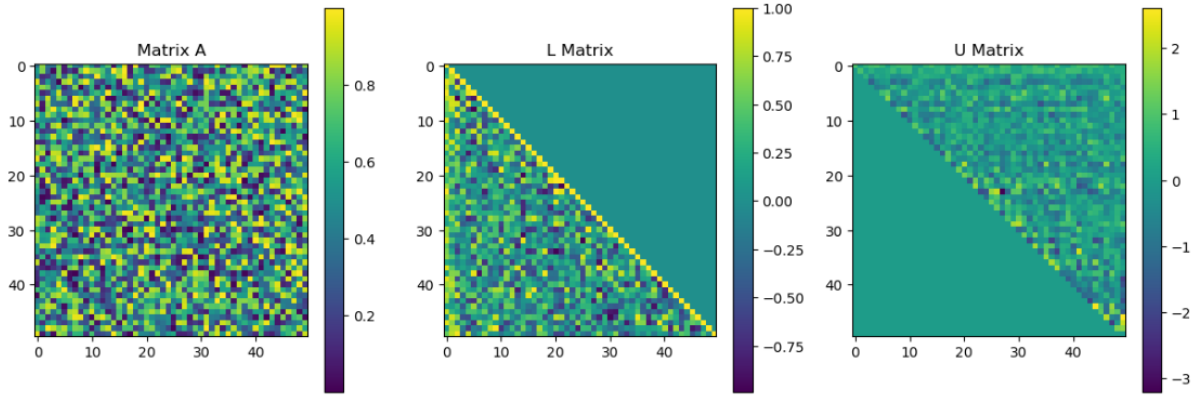
For Matlab, we will use qr which is a built-in method for calculating QR decomposition of a matrix.

```
% Start timer
tic;

% Perform QR Decomposition
[Q, R] = qr(A);

% End timer and print execution time
elapsedTime = toc;
fprintf('QR Decomposition in MATLAB took: %f seconds\n', elapse
```

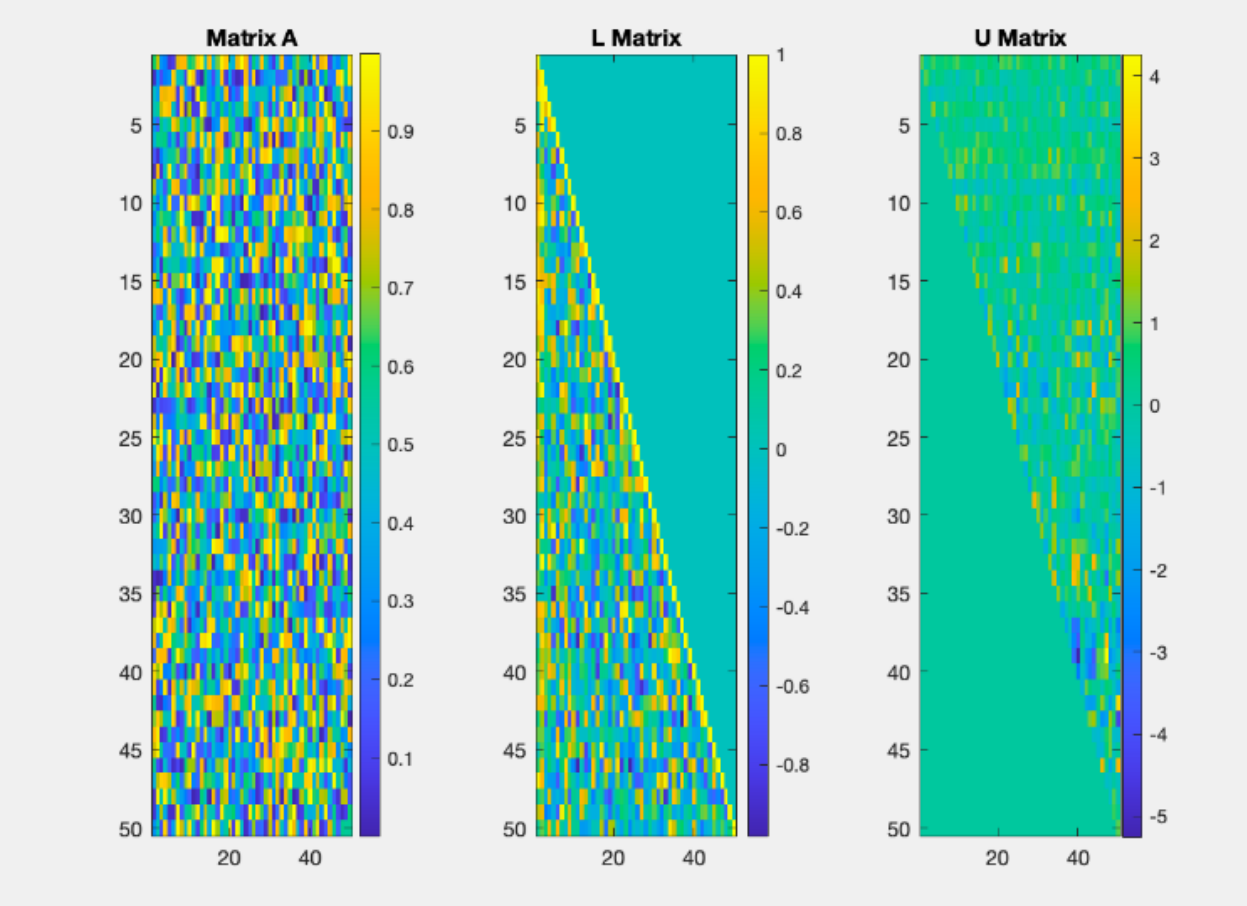
3.4 Results:

LU Decomposition:

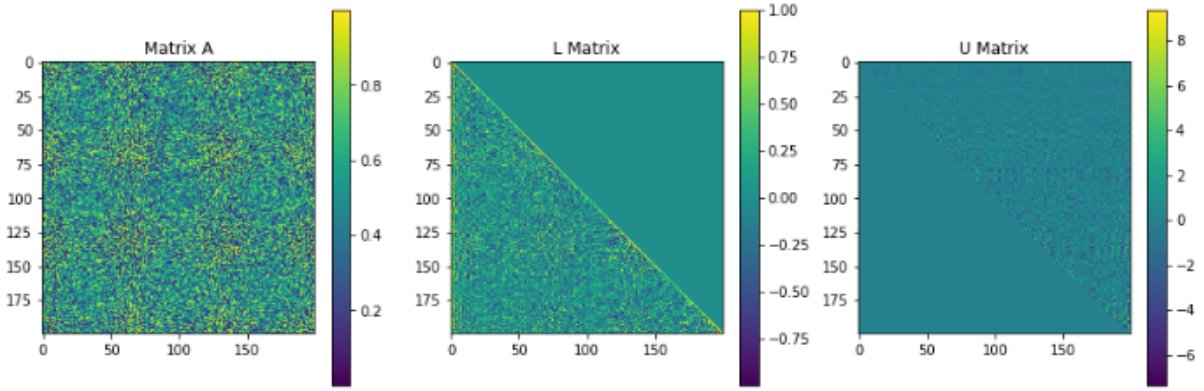
- Python (50x50 square matrix) it took `0.0015251636505126953` seconds for LU decomposition.



