ICSI499 Capstone Project Report

PySpady Library Extension

Project Team

Michael Paglia (001413513)
Proshanto Dabnath (001445682)
Micheal Alexander Smith (001466971)
Joseph Regan (001442720)
.....
College of Engineering and Applied Sciences
University at Albany, SUNY

Project Sponsor

Dr. Petko Bogdanov
Data Mining and Management Lab
Department of Computer Science
1400 Washington Ave, UAB 416
Albany, NY 12222

05-07-2024

Acknowledgements

We collectively extend our heartfelt gratitude to Petko Bogdanov, our sponsor, and teacher, whose unwavering support, invaluable guidance, and dedication shaped the direction and quality of this Capstone Project. We are deeply thankful for Maxwell McNeil, for giving of himself so willingly that we might find it easier to make progress, and to Boya for being there and offering helpful insight when we got stuck. Additionally, we wish to express our profound appreciation to Pradeep Atrey, our advisor in this project. Prof. Atrey always made sure there was a clear and organized path to completion of this project. His support was unwavering, and invaluable throughout the duration of the project.

Their contributions, alongside the support of others, have been instrumental along the way of the successful completion of this Capstone Project, and we are profoundly grateful for the opportunity to have worked for and alongside such exceptional individuals.

Abstract

Our team developed software for PySpady, a free-to-use Python library which enables users to leverage state-of-the-art and classical sparse encoding algorithms and methodologies to analyze spatialtemporal data. The library contains features including data compression, missing value imputation, future value prediction, and more. The design and implementation of this software contains the ability to determine the best combination of dictionaries/models/coding optimizers for a dataset based on some learning score. The auto-configuration of the tool is fit with default parameters and returns figures such as residual percent and coefficient precent. Implemented dictionaries include Graph/Discrete Fourier Transform, Spline, Ramanujan, etc. The only implemented model is Low-Rank Dictionary Selection. the included optimization techniques for matrix and tensor decomposition are ADMM and gradient descent

On the front end, the input data will be preprocessed based on a .csv file and a console-based GUI. Methods found in PySpady will be applied on real-world datasets, which our team has prepared, to derive meaningful insights. This library is especially important to those who may not be involved in a computer science-related discipline and allow them to easily interpret their data using the aforementioned algorithms and methodologies.

Contents

1 Problem Analysis

1.1 Project Statement

The goal of the project is to develop software for PySpady. This free-to-use Python library enables users to leverage novel and classical sparse encoding algorithms and methodologies to analyze spatial-temporal data. The library contains features including missing value imputation, future value prediction, and more.

The nature of the problem TGSD is trying to solve is non-trivial, from the size of the datasets to the complexity of the computations and algorithms. However, as we are not directly responsible for the research - it is beyond our domain of knowledge to more directly answer the question.

Temporal graph signals are multivariate time series with individual components associated with nodes of a fixed graph structure [8]. Data of this kind arises in many domains including activity of social network users, sensor network readings over time, and time course gene expression within the interaction network of a model organism [8]. Traditional matrix decomposition methods applied to such data fall short of exploiting structural regularities encoded in the underlying graph and also in the temporal patterns of the signal [8].

1.2 What are existing solutions?

Some existing frameworks do not implement sparse modeling, a key aspect of the TGSD framework [5, 9]. Most existing methodology focuses on modeling either the graph structure or the structure of the temporal signal. The interplay between temporal and structural graph properties gives rise to important behaviors, however, no frameworks currently exist to facilitate a joint representation [8]. For example, traffic levels in a transportation network [4] are shaped by both network locality and the time of the day [8]. The TGSD framework facilitates this joint representation.

Furthermore, irregular sampling as well as missing values in either the time or graph domains also pose key challenges in TGS data analysis [8]. Joint structural modeling of the graph and temporal patterns can be particularly advantageous in settings such as missing or incomplete data [8].

Table 1: Existing Solutions

Solution	**Main Utilities**	**Main Strength**	**Not Used For**	**Weakness**
MCG [5]	Missing value imputation (low-rank, regularizers)	Good missing value imputation when used with graph priors (graph)	Sparse modeling	Requires more coefficients on datasets and ignores periodicity
LRDS [9]	Decomposition; Missing value imputation (low-rank, regularizers); interpolation	Good missing value imputation when used with graph and temporal priors (graph and temporal smoothing)	Sparse modeling	Requires more coefficients on datasets
Gems-HD+ [12]	Decomposition; interpolation	Community preservation with larger models in the dataset	Imputation with many values missing, finer temporal patterns	Less sparse encoding, uses a single dictionary
BRITS [2]	Missing value imputation	Good reconstruction between low/high amounts of missing values	Finer temporal patterns	Too many parameters could cause overfitting of observed values
2D-OMP [11]	Decomposition; imputation	Good sparse reconstruction of a 2D signal as a sum of 2D atoms	Large dictionaries with a large number of coefficients	Outputs only one encoding dictionary, extremely slow with large dictionaries
CCTN [6]	Clustering	Minimize reconstruction error	Interpreting temporal patterns	High number of embedding dimensions
PCA [3]	Clustering	Speed	Rich graph and temporal structures	Too general

