antGLasso: An Efficient Tensor Graphical Lasso Algorithm

Anonymous Author(s)
Affiliation
Address
email

Abstract

The class of bigraphical lasso algorithms (and, more broadly, 'tensor'-graphical lasso algorithms) has been used to estimate dependency structures within matrix and tensor data. However, all current methods to do so take prohibitively long on modestly sized datasets. We present a novel tensor-graphical lasso algorithm that directly estimates the dependency structure, unlike its iterative predecessors. This provides a speedup of multiple orders of magnitude, allowing this class of algorithms to be used on large, real-world datasets.

1 Introduction

We often make independence assumptions to simplify the modelling of complex data. The strength of an independence assumption exists on a spectrum. One could assume complete independence, independence except on a sparse subset, or independence of samples but not features, among others.

For conditional dependencies, there is a natural way to model them in Gaussian data. Two datapoints x, y are *conditionally independent* (with respect to a dataset $\mathcal{D}$) if knowing one provides no information about the other that is not already contained in the rest of the dataset: $\mathbb{P}(x|y, \mathcal{D}) = \mathbb{P}(x|\mathcal{D})$. For normally distributed data, conditional dependencies are encoded in the inverse of the covariance matrix (the 'precision' matrix). Two datapoints are conditionally dependent on each other if and only if their corresponding element in the precision matrix is zero. If our dataset $\mathbf{d}$ were in the form of a vector, we could then model it as $\mathbf{d} \sim \mathcal{N}(\mathbf{0}, \Psi^{-1})$ for precision matrix Ψ.

However, datasets are often not simple vectors - for example, single-cell RNA sequencing data (scRNA-seq) comes in the form of a matrix of gene expression counts whose rows are cells and columns are genes. Video data naturally requires a third-order tensor of pixels to represent it - rows, columns, and frames. To address this, we 'vectorize' matrices by stacking the columns vertically. Vectorization of tensors proceeds in much the same way.

Our matrix-variate dataset $\mathcal{D}$ can then be represented as $\mathrm{vec}[\mathcal{D}] \sim \mathcal{N}(\mathbf{0}, \Psi^{-1})$. Suppose our matrix were $n \times p$ dimensional - then $\mathrm{vec}[\mathcal{D}]$ has np elements, and its precision matrix has n^2p^2 elements. This poses a problem: the estimation of our precision matrix requires substantially more parameters than we have in our dataset. Since the columns of the matrix have interdependencies and the rows of the matrix have interdependencies, to reduce space we can compute the interdependencies of the elements of the matrix via some deterministic function of the row-wise and column-wise dependencies. That is, for some function ζ, $\mathrm{vec}[\mathbf{D}] \sim \mathcal{N}(\mathbf{0}, \zeta(\Psi_{\mathrm{row}}, \Psi_{\mathrm{col}})^{-1})$.

2 Background

The Kronecker sum bigraphical lasso (BiGLasso) model was first considered by Kalaitzis et al. [7]. BiGLasso is the multi-axis analog to graphical lasso methods [3, 1], which are used to estimate

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

covariance matrices of data of a multivariate Gaussian distribution. The Kronecker sum of two matrices, $\mathbf{A} \oplus \mathbf{B}$, can be expressed in terms of Kronecker products: $\mathbf{A} \otimes \mathbf{I} + \mathbf{I} \otimes \mathbf{B}$. When the matrices $\mathbf{A}, \mathbf{B}$ are adjacency matrices of graphs, the Kronecker sum has the interpretation as the Cartesian product of those graphs[16]. This sum is one choice of function ζ to combine the per-axis precision matrices into the precision matrix of the vectorized dataset, $\mathrm{vec}[\mathbf{D}] \sim \mathcal{N}(\mathbf{0}, (\Psi_{\mathrm{row}} \oplus \Psi_{\mathrm{col}})^{-1})$.

Other choices for ζ have been considered, such as using the Kronecker product[18, 23], or the square of the Kronecker sum[20]. Each method has its strengths; the benefits of a Kronecker sum structure are its interpretability as a graph product, stronger sparsity, and its allowance of inter-task transfer[7].