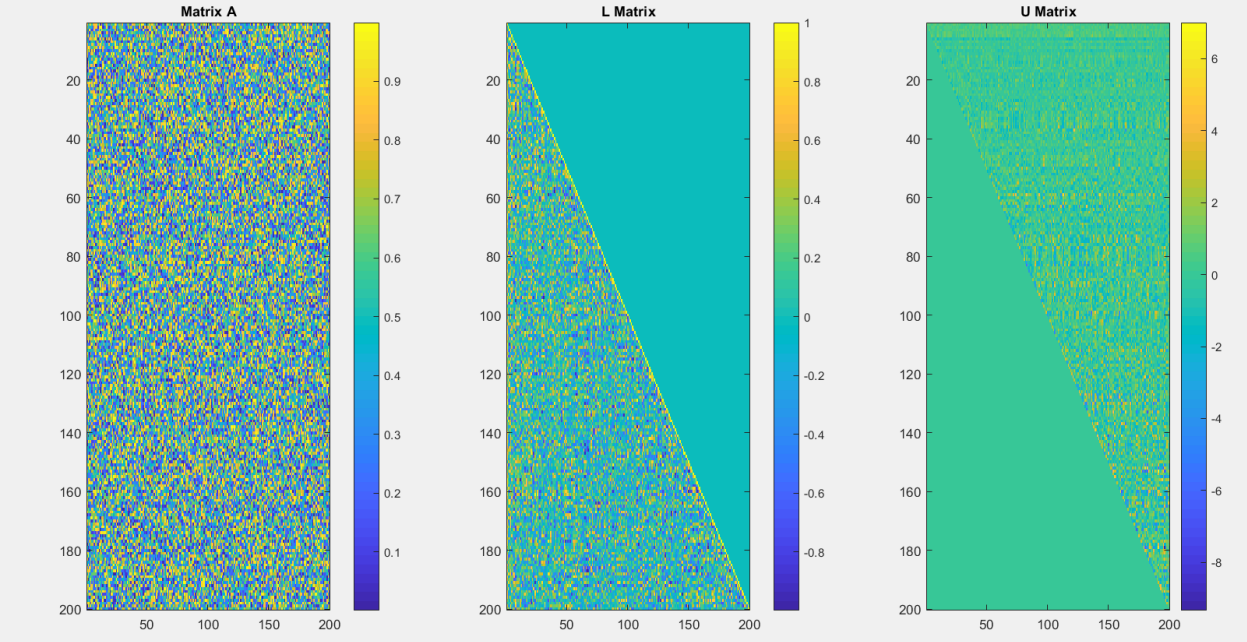
- MATLAB (50x50 square matrix) it took `0.001066` seconds for LU decomposition.



- Python (200x200 square matrix) it took `0.0219419002532959` seconds for LU decomposition.

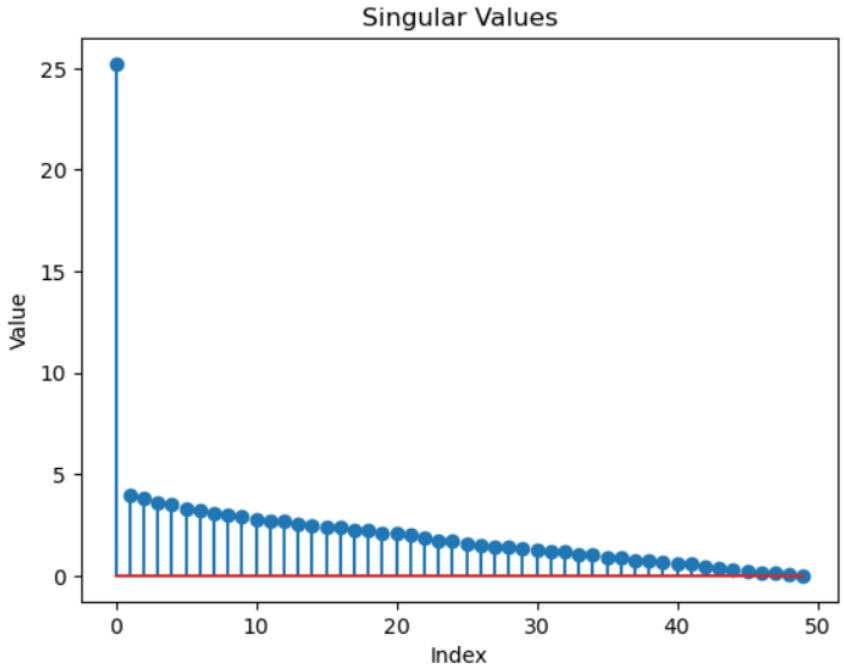


- MATLAB (200x200 square matrix) it took `0.004550` seconds for LU decomposition.

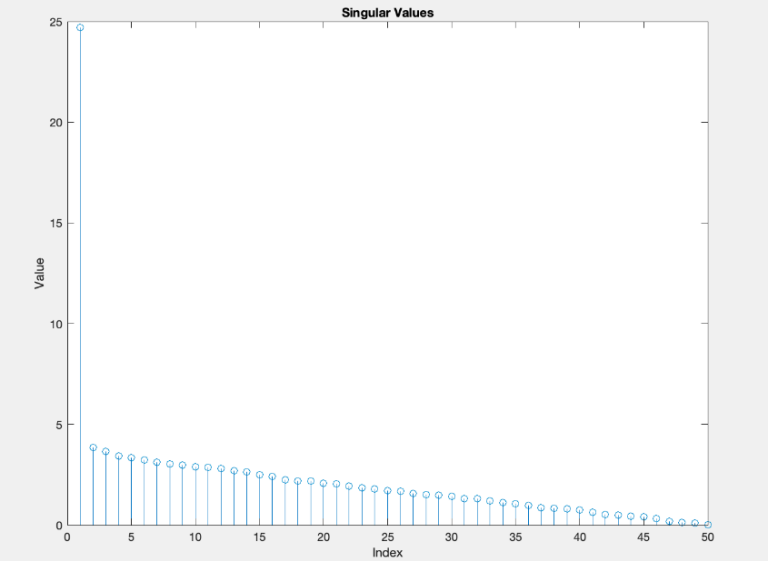


SVD

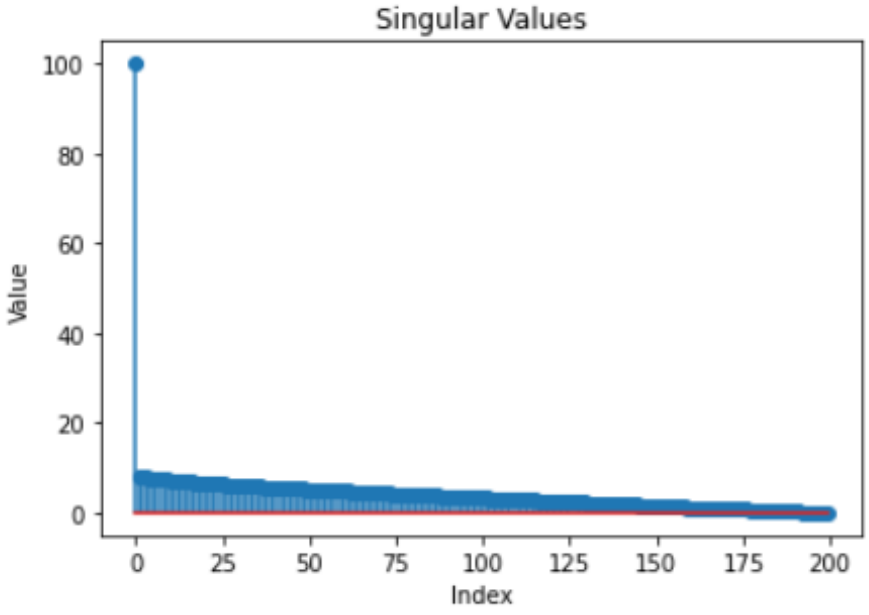
- Python (50x50 square matrix) it took `0.000436091423034668` seconds for SVD decomposition.



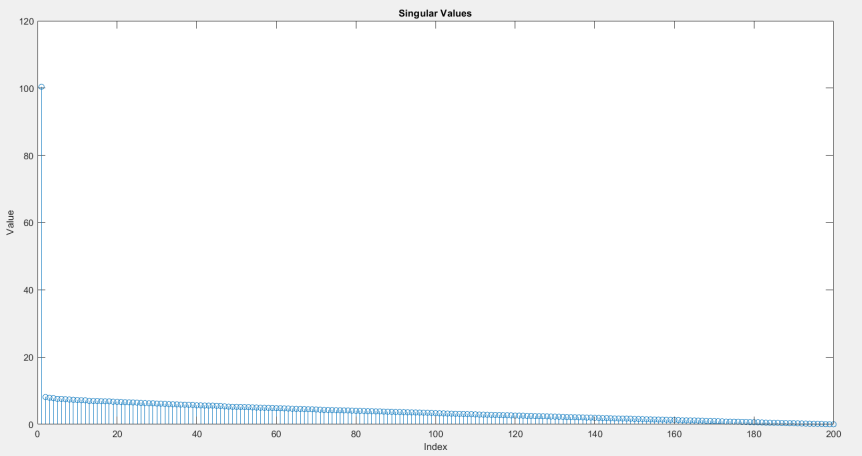
- MATLAB (50x50 square matrix) it took `0.000950` seconds for SVD decomposition.