1.3 Our solution

The TGSD framework proposes an extremely scalable decomposition algorithm for complete and incomplete data. Its advantages are in matrix decomposition, missing value reconstruction, temporal interpolation, clustering, period estimation, and rank estimation. It shines with both synthetic and real-world data. The framework achieves notable perforce gains for matrix decomposition, missing value reconstruction, temporal interpolation, node clustering, and periodicity detection over baselines. Furthermore, TGSD scales much better than most competitors. For example, for temporal interpolation, TGSD achieves 28% reduction in RMSE compared to baselines when up to 75% of the observations are missing and completes in under 20 seconds on 3.5 million data points [8].

TGSD's contributions are:

- **Generality and novelty**: TGSD is a dictionary-based decomposition framework for temporal graph signals. It is the first method to unify both graph signals and time series analysis [8].
- **Scalability and parsimony**: Compared to high-accuracy baselines, TGSD is on par or better when scaled to larger datasets. Furthermore, TGSD produces low complexity and interpretable representations of the input [8].
- **Applicability and accuracy**: TGSD is useful for data decomposition, missing value imputation, interpolation, clustering, period detection, and rank estimation. The quality of TGSD's output is superior to baselines across frameworks [8].

1.4 Overview of report

This report delves into several key aspects: the proposed System, an in-depth application study, the experimental design and testing process, considerations of legal and ethical practices, allocation of effort, and a glimpse into future endeavors. Each section contributes to a comprehensive understanding of the project's scope and implications.

2 Proposed System/Application/Study

2.1 Overview

In this project, we aimed to develop a data analysis tool catering to diverse user classes including scientists, computer scientists, and enthusiasts. Our tool was built around the pillars of Sparse Coding, and Data Mining. Functional requirements include a user-friendly data management system, a graphical interface accessible via web or local interfaces, and implementation of scalable decomposition and tensor frameworks. Auto-configuration, optimization techniques, outlier detection, and community detection are key features. The technical design encompasses system flow diagrams and implementation details. Development utilized various environments and version control via GitHub. Computer science theories such as signal processing, data mining, and sparse decomposition are integrated into the project, alongside software development fundamentals like source control, modularity, and unit testing, ensuring robustness and flexibility.

2.2 Project Requirements

2.2.1 User Classes

- Our users will include anyone in need of data analysis, particularly those requiring matrix decomposition, imputation of missing values, and related techniques. This could encompass scientists without a strong background in computer science, computer scientists, and computer hobbyists.

2.3 Functional Requirements

2.3.1 Data Management System

- The software will provide a user-friendly data management system that enables users to input their datasets and efficiently store the results generated by the implemented frameworks.

2.3.2 Graphical User Interface (GUI)

- The software will implement a GUI accessible either through a web interface or a local interface, providing users with intuitive interaction and visualization capabilities.

2.3.3 TGSD

- The software will implement TGSD; a highly scalable decomposition framework for complete and incomplete matrices. The algorithm has advantages in matrix decomposition, imputation of missing values, temporal interpolation, clustering, period estimation, and rank estimation [8].

2.3.4 MDTD

- The software will implement the MDTD framework to leverage prior structural information about tensor modes by coding dictionaries to obtain sparsely encoded tensor factors. MDTD learns more concise models in comparison to dictionary-free counterparts, improving reconstruction quality, missing value imputation, and tensor rank estimation [7].

2.3.5 Auto-configuration

- The software will implement an auto-configuration tool that will have default parameters, an estimated time to complete, and return explanatory figures to the user. Dictionaries include Graph/Discrete Fourier Transform, Ramanujan, Spline-Basis, etc.

2.3.6 Optimization Techniques

- The software will implement two optimization techniques to the TGSD [8] and MDTD [7] frameworks: Alternating Direction Method of Multipliers (ADMM) and Gradient Descent. ADMM is an optimization algorithm that solves problems by alternatively minimizing over subsets of the variables, resulting in faster convergence compared to methods that update all variables simultaneously [1] and is the default optimization algorithm used in TGSD [8]. Gradient Descent takes steps proportional to the negative of the gradient of the objective function at the current point to find a local minimum [10]. PySpady users have the choice of either ADMM or Gradient Descent based optimization for their data. For interested readers, the derivation of the gradient update rule for sparse encoding matrices can be found in Appendix 8.

2.3.7 Outlier Detection

- The software will use TGSD and MDTD to implement outlier detection. This functionality allows users to detect and identify anomalies within their datasets. Outlier detection plays a crucial role in various domains, including anomaly detection, fraud detection, and quality control, making it a valuable addition to the software's feature set.

2.3.8 Community Detection

- The software will use TGSD and MDTD to implement community detection. This functionality allows users to detect and identify communities within their datasets. Community detection plays a crucial role in various domains of complex networks, grouping nodes by their structural properties, making it a valuable addition to the software's feature set.