The original BiGLasso model was very slow to converge to a solution, in large part due to its non-optimal space complexity of $O(n^2p^2)$. This prohibited its use on large datasets (measuring in a couple hundred samples and/or features). Numerous modifications have been made to the algorithm to improve its speed and achieve an optimal space complexity of $O(n^2+p^2)$, such as scBiGLasso[10], TeraLasso[4], and EiGLasso[21]. Of these, TeraLasso is notable in that it generalizes to an arbitrary amount of axes, i.e. $\zeta(\mathbf{\Psi}_1, ..., \mathbf{\Psi}_k) = \mathbf{\Psi}_1 \oplus ... \oplus \mathbf{\Psi}_k$. However, all of these algorithms are still iterative. The convergence time can vary greatly depending on the precise values of the data, and in general they struggle to handle datasets with a couple thousand samples and/or features. Our proposed variant, Heuristic antGLasso (**an**alytic **t**ensor-**g**raphical **lasso**), does not suffer from these issues, preserves optimal memory complexity, and like TeraLasso can work on datasets with an arbitrary amount of axes.

All algorithms listed rely on a normality assumption. One could manually transform non-normal data to a Gaussian, if they knew the right transform to use. An alternative method is to use the Nonparanormal Skeptic[11], an algorithm which directly estimates the Gram matrices[1] of data that has been transformed to a Gaussian by some unspecified function[10]. Since all previous BiGLasso algorithms mentioned can be framed in terms of accepting these Gram matrices as input, they can be made to circumvent the normality assumption.

3 Methodology and Results

In the following we introduce the main theorems and results of our study. For our implementation, we used Python Version 3.9.12, Numpy Version 1.22.2[5], Scipy Version 1.7.3[19], Sklearn Version 1.0.2[14], Matplotlib Version 3.5.1[6], and Pandas Version 1.4.2[17, 12]. All tests were run on a MacBook Pro (13-inch, M1, 2020) with 8GB of RAM, using Jupyter Notebooks[15, 8].

4 Theorems

The general structure of the algorithm is as follows: we first directly estimate the eigenvectors of the solution. We then use the eigenvectors to diagonalize the covariance matrix of the tensor-variate distribution, such that we can directly estimate the eigenvalues from the variances of the data. We follow the notation of Kolda and Bader [9] for tensor operations, and Greenewald, Zhou, and Hero [4] for BiGLasso variables. $\mathbf{K}_b^a$ is shorthand for the matrix with a on the diagonals and b elsewhere. Our proofs of the following theorems are given in the supplementary material, and rely on a lemma by Dahl et al. [2].

Theorem (Eigenvectors of Precision Matrices)**.** *$\mathbf{V}_\ell$, the eigenvectors of $\mathbf{\Psi}_\ell$, are the eigenvectors of $\mathbf{S}_\ell \circ \mathbf{K}_{m_\ell}^{2m_\ell-1}$, where $\mathbf{K}_b^a$ is the matrix with diagonal elements a and off-diagonal elements b.*

Theorem (Eigenvalues of Precision Matrices)**.** *Let $\mathbf{a}$ be the vector of length $\prod_\ell^K d_\ell$ whose $(\sum_\ell i_\ell \prod_{j=1}^{\ell-1} d_j)$th element is $\frac{1}{\mathrm{var}[[\mathcal{Y} \times_1 \mathbf{V}_1^T \times_2 ... \times_K \mathbf{V}_K^T]_{i_1 ... i_K}]}$. Then there is a system of linear equations relating $\mathbf{a}$ to the eigenvalues of $(\mathbf{\Psi}_1, ..., \mathbf{\Psi}_K)$.*

Unfortunately, the calculation of the eigenvalues requires the whole dataset $\mathcal{Y}$, whereas the use of the Nonparanormal Skeptic requires that the algorithm be parametrizable in terms of the Gram matrices. We can circumvent this by introducing a heuristic, $\mathrm{var}\left[[\mathcal{Y} \times_1 \mathbf{V}_1^T \times_2 ... \times_K \mathbf{V}_K^T]_{i_1,...,i_K}\right] \approx \Delta_{i_\ell} \mathbf{V}_\ell^T \mathbf{S}_\ell \mathbf{V}_\ell \Delta_{i_\ell}^T$, where Δ_{i_ℓ} is a row vector of zeros except for a one at position i_ℓ. We call this variant of the algorithm 'Heuristic antGLasso', in contrast to the previous 'Vanilla antGLasso'; further details are available in the supplementary material.