- Python (200x200 square matrix) it took `0.037528038024902344` seconds for SVD decomposition.

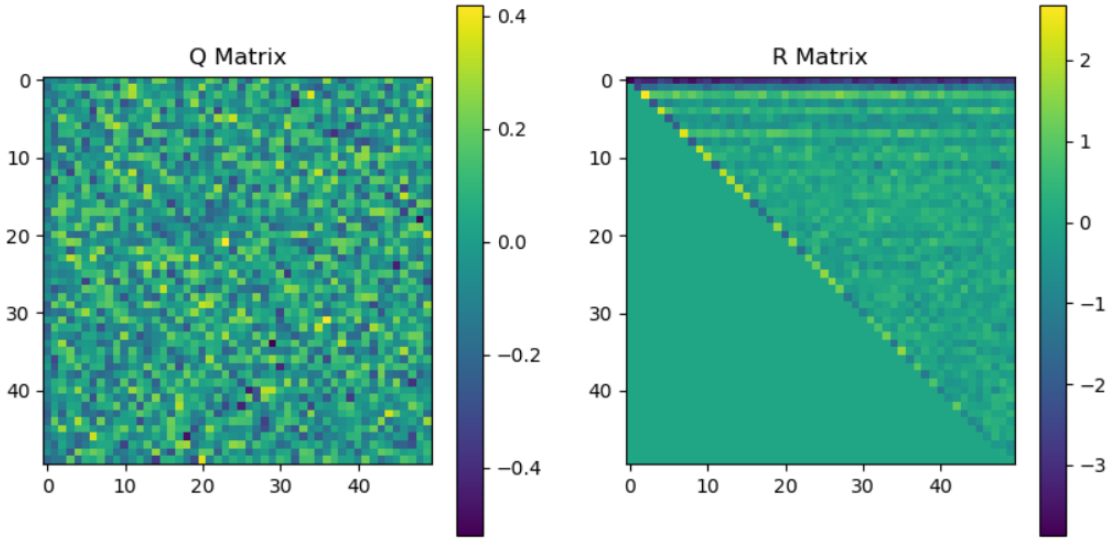


- MATLAB (200x200 square matrix) it took `0.009594` seconds for SVD decomposition.

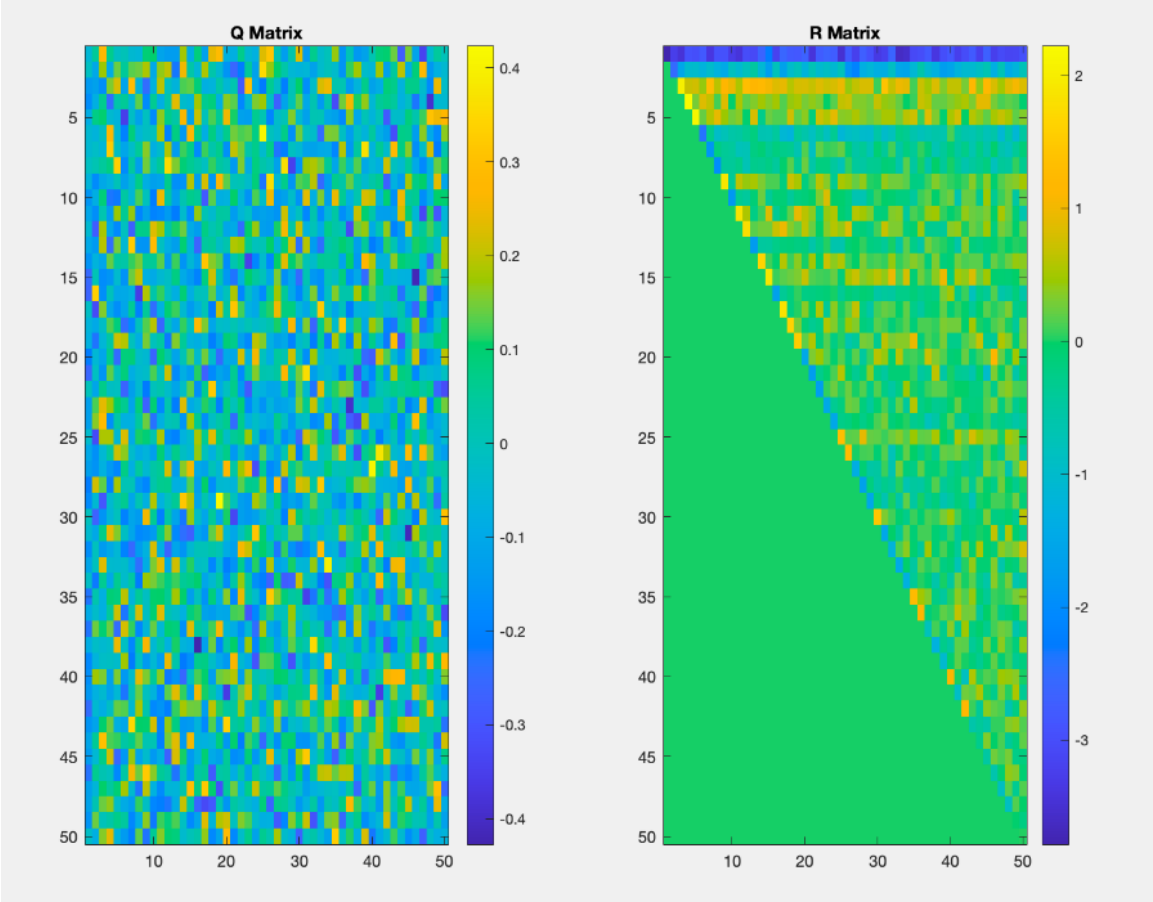


QR Decomposition:

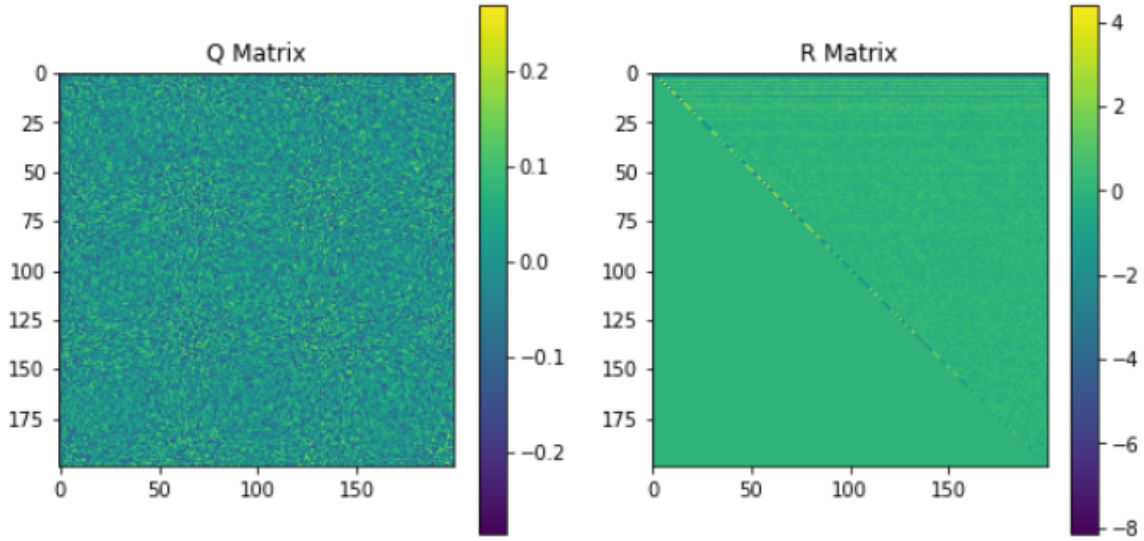
- Python (50x50 square matrix) it took `0.0008428096771240234` seconds for QR decomposition.