2.4 Technical Design

Computer science theories like signal processing, data mining, and sparse decomposition fuel practical applications. Signal processing, utilizing techniques like Fourier transforms, aids tasks such as image and audio analysis. Data mining extracts insights from large datasets through methods like clustering. Sparse decomposition optimizes algorithms for sparse data scenarios, benefiting machine learning and scientific computing. In software development, principles like source/version control, exemplified by GitHub in projects like PySpady, facilitate collaboration and organization. PySpady's modular design promotes integration and code reusability. Rigorous unit testing ensures dependable performance. The fusion of theory with pragmatic engineering drives innovation in computer science.

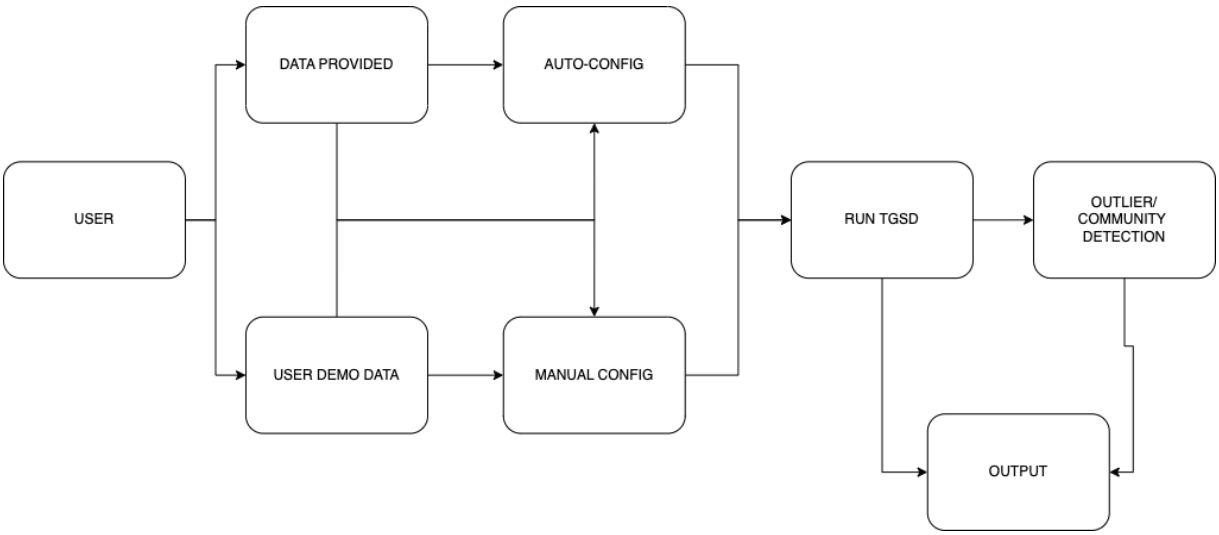


Figure 1: System Flow Diagram

- The user can use demo data or input data.
- The user can use manual config or auto config.
  - (*) The user can run TGSD.
  - (*) The user can run outlier detection.
- Stared points (*) denote final output.