[1] Defined in the supplementary material.

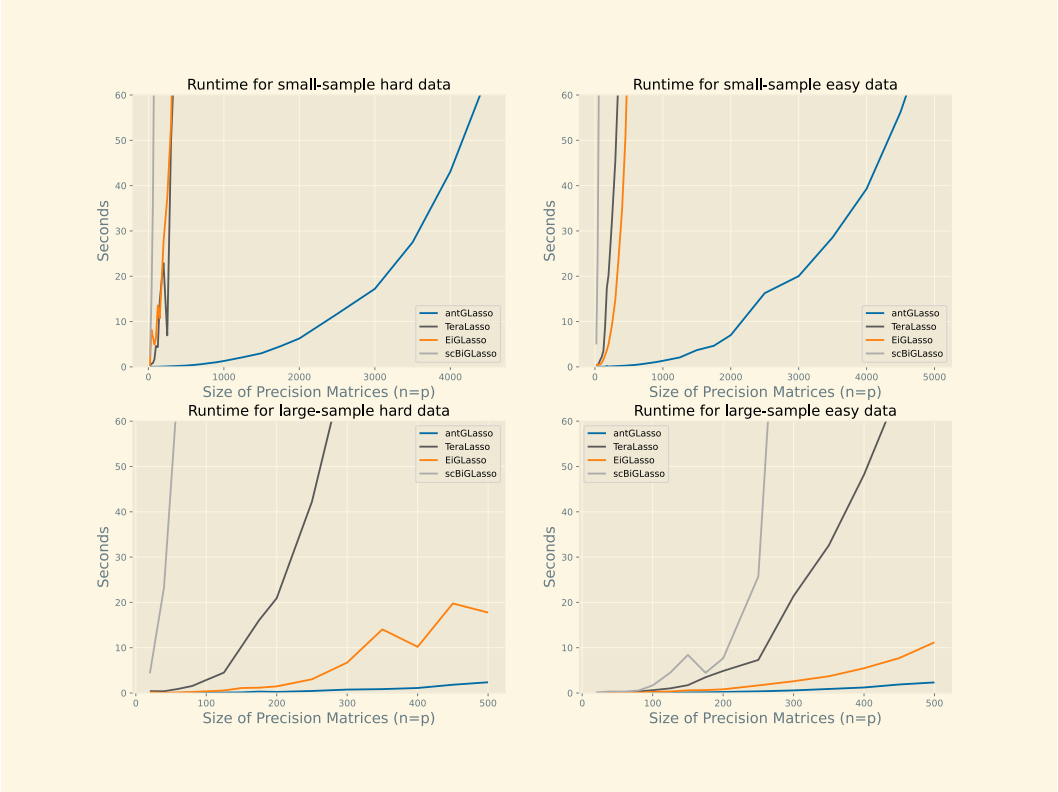

Figure 1: A comparison of runtimes of four BiGLasso algorithms on simulated data. We consider small samples ($s = 1$) and large samples ($s = 100$), as well as data drawn from distributions that tend to cause algorithms to converge slowly ('hard') and quickly ('easy').

5 Results on Simulated Data

We first tested our algorithm on simulated data. The precision matrices were drawn from an inverse Wishart distribution. The iterative algorithms tended to converge slowly when the degrees of freedom parameter of the distribution was small, and quickly when it was large. Additionally, they converged much slower on small sample data (Figure 1). We can see that antGLasso is by far the fastest algorithm, giving up to two orders of magnitude improvement over the next fastest algorithm (EiGLasso).

Not pictured are the results for tensor data, which are in the supplementary material. For three-axis tensors, antGLasso is still markedly faster than TeraLasso in the small sample case - we did not have enough RAM to test the large sample case. For four-axis and larger, the difference between the algorithms becomes negligible - this is because the major runtime constraint becomes the calculation of the Gram matrices, whose complexity grows exponentially with the amount of axes and thus dominates.