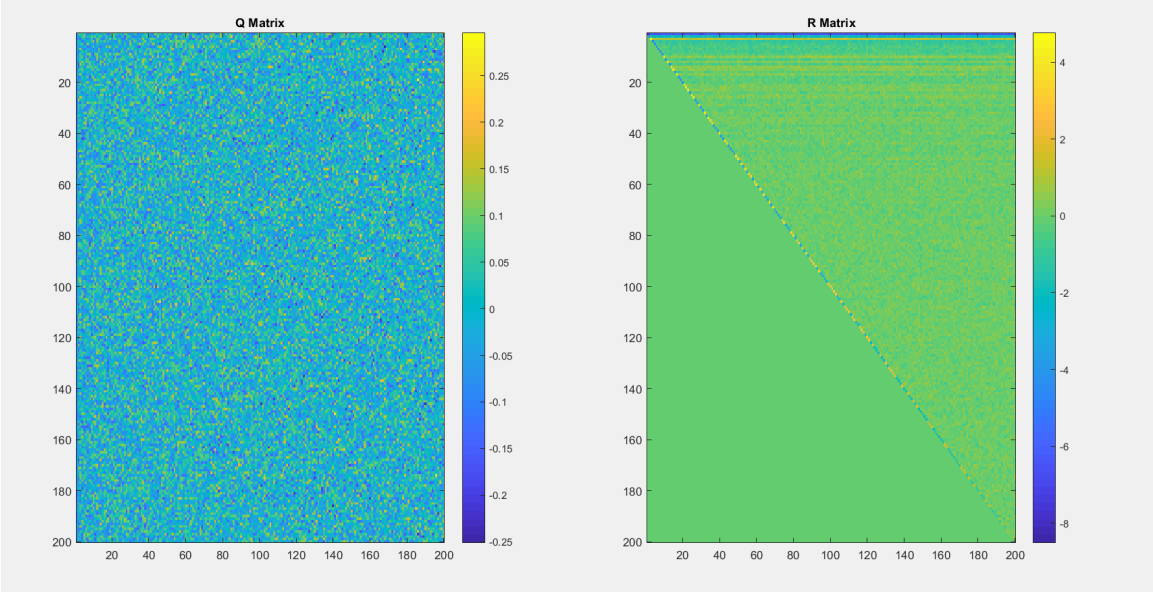
- MATLAB (50x50 square matrix) it took `0.004550` seconds for QR decomposition.



- Python (200x200 square matrix) it took `0.02293848991394043` seconds for QR decomposition.



- MATLAB (200x200 square matrix) it took `0.004539` seconds for QR decomposition.



It was concluded that MATLAB is faster for all the cases (smaller matrix or bigger matrix), except SVD when the matrix is small.

3.5 SOTA Implementation:

Now, we can apply Tensor decomposition on a 50x50 matrix for matrix factorization.

```
# Define the model
class MatrixFactorization(tf.Module):
    def __init__(self, num_users, num_items, hidden_factors):
        self.user_factors = tf.Variable(tf.random.normal([num_us
        self.item_factors = tf.Variable(tf.random.normal([num_it

    def __call__(self):
        return tf.matmul(self.user_factors, self.item_factors, t

# Define the loss function
def loss(model, ratings, mask):
    predicted_ratings = model()
    squared_diff = tf.square(predicted_ratings - ratings)
    masked_squared_diff = tf.multiply(squared_diff, mask)
```

```
    return tf.reduce_sum(masked_squared_diff)

# Create an instance of the model
model = MatrixFactorization(users, items, hidden_factors)
```

3.6 SOTA Results:

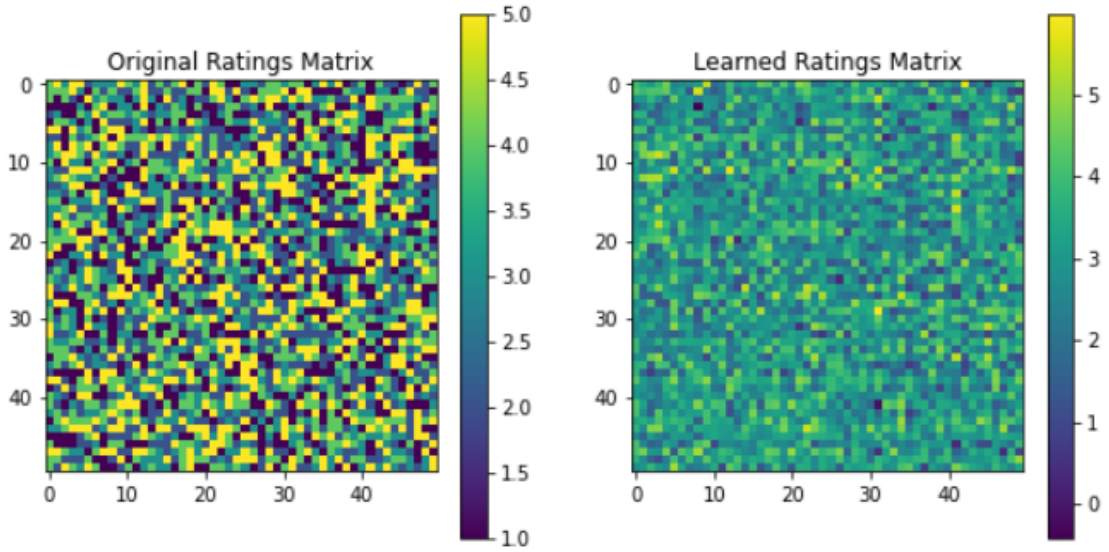
This is the result of tensor decomposition.

We can see that the matrix factorization takes about 2.66 seconds. This is more than traditional approaches, but as our dataset is smaller Tensor decomposition does not work properly.

If more complex datasets are used Tensor decomposition will provide more scalability and dimensionality.

```
Epoch 100/1000, Loss: 7930.25
Epoch 200/1000, Loss: 4083.683837890625
Epoch 300/1000, Loss: 3597.971435546875
Epoch 400/1000, Loss: 3341.28662109375
Epoch 500/1000, Loss: 3155.83642578125
Epoch 600/1000, Loss: 3010.807373046875
Epoch 700/1000, Loss: 2895.38330078125
Epoch 800/1000, Loss: 2806.357177734375
Epoch 900/1000, Loss: 2739.640625
Epoch 1000/1000, Loss: 2689.866455078125
Matrix factorization took 2.66 seconds
Learned User Factors:
[[-3.86206433e-02 -4.09558177e-01 -1.67589188e+00  7.87880957e-01
   9.63994682e-01  1.01088822e+00  1.17423081e+00  5.05018890e-01
   1.73459083e-01 -1.30011931e-01]
 [-5.23521423e-01 -1.63150501e+00 -1.23782504e+00  1.82596612e+00
   1.75470579e+00  3.04497872e-02  1.64378870e+00  1.93487918e+00
   1.71384811e+00  1.16528049e-01]
 [-4.28863436e-01  1.27076316e+00 -3.17256182e-01 -8.64457786e-02
```

Let us now plot the original matrix and learned ratings matrix.



3.7 Traditional NLA Approach vs Modern Approach

- LU, SVD, and QR decomposition follow traditional numerical linear algebra, which is fundamental but struggles with the sheer scale and complexity of contemporary datasets.
- In contrast, contemporary methods such as Randomized algorithms, Tensor decomposition, and machine learning techniques demonstrate superior efficiency and scalability when confronted with intricate data exhibiting non-linear patterns.
- Traditional numerical linear algebra methods excel in scenarios where datasets are small or medium-sized, whereas modern approaches prove most effective for larger datasets.

4. Conclusion

- This project delved into LU, SVD, and QR Decomposition in Matrix Factorization using Python and MATLAB.
- We conclude that contemporary approaches are necessary today, given that most datasets exhibit intricate patterns, challenging traditional NLA in effectively managing such complex datasets.

5. Acknowledgements

We are grateful to Prof. Tsui-Wei Weng, Halıcıoğlu Data Science Institute, San Diego for her continuous support, encouragement, and willingness to help us throughout this proj

6. References

[1] Computationalsciences. (2013, June 6). LU Decomposition. Retrieved from https://computationalsciences.wordpress.com/2013/06/06/lu-decomposition/

[2] Golub, G. H., & Van Loan, C. F. (2013). Matrix Computations. The Johns Hopkins University Press. p. 111

[3] Golub, G. H., & Reinsch, C. (1970). Singular value decomposition and least squares solutions. Numerische Mathematik, 14(5), 403-420. DOI: 10.1007/BF02163027

[4] Trefethen, L. N., & Bau, D. (1997). Numerical Linear Algebra. SIAM Review.