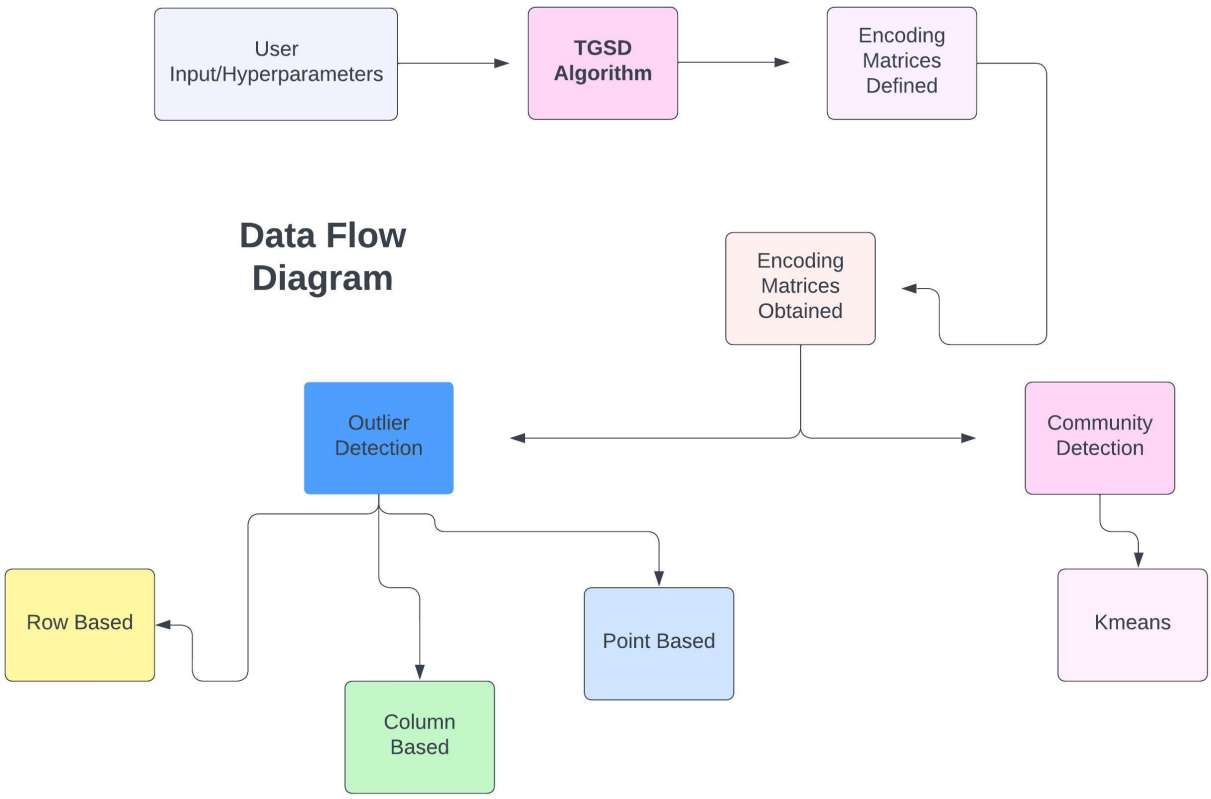


Figure 2: Config Diagram

- The user inputs data (could be demo data).
- The user can use manual config or auto config.
  - Process 1: Loads data to memory and reads the config.
  - Process 2, 3, and 4 generate required inputs if they are not provided.
  - The 4 databases correspond to required inputs for TGSD.
  - Process 5 is to run TGSD/Outliers.

2.5 System Implementation

Our team used a variety of development environments - some of us used PyCharm, which others used VSCode. We handled our source/ version control using GitHub, where we maintained individual branches that we would merge with main when combining our code.

2.6 Use of Computer Science Theory and Software Development Fundamentals

2.6.1 Use of Computer Science Theories

Signal Processing A vital aspect of computer science, involving the manipulation, analysis, and interpretation of signals. Techniques like Fourier transforms, filtering, and noise reduction are fundamental for tasks like image and audio processing, telecommunications, and data compression.

Data Mining Data mining extracts valuable insights from large datasets through various techniques like clustering, classification, and association rule mining. It uncovers patterns, trends, and relationships within data, aiding decision-making processes in diverse fields such as marketing, finance, healthcare, and scientific research. Data mining is integral to modern data-driven approaches.

Sparse Decomposition Sparse programming focuses on optimizing algorithms for problems involving sparse data, where most elements are zero. It leverages techniques like compressed sensing and sparse matrix operations to efficiently solve problems in fields like machine learning, signal processing, and scientific computing. Sparse programming plays a key role in optimizing resource utilization and computational efficiency.

2.6.2 Use of Software Development Fundamentals

Source/Version Control The PySpady team used GitHub for source control, where the main repository can be found at this link. Each team member had their own branch and periodically made "pushes" and synced with the main branch.

Modularity PySpady's modular design allows users to easily integrate its classes and functions into their projects. By organizing components based on their purpose, PySpady provides a flexible and intuitive framework that simplifies the development process and promotes code reusability.

Unit Testing PySpady's development process prioritizes unit testing and numerical stability. Helper methods have been carefully constructed to validate results and ensure accurate computations across a wide range of inputs.

3 Experimental Design and Testing

In the following subsections, we provide a detailed discussion of our experimental setup, dataset, and results, along with an in-depth analysis of our findings.

3.1 Experimental Setup

The main objectives of the experiments are twofold: first, to validate the correctness of the PySpady library by comparing its output with the established MATLAB source code for matrix and tensor signal decomposition; and second, to evaluate the scalability and efficiency of the Python library as the input size grows exponentially. By conducting these experiments, we aim to demonstrate the reliability and performance of PySpady, ensuring its suitability for handling large-scale signal decomposition tasks.
A MacBook Air (M1, 2020) was used as the testing machine to conduct all experiments. The PySpady library was set up in a virtual environment, and the PyCharm IDE was employed for code development and execution.
It is important to note that Experiments #2 through #5 share the same input data. This approach ensures consistency and comparability across the different performance metrics of PySpady. The input data consists of synthetically generated temporal graph signals and adjacency matrices, created using a pre-existing MATLAB implementation for demonstration purposes.
Three matrix sizes were used as temporal graph signal inputs: a 100×100 matrix, a 500×500 matrix, and a 1000×1000 matrix. For each matrix of size $n \times t$, a Graph Fourier Transform (GFT) dictionary of size $n \times n$ and a Discrete Fourier Transform (DFT) dictionary of size $t \times t$ were employed for temporal graph signal decomposition. Additionally, for each matrix tested, rank values of $k \in \{5, 10, 20\}$ were utilized, such that the temporal graph signal reconstruction of $X \in \mathbb{R}^{n \times t}$ can be approximated as $X \approx \Psi Y W \Phi$, where $\Psi \in \mathbb{R}^{n \times m}$ is a fixed graph dictionary, $\Phi \in \mathbb{R}^{s \times t}$ is a fixed temporal dictionary and $Y \in \mathbb{R}^{m \times k}$ and $W \in \mathbb{R}^{k \times s}$ are corresponding sparse encoding matrices.