Vanilla antGLasso does not perform as well as other algorithms (Figure 2), although this gap drops as the number of samples increase. Interestingly, Heuristic antGLasso performs comparably to the others while preserving the speed of Vanilla antGLasso. We believe this is because Vanilla antGLasso directly estimates the variances of the data, which will be inaccurate for small samples. The heuristic also estimates variances, but by using info on the variance across other axes to in essence artificially increase the sample count. The gap between the two variants drops as the number of samples increases.

6 Temporal Recovery

We have shown that on simulated data Heuristic antGLasso performs comparably but much quicker than existing algorithms. This enables the application of BiGLasso methods to datasets in the size of

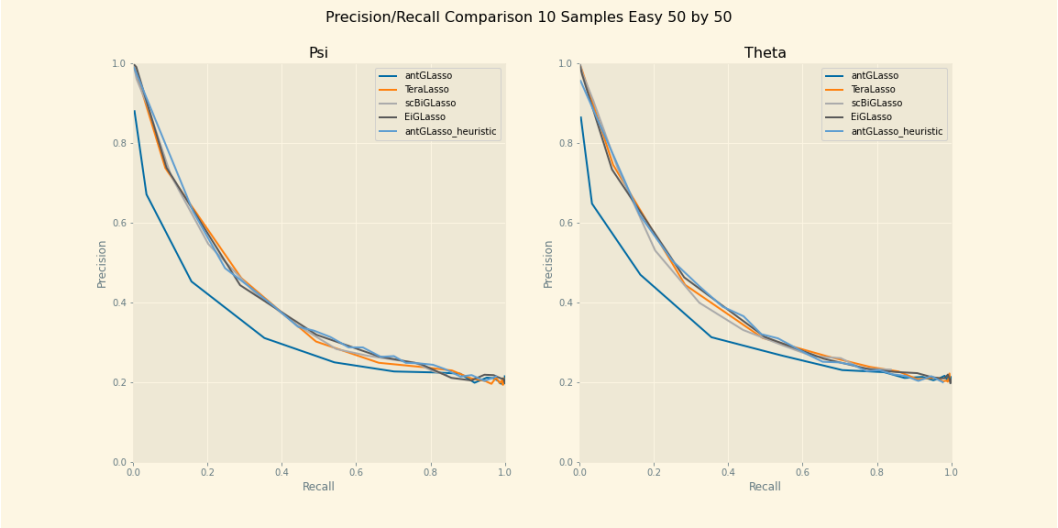

Figure 2: Precision-recall curves for two-dimensional simulated data from a distribution that converged quickly ('easy'), for 50x50 input matrices.

Figure 3: Recovery of a video of a spinning rubber duck. (Left) A frame from the original video, (Middle) a frame from the shuffled video, (Right) frame from the reconstructed video. Not represented in the image is the recovery of the frames of the video, which was perfect.

thousands. For real world experiments, we can consider the case of video data. Given the graph of conditional dependencies, we might expect to be able to order the frames, as a frame should only be conditionally dependant on its nearest neighbors.

For example, consider the video of a rubber duck from the COIL-20 dataset[13]. If we scramble the rows, frames, and columns of the video, we can recover the original video from the outputs of antGLasso, as seen in Figure 3. The reconstruction is perfect, except for the duck being cut in half. This is due to the difficulty in figuring out which row to start on.

The strong results for video reconstruction are encouraging. However, when we modify the video by duplicating frames and adding a bit of noise our reconstruction performs much worse. Already by duplicating each frame 5 times we dip below 10% reconstruction accuracy. This is arguably because duplicating frames interferes with the conditional dependency graph - each frame has more similar frames in the rest of the dataset, which then get conditioned out, reducing informativeness. This suggests that temporal recovery can only be done in highly structured cases with few redundancies. We verified that this behavior is seen in EiGLasso as well, and hence is a systemic problem of BiGLasso methods rather than a quirk of antGLasso.