[5] R.C. Gonzalez and R.E. Woods (2017). "Digital Image Processing," Pearson Education.

[6] Y. Koren, R. Bell, and C. Volinsky (2009). "Matrix factorization techniques for recommender systems," Computer.

[7] Golub, G. H., & Van Loan, C. F. (2012). "Matrix Computations." Johns Hopkins University Press.

[8] N. Halko, P.G. Martinsson, J.A. Tropp. (2009) ” Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions.” SIAM Review

[9] R. A. Harshman, M. E. Lund. (1984) "Parallel factor analysis in sensor array processing.” IEEE Transactions on Acoustics, Speech, and Signal Processing

[10] Y. Koren, R. Bell, C. Volinsky (2009) "Matrix factorization techniques for recommender systems.” Computer

UC San Diego

Exploring Matrix Factorization: LU, SVD, and QR Decomposition in Python and MATLAB

Group number : 22
Shobhit, Mansi, Jahnavi
12/07/2023

Sec 1: Introduction

Overview

- Matrix factorization is a cornerstone in linear algebra, providing essential methods for solving linear systems and understanding matrix structures. This project explores three key factorization techniques - LU, SVD, and QR decomposition, using Python and MATLAB, highlighting their applications and comparing their implementations.

Importance of the topic

Understanding these concepts is crucial for several reasons:

1. Solving Linear Systems
2. Data Analysis and Signal Processing
3. Machine Learning and Artificial Intelligence
4. Computer Graphics and Imaging
5. Quantitative Finance
6. Numerical Stability and Efficiency…….. And the list goes on

Sec 2: Problem Formulation

#1 Problem formulation

- At a high level, the problem is about finding simpler, more insightful representations of complex matrix data. In many real-world scenarios, matrices can be large, dense, and difficult to work with directly. The goal is to break down these matrices into components that are easier to handle and interpret.

LU Decomposition: The problem here is to decompose a matrix into lower and upper triangular matrices, which simplifies the process of solving linear systems and inverting matrices.

SVD (Singular Value Decomposition): The aim is to break down a matrix into components that reveal its geometric and algebraic properties, which is vital in data analysis, dimensionality reduction, and image processing.

QR Decomposition: This technique is used to decompose a matrix into an orthogonal matrix and an upper triangular matrix, facilitating the process of solving linear equations and eigenvalue problems.

#2 Relation to Numerical Linear Algebra

- "Numerical Linear Algebra (NLA) focuses on algorithmic solutions for matrix-related problems, essential in computational applications."
- "Matrix factorizations like LU, SVD, and QR are pivotal in NLA for simplifying complex matrices, enhancing computational efficiency and stability."
- "NLA prioritizes stable and efficient algorithms. LU and QR decompositions, for instance, offer robust solutions for linear systems over direct matrix inversion."
- "SVD plays a crucial role in NLA for approximation and error analysis, aiding in applications like dimensionality reduction and data compression."

#3 Approach of Numerical Linear Algebra (NLA)

- For Python, we're using NumPy for matrix operations, SciPy for advanced linear algebra functions, and Matplotlib for visualizations. In MATLAB, we'll utilize its built-in functions for all three decompositions. These tools were chosen for their widespread use and robustness in mathematical computations.

#4 Experiment setup and results

We begin by conducting LU, SVD and QR decompositions on the matrices. And perform efficiency comparison.

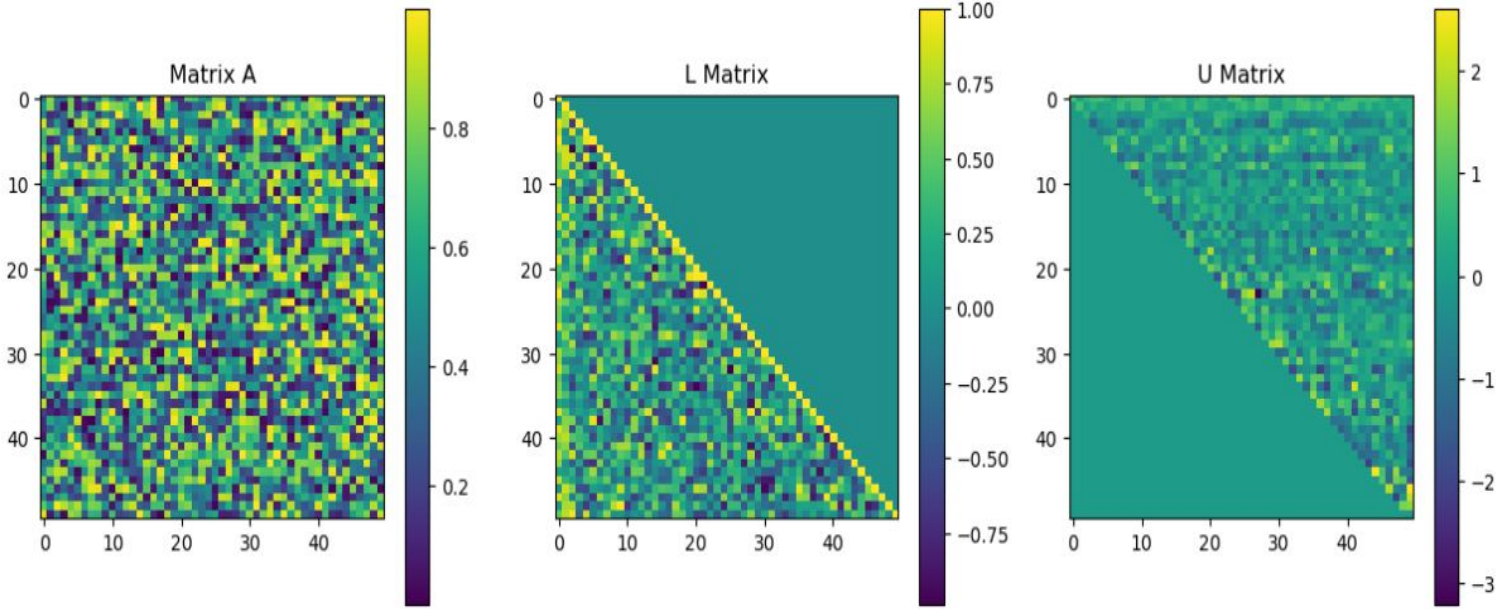
LU DECOMPOSITION

LU Decomposition involves breaking a matrix A into a lower triangular matrix L and an upper triangular matrix U. In Python, we use numpy.linalg.lu and in MATLAB, lu(A).