3.1.1 Experiment #1: Verifying MATLAB vs. Python Output

The primary objective of this experiment is to validate the correctness of PySpady's output by comparing it with the results obtained from the pre-existing MATLAB source code for matrix and tensor signal decomposition. The MATLAB code, which serves as a reference, is available at this link. This experiment was conducted as a prerequisite to the subsequent experiments,

once a minimum viable product for PySpady was developed. The goal was to ensure that the reconstructed matrix/tensor's Euclidean norm matched the MATLAB and Python outputs.

3.1.2 Experiment #2: Runtime Tests vs. Nodes for Different Matrix Ranks

This experiment aims to assess the runtime performance of PySpady across various numbers of nodes and matrix ranks, providing valuable insights into its scalability and efficiency. The primary objective is to determine whether the matrix decomposition runtime is adversely affected as the number of matrix elements increases exponentially.

3.1.3 Experiment #3: Memory Allocation vs. Nodes for Different Matrix Ranks

This experiment focuses on assessing the memory allocation of PySpady across various numbers of nodes and matrix ranks, helping to understand its resource utilization. The goal was to determine whether or not the memory allocation of temporal graph signal X scaled as the number of matrix elements of X increased exponentially.

3.1.4 Experiment #4: RMSE vs. Nodes for Different Matrix Ranks

This experiment aims to analyze the Root Mean Square Error (RMSE) of PySpady's output across different numbers of nodes and matrix ranks, providing a measure of its accuracy. The objective is to calculate the RMSE between the original temporal graph signal X and its reconstructed version obtained through PySpady's decomposition and reconstruction process. A lower RMSE indicates a higher fidelity between the original and reconstructed signals, demonstrating the library's ability to preserve the essential information present in the temporal graph signals.

3.1.5 Experiment #5: Number of Non-Zero Elements vs. Nodes for Different Matrix Ranks

This experiment investigates the sparsity of the decomposed signals by examining the number of non-zero elements in the output across different numbers of nodes and matrix ranks. Sparsity is a desirable property in signal processing, as it allows for efficient storage, transmission, and computation of the decomposed signals. A lower number of non-zero elements, relative to the size of Y and W, indicate that the encoding matrices are sparser.

3.2 Dataset

The dataset used for all the experiments mentioned prior was synthetically generated using a pre-existing MATLAB library. This dataset is readily available within the PySpady library,

allowing users to test the library's functionality with synthetic data immediately upon launch. Alternatively, the dataset can be downloaded externally from this link.
The synthetic temporal graph signal used in the experiments had an initial size of 175x200. However, to accommodate the testing of matrices with different dimensions, the signal was resized as needed. The input data was randomly generated and then augmented with a small amount of Gaussian noise for variance.
Throughout the experiments, the temporal graph signal decomposition process employed Graph Fourier Transform (GFT) and Discrete Fourier Transform (DFT) dictionaries. The source code for generating these dictionaries was originally available in the previously linked MATLAB file but was then implemented and integrated into the PySpady library as well.

3.3 Results and Analysis

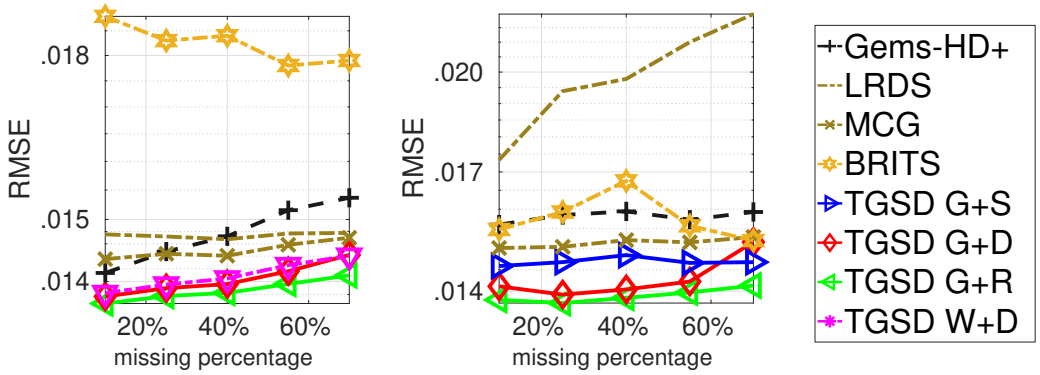
Table 2: Existing Solutions

Task	Baselines
Decomposition	MCG, LRDS, GEMS-HD
Imputation	MCG, LRDS, GEMS-HD, BRITS
Interpolation	MCG, LRDS, GEMS-HD, BRITS

McNeil et. al [8] compared the temporal graph signal decomposition algorithm's performance for matrix decomposition to three baselines: MCG [5], LRDS [9], and Gems-HD [12]. MCG and LRDS utilize rank minimization and graph regularization, while Gems-HD is a graph signal processing technique. As Gems-HD does not handle missing values directly, missing values were imputed prior; this approach can be denoted as Gems-HD+. BRITS [2] was also used to interpolate missing values in times series using a recurrent neural network.