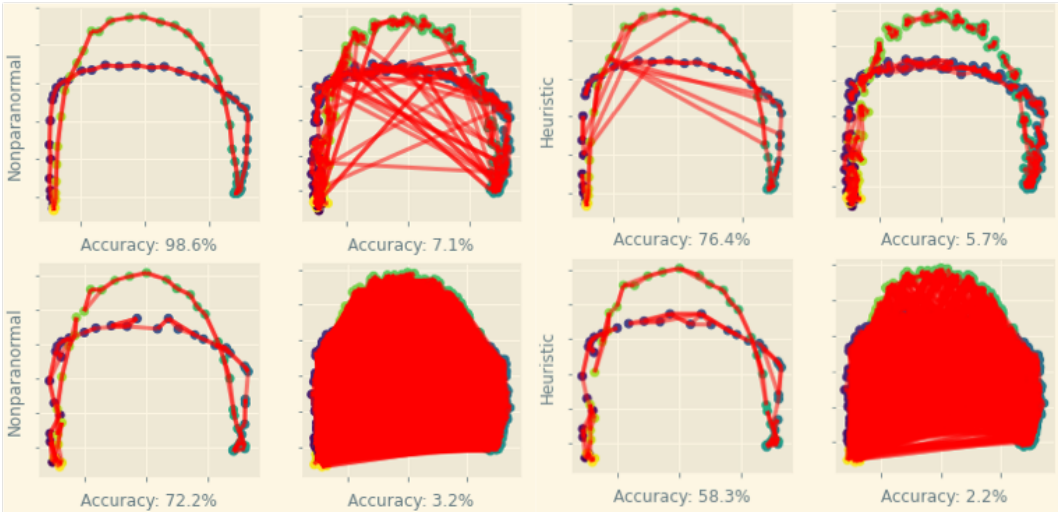

Figure 4: A comparison of temporal recovery on the COIL duck video visualized with the first two principal components. Unlike elsewhere in the paper, we do not treat the video as a three dimensional tensor, but rather we combine the two pixel dimensions so that we can make use of PCA. Lines linking nodes mean that one of the nodes has the other as one of its top two strongest connections according to the precision matrix. The left half of the graph was constructed with antGLasso+Nonparanormal Skeptic, and the right half used Heuristic antGLasso. In each half, the top-left graph shows performance on the unaltered COIL video. The bottom-left shows how performance degrades when we add noise, and the top-right shows what happens when we duplicate each frame 5 times. The bottom-right has each frame duplicated 25 times.

In Figure 4, we can see how choice of algorithm as well as duplicate frames and noise affect temporal recovery. When frames are not duplicated, using the Nonparanormal Skeptic performs the best. However, once frames are duplicated it seems that using Heurstic antGLasso is better. Even though the heuristic results in a lower accuracy, visually we can see that when it is wrong we can still trace a sensible progression along the graph. Once the Nonparanormal Skeptic starts going wrong, it starts connecting completely disparate parts of the graph rather than failing gracefully.

7 Clustering

To test antGLasso's performance on clustering realistic data, we generated a dataset using Splatter[22]. Splatter generates scRNA-seq data, which does not follow a normal distribution - hence using the Nonparanormal Skeptic is justified.

There are three different interpretations of the output graphs that we can use for clustering - we can use the outputs directly (the precision matrices), we can invert the outputs (covariance matrices), or we can consider only the negative values of our precision matrices. Inversion is relatively cheap as our algorithm estimates the eigenvalues and eigenvectors separately. Values in the precision matrix of a Gaussian Graphical Model can be interpreted as the 'negative stiffness' of a spring connecting two elements together, motivating the consideration of the negative values of our outputs.

In addition to the three interpretations of our outputs, we have three different models to use: Vanilla antGLasso, Heuristic antGLasso, and Heuristic antGLasso with the Nonparanormal Skeptic. The results can be seen in Figure 5. Heuristic antGLasso is the best to use, whereas the Nonparanormal Skeptic does very poorly. We also tested EiGLasso's performance on the same dataset, and found similar results.