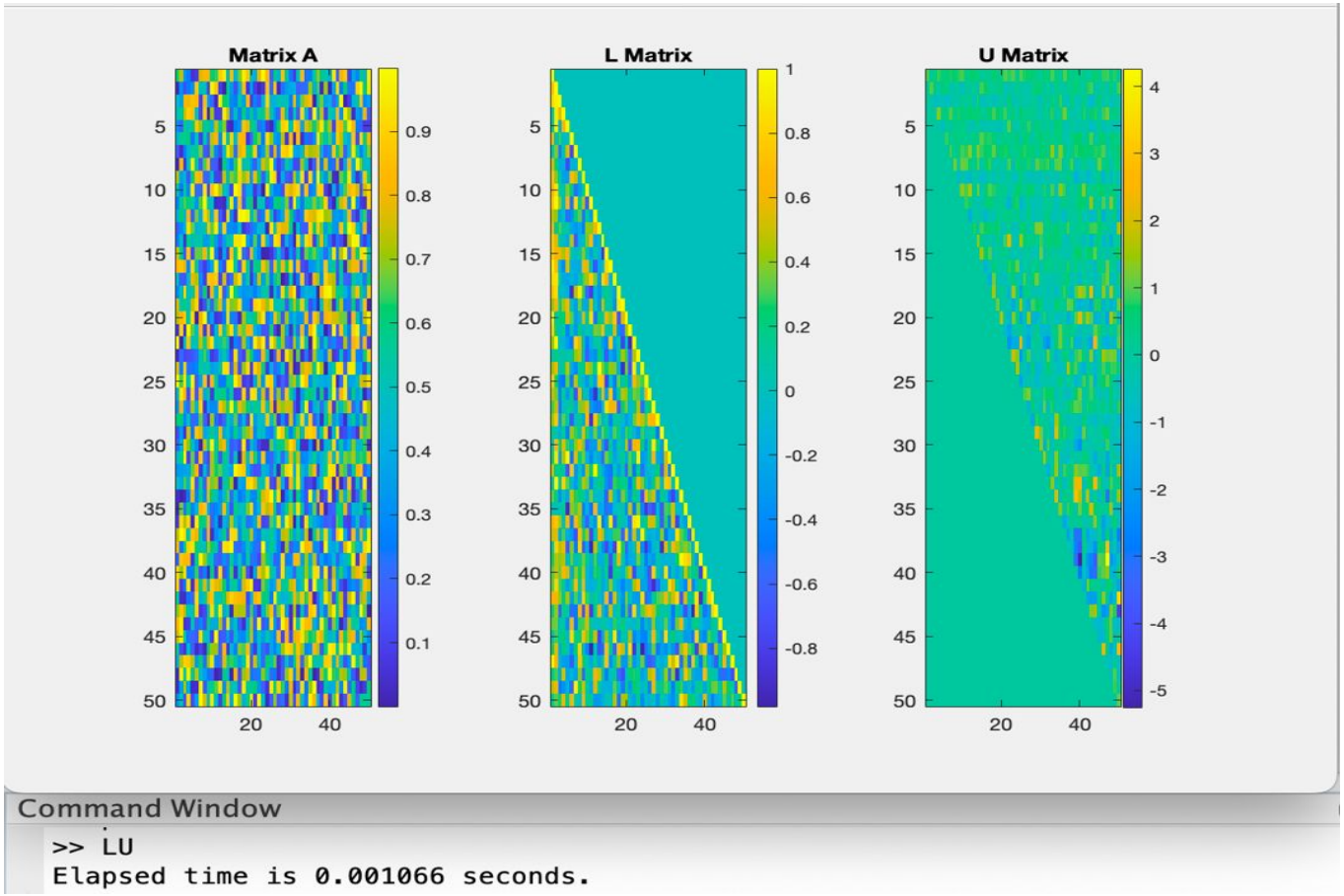
VISUALIZING LU DECOMPOSITION

Python

```
LU Decomposition in Python took: 0.0015251636505126953 seconds
```



Matlab


Matrix A
L Matrix
U Matrix
Command Window
>> LU
Elapsed time is 0.001066 seconds.

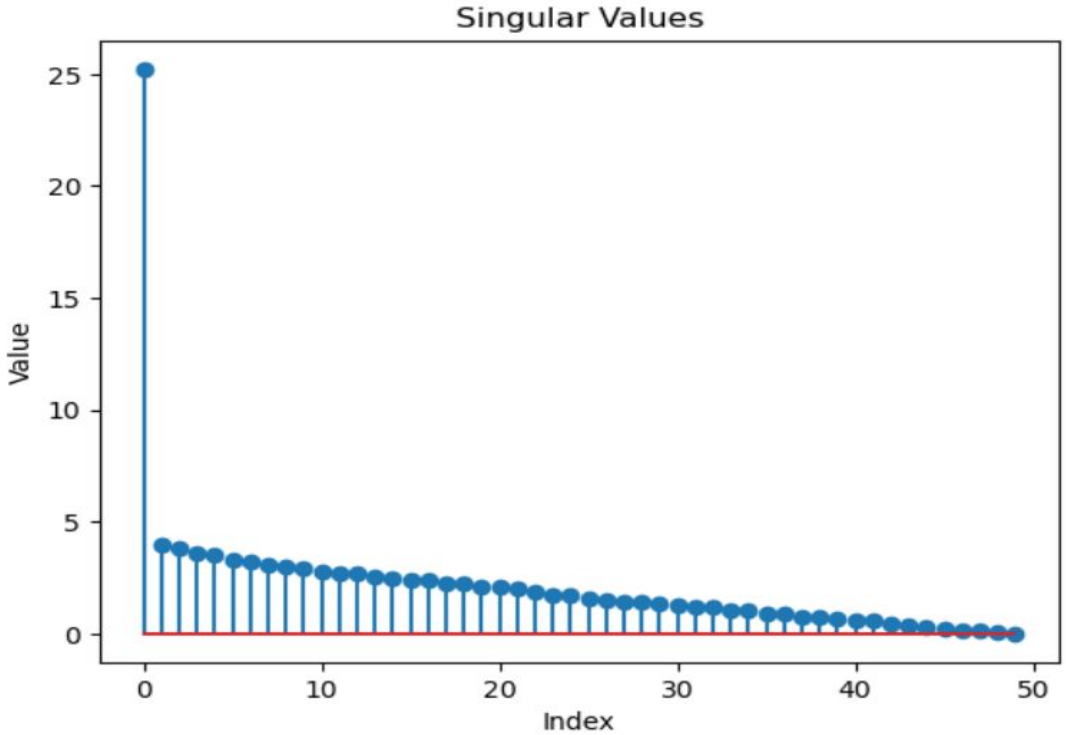
SINGULAR VALUE DECOMPOSITION

SVD decomposes a matrix into three matrices - U, Σ, and V*. Python's numpy.linalg.svd and MATLAB's svd(A) functions are used.