3.3.1 Failure Cases

Potential failure cases for PySpady include overfitting due to using a rank value (K) that is too large, underfitting when K is too small, and suboptimal results from improper hyperparameter selection. These issues can be mitigated through careful hyperparameter tuning, which is where PySpady's auto-configuration option may be useful. Additionally, performance problems may arise when working with extremely large datasets in the PyCharm IDE, leading to slowdowns or crashes.



(a) Synthetic-impute (b) Synthetic-interpolate (c) Legend

Figure 3: Comparison of quality for missing value imputation and interpolation [8]

As seen in Figure 3, McNeil et al. [8] compare the performance of TGSD with various baselines on synthetic data with strong periodicity. They find that the GFT + Ramanujan combination of TGSD outperforms all other methods, followed by MCG and LRDS, while Gems-HD+ and BRITS show degraded performance, especially when a large number of values are missing.

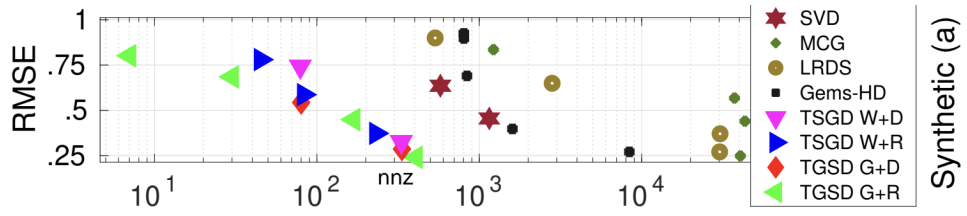
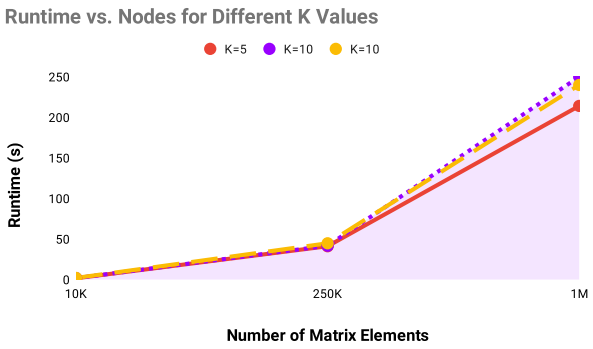
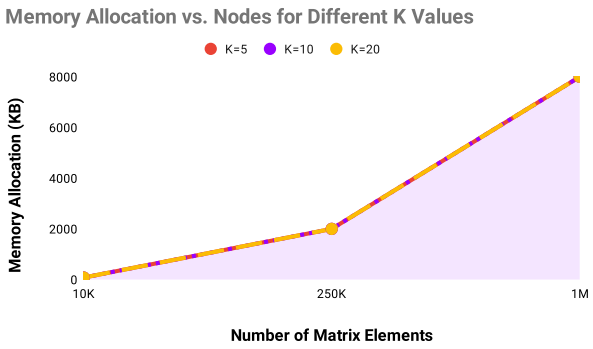


Figure 4: Decomposition quality as a function of model size [8]

In addition to the baselines provided, McNeil et. al. [8] use singular value decomposition (SVD) as a reference. They evaluate the effectiveness of their approach by measuring the root mean square error (RMSE) of the reconstructions as a function of the number of non-zero model coefficients (NNZ). As shown in Figure 4, TGSD variants outperform all baselines with a low NNZ, which is attributed to the joint encoding scheme that requires fewer coefficients to capture the signal trends effectively.



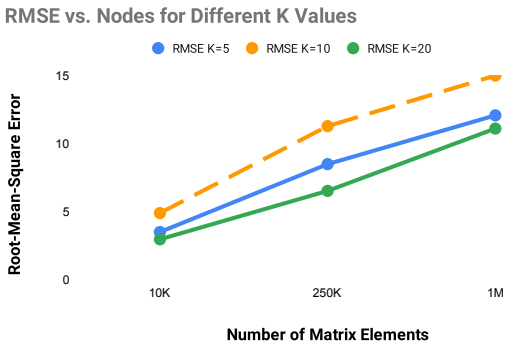
(a) Runtime (seconds) vs. number of matrix elements for different rank values.