8 Regularization

The algorithm is based on an analytic solution to the maximum of the unregularized likelihood function. We do not have an analytic solution for the L1-penalized likelihood function. Instead of

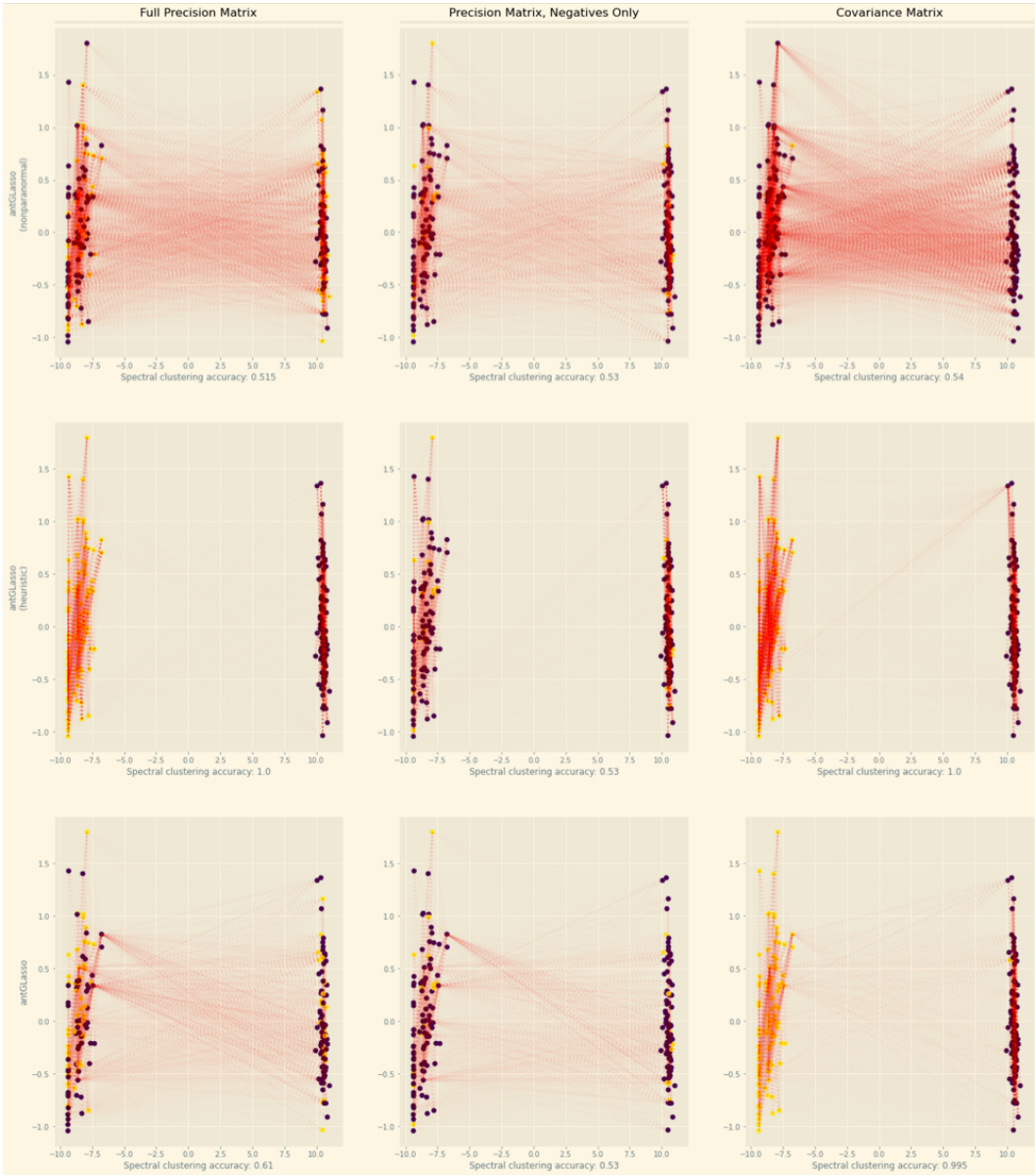

Figure 5: A comparison of different versions of the algorithm (rows) and ways to interpret the results (columns). Only Heuristic antGLasso clusters robustly in all interpretations, as evidenced by the lack of cross-connections. We report the accuracy of performing spectral clustering on these matrices as an numerical measure of clustering acumen.