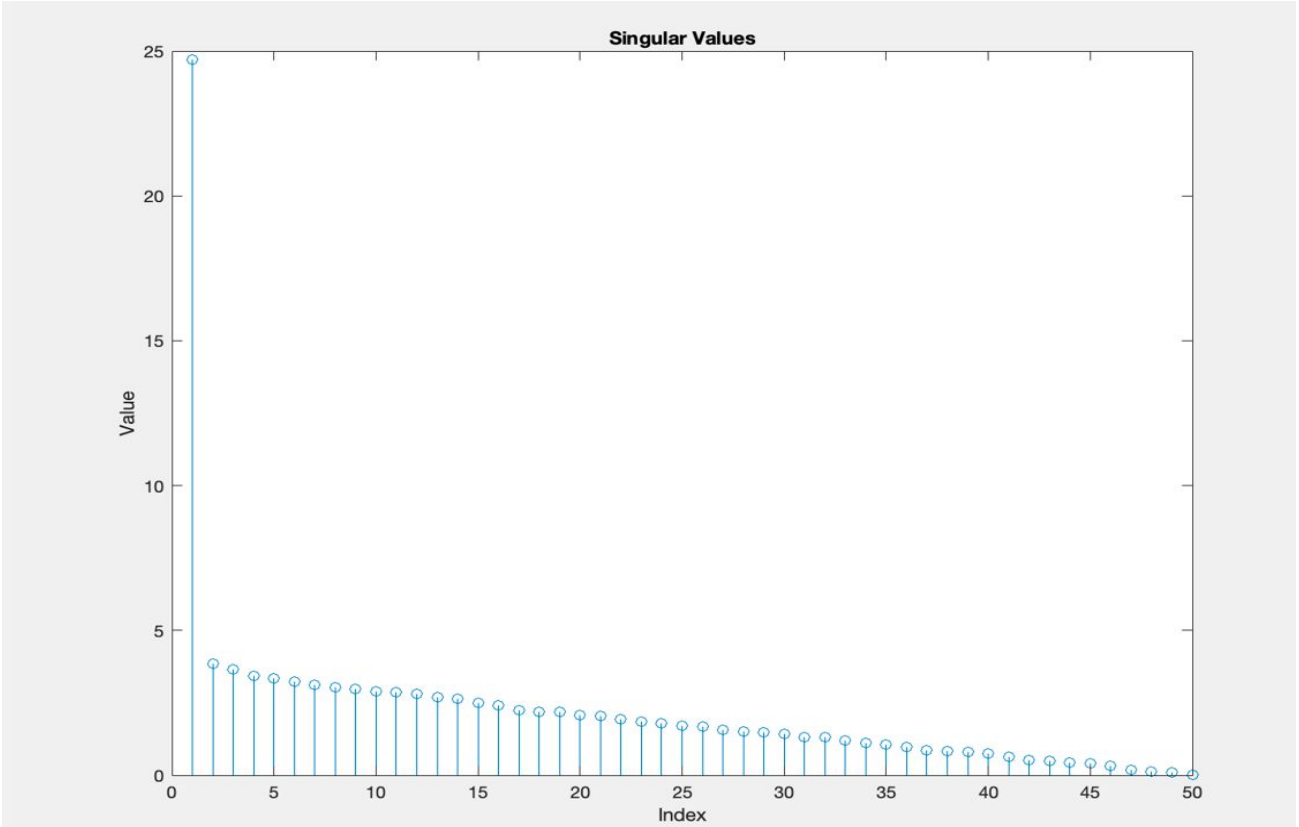
VISUALIZING SVD

Python

```
SVD in Python took: 0.00436091423034668 seconds
```



Matlab

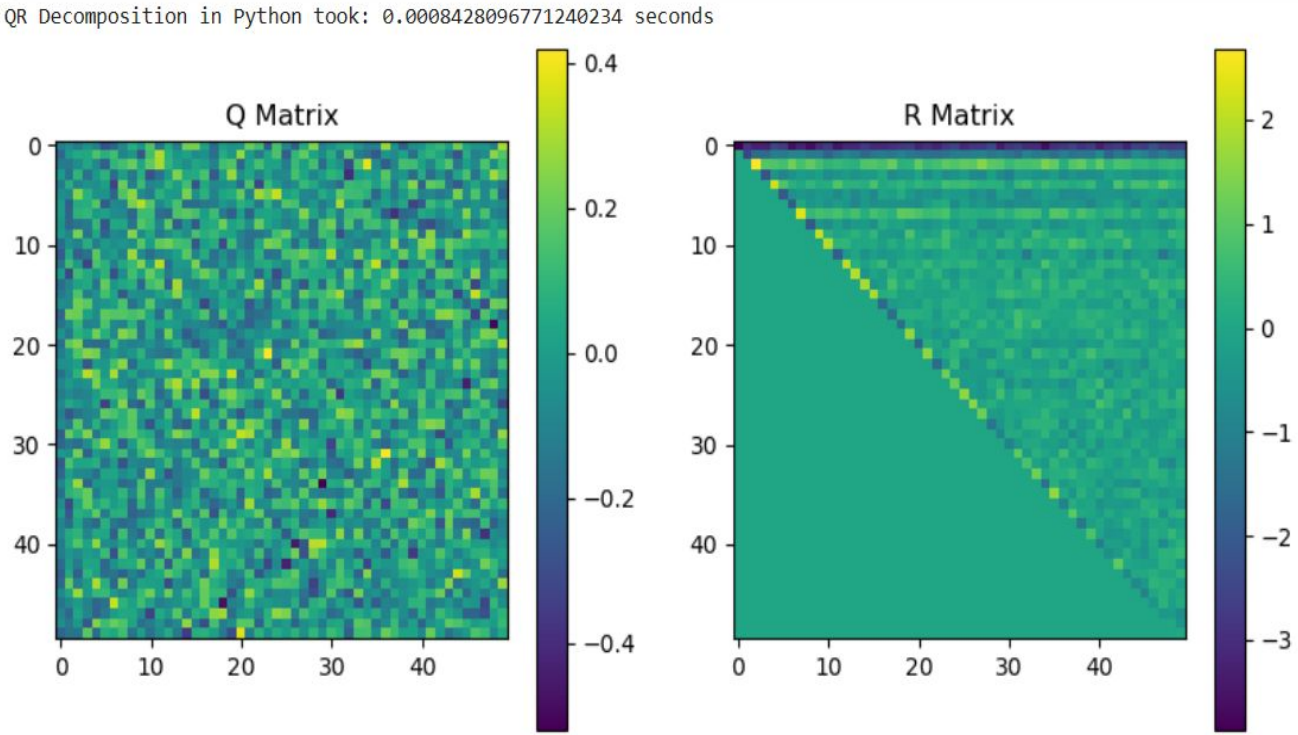

Singular Values
Value
Index
0
5
10
15
20
25
30
35
40
45
50
Command Window
>> SVD
SVD in MATLAB took: 0.000950 seconds

QR DECOMPOSITION

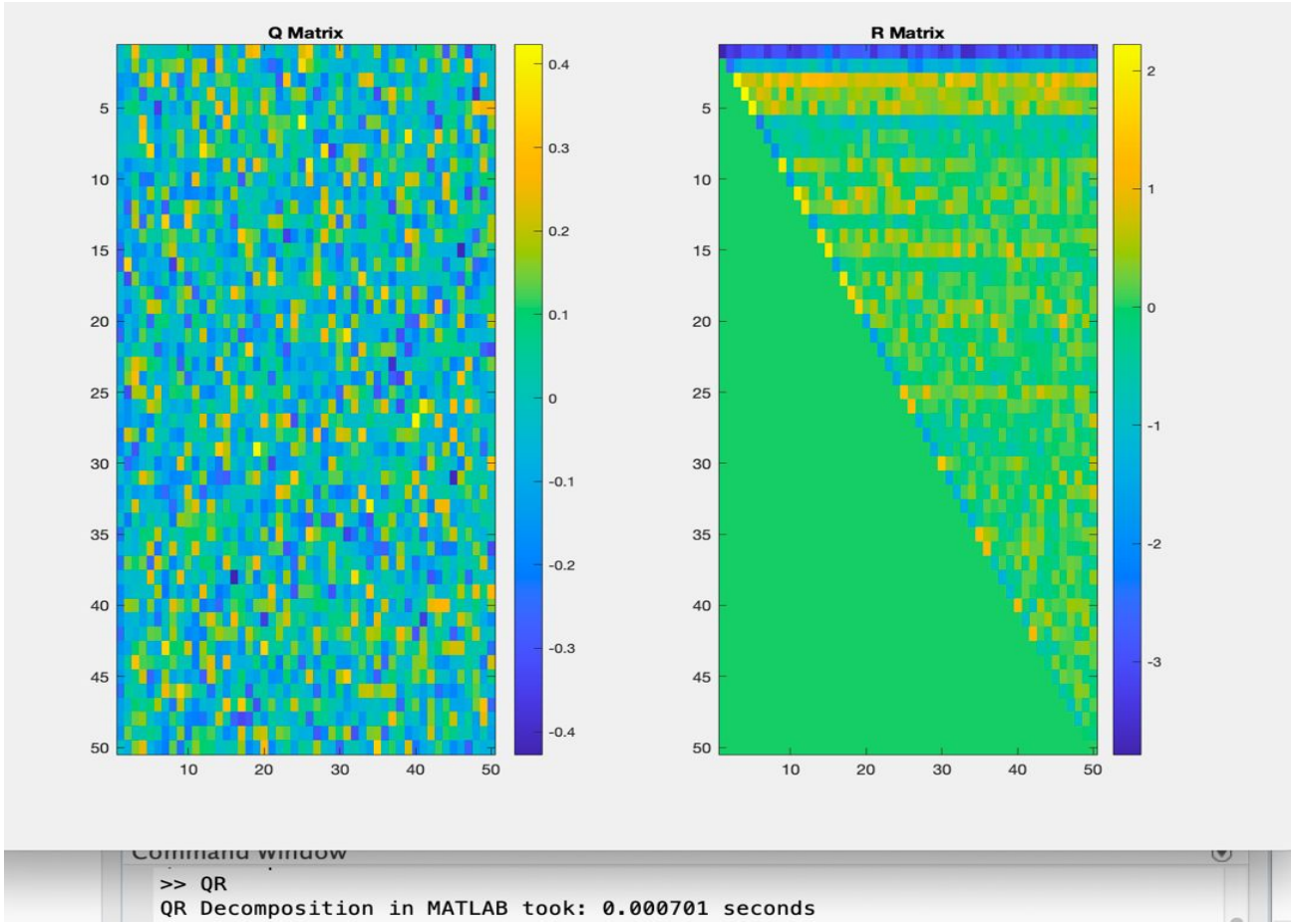
QR Decomposition breaks a matrix A into an orthogonal matrix Q and an upper triangular matrix R. We use numpy.linalg.qr in Python and qr(A) in MATLAB.

VISUALIZING QR DECOMPOSITION

Python



Matlab


Q Matrix
R Matrix
Command Window
>> QR
QR Decomposition in MATLAB took: 0.000701 seconds

COMPARATIVE ANALYSIS

We observed that MATLAB was faster for decompositions in general.

Sec 3: State of the Art (SOTA)

Bridge to SOTA: what are the approaches in the literature?