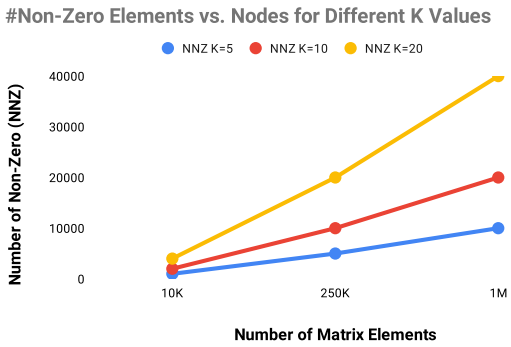
(b) Memory allocation (kilobytes) vs. number of matrix elements for different rank values.

Figure 5: Runtime and memory allocation comparison for different rank values and matrix sizes.

PySpady was evaluated for runtime efficiency and memory allocation across varying rank values (K) and matrix sizes in Figure 5. The evaluation of PySpady's runtime efficiency and memory allocation across varying rank values (K) and matrix sizes confirmed the hypothesis that larger rank values lead to longer runtimes due to the increased number of reconstructed coefficients. However, the memory overhead remained constant for each matrix size, regardless of the rank value. The runtime scaled well with increasing matrix size, while the memory allocation remained efficient, even for large matrices.



(a) RMSE vs. number of matrix elements for different rank values.



(b) Number of non-zero elements vs. number of matrix elements for different rank values.

Figure 6: RMSE and number of non-zero elements comparison for different rank values and matrix sizes.

The performance of PySpady was evaluated using the root mean square error (RMSE) and the number of non-zero elements (nnz) across different rank values (K) and matrix sizes in Figure 6. The RMSE generally improved with increasing rank value, as higher rank values allow for more accurate approximations. However, this comes at the cost of a larger number of non-zero elements, which can impact storage requirements and computational efficiency, thus resulting in a less sparse decomposition.

4 Legal and Ethical Practices

4.1 Legal Considerations

- PySpady is an open source project protected by the MIT license
- The license grants users permission to use, modify, and distribute the software for any purpose, both commercially and non-commercially, without any restrictions.
- The license includes a disclaimer stating the software is provided "as is" without any warranty of fitness for a particular purpose, protecting the team from liability in case the software causes any damage or does not meet the user's expectations
- The license requires that the user acknowledges that the author is not liable for any claims or damages from the use of the software.
- The license requires that a copy of the license and copyright notice be included in all copies or substantial portions of the software.
- The license allows the team to retain copyright ownership of the software while granting users the right to use, modify, and distribute it under the terms of the license, ensuring the team received credit for their work and maintains control over the distribution of the project.

4.2 Ethical Considerations

- PySpady is open source and publicly available on GitHub for use, contribution, and citation.
- PySpady does not store any data from the user nor any results the library obtains after processing algorithms on the data.
- PySpady applies sparse coding techniques, intended for large datasets, and obtains relevant results. Users should be aware of the nature of the results of the algorithms which can be found on the README markdown file on the GitHub.

5 Effort Sharing

Joint tasks included meetings with our sponsors, team meetings on Zoom and in-person to discuss elements of the project or to complete task, and completing reports and presentations. Coordination for joint tasks was facilitated through transparent discussions within the Slack channel, establishing clarity regarding timing and logistics.

- **Joseph Regan**: He implemented TGSD clustering. He was essential in milestone reports and was involved in the final report and presentations.
- **Michael Paglia**: He worked on the initial setup of the Python script. He helped extend the GUI as new features were added. He implemented autosearch, optimizers, and outlier and community detection. He was involved in milestone reports, presentations, and the final report.
- **Michael Smith**: He created the format for the config files and the script to read them. He, originally, implemented the GUI and helped update it as new features were added. He implemented screening. He was involved in milestone reports, presentations, and the final report.
- **Proshanto Dabnath**: He helped code the config and worked on dataframe integration. He was involved in milestone reports, presentations, and the final report.

Table 3: Effort sharing

Team size	Joint efforts	Joseph Regan	Michael Pagalia	Michael Smith	Proshanto Dabnath
4	J (≈30%)	I (≈15.5%)	I (≈19.5%)	I (≈17.5%)	I (≈17.5%)

J = description of tasks jointly performed
I = description of tasks individually performed

6 Conclusion and Future Work

At the beginning of this project, the team were novices to the ideas of spatial-temporal data, sparse dictionaries, and signal decomposition. However, through a dedicated period of learning and with invaluable support from our sponsors, we delved deep into these concepts and developed a comprehensive Python library tailored for sparse encoding. The team successfully implemented the TGSD algorithm, expanded the library with additional dictionaries and optimizers, and integrated various downstream tasks, a screening method, a smart auto configuration system, a data management system, and a user-friendly command line GUI.

The team envisions this library as a powerful tool for data analysis, capable of reconstructing massive real-life datasets and unveiling the intricacies hidden within them, thus offering insight into the underlying structures within the data. In regards to future work for the project, the library's versatility can be broadened with the incorporation of more dictionaries and optimizers, alongside the integration of promising ideas such as graph learning, greedy OMP and 2D OMP.