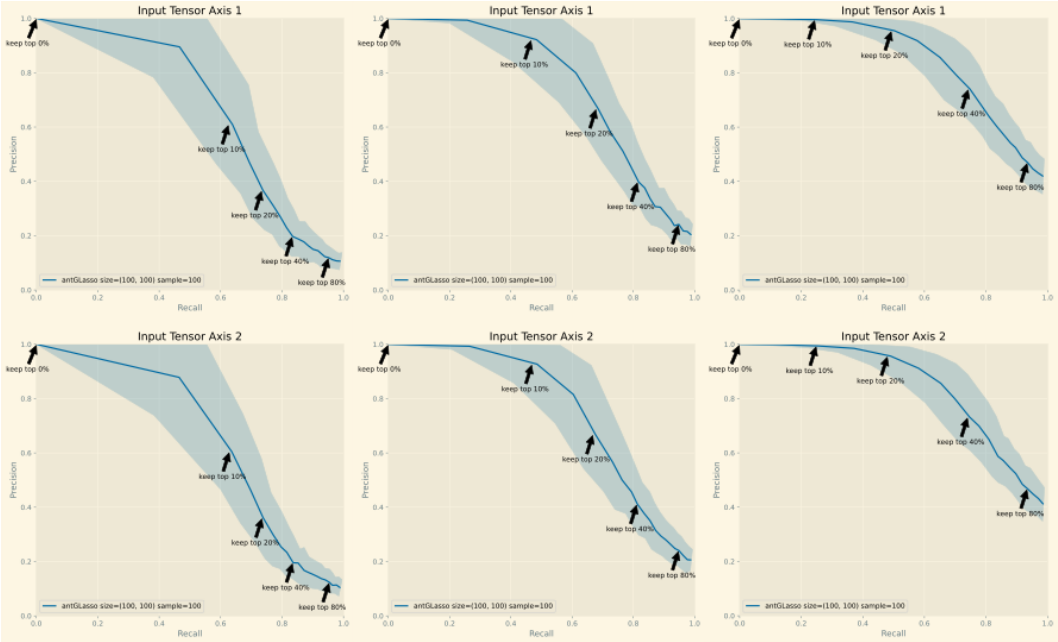

Figure 6: Precision recall curves with hyperparameter values labeled. (Left column) Simulated data with 10% true edges. (Middle column) Simulated data with 20% true edges. (Right column) Simulated data with 40% true edges.

regularizing during the calculation of precision matrices, we first find the unregularized solution, and then apply a regularization step. By regularizing the solution at the end, rather than simultaneously, it allows us to efficiently examine the solution across many different penalty strengths. However, it does not necessarily converge to the solution to the L1-penalized likelihood function, unlike previous algorithms. As seen in Figure 2, Heuristic antGLasso still achieves state-of-the-art recovery in spite of this.

If we were to apply Lasso to our unregularized solution, in a similar manner to how BiGLasso and scBiGLasso apply Lasso at the end of each iteration, it would be equivalent to just thresholding. This is convenient: not only is thresholding a very fast operation, but it can also be framed in terms of the percentage of edges to keep. As seen in Figure 6, if we simulate data to have a given sparsity level, and then use the known ground truth sparsity as the hyperparameter, we achieve a balance between precision and recall. Thus, if we have an idea of how densely connected our data should be and the extent to which we wish to prioritize precision over recall, we will know in advance a good regularization strength.

Despite the benefits of this sparsity-based regularizer, we often chose to use problem-specific regularization regimes. For temporal recovery, a simple technique is to keep only the two highest values in each row (for the immediate past and immediate future). For clustering, it can make sense not to regularize, as the additional edges may contain useful information for the clusterer despite not actually being present in the real data.

9 Discussion and Conclusion

We have created a new BiGLasso algorithm, antGLasso, which is much faster than current competitors, while preserving (in the heuristic case) their accuracy. It can be applied to tensors of arbitrary dimensions, like TeraLasso, and performs well on simulated data as well as highly-structured real-world data. It falters when applied to more complicated data, which is a major limitation. However, this is a systemic problem with BiGLasso algorithms rather than our implementation specifically.

For future work, we would like to understand better when the Nonparanormal Skeptic should be used. We know that the Nonparanormal Skeptic can be useful for clustering, as Li et al. [10] used it to

find interpretable results on real scRNA-seq data. It seems that the Nonparanormal Skeptic performs poorly when the data is multimodal - in the case of duplicating frames in Figure 4, the duplication has the effect of creating easily separable clusters of duplicates, and we believe this is what caused the Nonparanormal Skeptic variant to perform so poorly just as it did in the clustering tests of Figure 5. Further investigation is needed.