- **Introduction to Traditional NLA**: "Traditionally, Numerical Linear Algebra (NLA) has focused on LU, SVD, and QR decompositions for matrix factorization. These methods are foundational but often computationally intensive for large-scale data."
- **Transition to SOTA Methods**: "Recent advancements leverage increased computational power and algorithmic innovations, with a significant shift towards machine learning, parallel computing, and handling massive datasets."
- **Overview**: "This presentation explores these cutting-edge methods, comparing them to traditional techniques, highlighting their advantages, limitations, and the challenges they aim to overcome."

Innovative Approaches in Matrix Factorizations

- **Randomized Algorithms**: "Randomized SVD and other probabilistic methods offer faster computations by approximating matrix factors, especially useful for very large matrices. Mathematically, this can be represented as A ~Q(Q^T A)."
- **Tensor Decompositions**: "For multi-dimensional data, tensor decompositions extend traditional matrix factorizations. They break down higher-order tensors into simpler components, though they can be computationally demanding."
- **Machine Learning Techniques**: "Emerging approaches use neural networks to learn and approximate matrix decompositions, offering potential in handling complex, non-linear data patterns."

Comparative Analysis

Comparison Table:

Methods	**Traditional (LU, SVD, QR)**	**Randomized Algorithms**	**Tensor Decompositions**	**Machine Learning Models**
Pros	Accuracy	Speed	Dimensionality	Flexibility
Cons	Scalability	Accuracy	Complexity	Data Requirement
Best Use-Cases	Small to Medium Datasets	Large Datasets	Multi-dimensional Data	Complex Patterns

Sec 4: Concluding remarks

Conclusions

Are Newer Approaches Needed?

Definitely. The evolution of data and computational challenges in the modern era necessitates newer approaches. Traditional methods like LU, SVD, and QR decompositions, while foundational, struggle with the sheer scale and complexity of contemporary datasets. Innovations such as Randomized Algorithms, Tensor Decompositions, and Machine Learning-based techniques are not just beneficial but essential to address these evolving demands.

- **Scalability and Efficiency**: With the exponential growth in data size, particularly in fields like big data analytics, bioinformatics, and neural networks, the efficiency and scalability offered by newer methods are crucial.
- **Complexity of Data**: Modern datasets often exhibit complex, non-linear patterns that traditional linear algebra techniques find challenging. Machine learning models, in particular, show promise in deciphering these complexities.

Limitations and Unresolved challenges

- **Trade-off Between Speed and Accuracy:** While methods like Randomized Algorithms offer speed, they sometimes do so at the cost of accuracy. Balancing this trade-off remains a key challenge.
- **Computational Overhead:** Advanced methods like Tensor Decompositions and certain ML models have high computational overheads, making them inaccessible for resource-constrained environments.
- **Interpretability and Reliability**: Machine learning models, especially deep learning ones, often act as black boxes, leading to challenges in interpretability. Ensuring their reliability and understanding their decision-making processes is a significant hurdle.
- **Data Dependency**: Many of the advanced techniques are heavily data-dependent. They require large datasets to be effective, which might not always be available or feasible to process.

Creative Insights and Future Directions

- **Quantum Computing**: Exploring the integration of matrix factorization techniques with emerging quantum computing technologies could lead to breakthroughs, especially in handling extraordinarily large and complex matrices.
- **Hybrid Models**: Combining the strengths of traditional NLA methods with modern approaches could lead to hybrid models that balance efficiency, accuracy, and computational demands.
- **Focus on Green Computing**: As computational demands rise, so does energy consumption. Developing energy-efficient algorithms will be crucial, aligning with the growing need for sustainable and green computing solutions.
- **Customized Solutions**: Tailoring algorithms to specific industry needs, like real-time data processing in finance or large-scale genomic studies in bioinformatics, could lead to more practical and industry-specific advancements.

References

1. Trefethen, Lloyd; Bau III, David (1997). Numerical Linear Algebra (1st ed.). Philadelphia: SIAM. ISBN: 978-0-89871-361-9.

2. Golub, Gene H. "A History of Modern Numerical Linear Algebra". University of Chicago Statistics Department.

3. von Neumann, John; Goldstine, Herman H. (1947). "Numerical inverting of matrices of high order". Bulletin of the American Mathematical Society. 53 (11): 1021–1099. doi:10.1090/s0002-9904-1947-08909-6.

4. Golub, Gene H.; Van Loan, Charles F. (1996). Matrix Computations (3rd ed.). Baltimore: The Johns Hopkins University Press. ISBN: 0-8018-5413-X.

5. Rickert, Joseph (August 29, 2013). "R and Linear Algebra". R-bloggers.

6. Murray, Riley; Demmel, James; Mahoney, Michael W.; et al. (2023). "Randomized Numerical Linear Algebra: A Perspective on the Field With an Eye to Software". EECS Department, University of California, Berkeley.

7. "Matrix Factorizations". Linear Algebra, Geometry, and Computation.

8. Advances in Numerical Linear Algebra and Its Applications. Special Issue in the journal Mathematics.

Relevant Links

Github link : Shobhitd7/Matrix-decomposition-MATLAB-vs.-Python-_ (github.com)

Documentation link : (1) Exploring Matrix Factorization: LU, SVD, and QR Decomposition in Python and MATLAB (notion.so)

DSC 210 Rehearsal Feedback Form

Group number: 22

Group topic: Exploring Matrix Factorization: LU, SVD, and QR Decomposition in Python and MATLAB

Group members: Shobhit Dronamraju, Mansi Sharma, Jahnavi Patel

Feedback from the Partner Group: 21

- Date: December 5, 2023
- Partner Group members: Har Simrat Singh, Xinyue Wang, Zimo Wang

Please summarize the feedback you get from the Partner group:

- What is the presentation about

The presentation thoroughly explores Matrix Factorization, specifically LU, SVD, and QR Decomposition in Python and MATLAB. It emphasizes the need for newer approaches, citing challenges in scalability and efficiency with traditional methods. The discussion covers limitations, including trade-offs in speed and accuracy, computational overhead, and the interpretability of advanced techniques. Creative insights suggest investigating quantum computing integration, hybrid models, a focus on green computing, and tailoring algorithms to industry-specific needs for future advancements.

- Evaluation criteria
  - (a) The group slightly exceeded the allotted time limit.
  - (b) The main message or purpose of the presentation is clearly communicated and the slides are well-organized, featuring a clear introduction, body, and conclusion.

- (c) The tone of voice is suitable, and the pace of speech is appropriate.
- (d) Team members work well together with mutual support.

- Good things
- 1. The presentation effectively navigates through LU, SVD, and QR Decomposition, showing a deep understanding of complex matrix factorization techniques in both Python and MATLAB.
- 2. The insights on the limitations and challenges demonstrate a thoughtful consideration of real-world applications and provide a well-rounded perspective on the practical implications of matrix factorization.
- 3. The presentation adeptly balances technical depth with clear communication, making the topic easy to understand.

improvements

1. The time limit which was already improved on for the final presentation
2. They were overly prepared, with a wealth of content, to the extent that the pace of speech was slightly fast. But in the final presentation it is well-balanced.

Please summarise the feedback you give to the Partner group:

- What is the presentation about (a short paragraph)

The presentation by Group 21 focuses on the application of Markov Chains, particularly in the context of the Ising Model and Markov Chain Monte Carlo (MCMC) methods. It begins with an introduction to Markov Chains, explaining their importance, definition, and relation to linear algebra through the Perron-Frobenius theorem. The application section delves into the 1D Ising Model, elaborating on its use in the statistical modelling of ferromagnetic materials and phase transitions. The presentation then extends the discussion to the 2D Ising Model, emphasising the utility of MCMC in complex, high-dimensional problems. It highlights the real-world applications of these concepts in various fields, including social sciences and quantum computing.

- Evaluation criteria

  (a) The presentation has all the required elements and even has snippets of most of the math behind the project. The presentation was also very clear.

  (b) All the information was there. The language used in the slides was easily understandable. The experiments section had a lot of plots with appropriate amount of text.

  (c) They were slightly over the time limit

- Good things

- The content is well organised and follows a logical structure. Each part clearly presents it's specific content elaborately.
- The group was able to implement the project and even had innovative new ideas to solve the problems at hand.
- The presentation uses visuals effectively, not only does it supplement the textual information but also makes the presentation more comprehensible.

- To-be-improved things
  - The time limit which was already improved on for the final presentation
  - The slides were very math heavy and were a little hard to comprehend.