None of this would have been possible without the unwavering support and guidance of the teams sponsors, whose expertise and encouragement propelled the team forward at every juncture. The team extends their sincerest gratitude to them for trusting the team with their project, believing in the team's vision and empowering the team to turn it into a reality. As the team concludes this final milestone, they do so with a sense of pride in what they accomplished and with eager anticipation for the future of this project. With every step forward, the team remains committed to pushing the boundaries of what is possible, driven by our passion for innovation and discovery.

7 Works Cited

References

[1] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[2] W. Cao, D. Wang, J. Li, H. Zhou, L. Li, and Y. Li. Brits: Bidirectional recurrent imputation for time series. In *Advances in Neural Information Processing Systems 31*, pages 6775–6785, 2018.

[3] 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*, 374(2065):20150202, 2016.

[4] M. Joshi and T. H. Hadi. A review of network traffic analysis and prediction. *arXiv preprint arXiv:1507.05722*, 2015.

[5] V. Kalofolias, X. Bresson, M. Bronstein, and P. Vandergheynst. Matrix completion on graphs. *arXiv preprint arXiv:1408.1717*, 2014.

[6] Y. Liu, L. Zhu, P. Szekely, A. Galstyan, and D. Koutra. Coupled clustering of time series and networks. In *2019 SIAM ICDM*, pages 531–539. SIAM, 2019.

[7] M. McNeil and P. Bogdanov. Multi-dictionary tensor decomposition. In *2023 IEEE International Conference on Data Mining*, Online, 2023.

[8] M. McNeil, L. Zhang, and P. Bogdanov. Temporal graph signal decomposition. In *ACM International Conference on Knowledge Discovery and Data Mining*, Online, 2021.

[9] K. Qiu, X. Mao, X. Shen, X. Wang, T. Li, and Y. Gu. Time-varying graph signal reconstruction. *IEEE J. of Selected Topics in Signal Processing*, 11(6):870–883, 2017.

[10] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[11] Fang Y., B. Huang, and J. Wu. 2d sparse signal recovery via 2d orthogonal matching pursuit.

[12] Y. Yankelevsky and M. Elad. Finding gems: Multi-scale dictionaries for high-dimensional graph signals. *IEEE Transactions on Signal Processing*, 67(7):1889–1901, 2019.

8 Appendix

Derivation of the Gradient Update Rule for Tensor Decomposition

M. McNeil and P. Bogdanov [7] propose tensor representation through a CPD-like dictionary-based decomposition:

$$\mathcal{X} = \sum_{n=1}^{k} \Phi_1 y_{n,1} \otimes \Phi_2 y_{n,2} \otimes \Phi_3 y_{n,3} = [[\Phi_1 Y_1, \Phi_2 Y_2, \Phi_3 Y_3]], \tag{1}$$

where Φ_i represent a model-specific dictionaries and Y_i are sparse encoding matrices of the input tensor X. The objective function used for multi-dictionary-based decomposition [7] employs an ADMM approach:

$$\min_{Y_1,Y_2,Y_3} \frac{1}{2} \left\| \Omega \boxdot (\mathcal{X} - [\Phi_1 Y_1, \Phi_2 Y_2, \Phi_3 Y_3]) \right\|_F^2 + \sum_{i=1}^{3} \lambda_i \|Y_i\|_1, \tag{2}$$

where $\boxdot$ represents the element-wise product. Iterative updates to Y_i are defined as:

$$\min_{Y_i} \frac{1}{2} \left\| D_i^T - \Phi_i Y_i (B \odot A)^T \right\|_F^2 + \frac{\rho_i}{2} \left\| Y_i - Z_i + \frac{\Gamma_i}{\rho_i} \right\|_F^2, \tag{3}$$

where $\odot$ is the Khatri-Rao product. For a gradient descent-based approach, the gradient with respect to Y_i is derived as:

$$\nabla_{Y_i} = \lambda_i \cdot sign(Y_i) - \Phi_i^T \left[D_i^T - \Phi_i Y_i (A \odot B) \right] (A \odot B)^T, \tag{4}$$

where D is an approximation tensor of X and A and B are shorthand matrices for simplicity, such that $A = \Phi_j Y_j$ and $B = \Phi_l Y_l$ and $j < l$. Thus, each iterative update to Y_i can be calculated by:

$$Y_i^t = Y_i^{t-1} - \eta(\lambda_i \cdot sign(Y_i) - \Phi_i^T \left[D_i^T - \Phi_i Y_i (A \odot B) \right] (A \odot B)^T), \tag{5}$$

The term $D_i^T - \Phi_i Y_i (A \odot B)$ represents the residual from the transposed mode-i unfolding of the approximation tensor D and its reconstruction obtained by multiplying the dictionary matrix Φ_i, the sparse encoding matrix Y_i, and the Khatri-Rao product of matrices A and B. Variable t denotes the iteration number in the updating process performed on sparse encoding matrix Y_i. The learning rate, denoted by η, is a hyperparameter that controls the step size of the gradient descent update.