We would also like to investigate how to circumvent the known issues with applying this method to less structured datasets, such as scRNA data, for temporal recovery and other tasks. This may involve changing aspects of the model to better fit the idiosyncrasies of specific types of datasets.

Finally, we believe that, despite our model's speed, it is still important for further improvements to be made. For datasets of size in the 1000s, the runtime is not significant. However, by extrapolating the (cubic) runtime curves to the largest dataset whose outputs could fit on our computer's RAM (8GB), we can estimate that antGLasso would take 2 days to run. We hope that the algorithm can reach a point where all problems that fit on a personal computer can run over a coffee break.

Improving speed would likely require a fundamentally different approach, as the majority of the runtime is taken up by a single eigendecomposition (per axis). We considered using other decompositions, but could not find any that used orthonormal matrices, had a sparse core, and behaved nicely under the block-wise trace of the inverse of the Kronecker product of cores. If such a decomposition does exist, and is faster than eigendecomposition, then further speed gains can be made using a structurally similar algorithm to antGLasso.

Despite this, the constraint limiting BiGLasso-style algorithms' applicability to even larger datasets is likely to be memory. Unfortunately, there is not much room for improving memory usage - modifications could be made to remove some intermediate products, but even in the ideal case we would be reducing memory usage by a small constant factor, since Heuristic antGLasso is asymptotically memory-optimal.

References

[1] Onureena Banerjee, Laurent El Ghaoui, and Alexandre d'Aspremont. "Model Selection Through Sparse Maximum Likelihood Estimation for Multivariate Gaussian or Binary Data". In: *Journal of Machine Learning Research* 9.15 (2008), pp. 485–516. URL: http://jmlr.org/papers/v9/banerjee08a.html.

[2] Andy Dahl et al. *Network inference in matrix-variate Gaussian models with non-independent noise*. 2013. DOI: 10.48550/ARXIV.1312.1622. URL: https://arxiv.org/abs/1312.1622.

[3] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *Sparse inverse covariance estimation with the lasso*. 2007. DOI: 10.48550/ARXIV.0708.3517. URL: https://arxiv.org/abs/0708.3517.

[4] Kristjan Greenewald, Shuheng Zhou, and Alfred Hero. *Tensor Graphical Lasso (TeraLasso)*. 2017. DOI: 10.48550/ARXIV.1705.03983. URL: https://arxiv.org/abs/1705.03983.

[5] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[6] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[7] Alfredo Kalaitzis et al. "The Bigraphical Lasso". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1229–1237. URL: https://proceedings.mlr.press/v28/kalaitzis13.html.

[8] Thomas Kluyver et al. "Jupyter Notebooks - a publishing format for reproducible computational workflows". In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. Ed. by Fernando Loizides and Birgit Scmidt. Netherlands: IOS Press, 2016, pp. 87–90. URL: https://eprints.soton.ac.uk/403913/.

[9] Tamara G. Kolda and Brett W. Bader. "Tensor Decompositions and Applications". In: *SIAM Review* 51.3 (Sept. 2009), pp. 455–500. DOI: 10.1137/07070111X.

[10] Sijia Li et al. "Scalable Bigraphical Lasso: Two-way Sparse Network Inference for Count Data". In: (Mar. 2022).

[11] Han Liu et al. *The Nonparanormal SKEPTIC*. 2012. DOI: 10.48550/ARXIV.1206.6488. URL: https://arxiv.org/abs/1206.6488.

[12] Wes McKinney. "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[13] Sameer A. Nene, Shree K. Nayar, and Hiroshi Murase. *Columbia Object Image Library (COIL-20*. Tech. rep. 1996.

[14] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[15] Fernando Pérez and Brian E. Granger. "IPython: a System for Interactive Scientific Computing". In: *Computing in Science and Engineering* 9.3 (May 2007), pp. 21–29. ISSN: 1521-9615. DOI: 10.1109/MCSE.2007.53. URL: https://ipython.org.

[16] Gert Sabidussi. "Graph Multiplication." In: *Mathematische Zeitschrift* 72 (1959/60), pp. 446–457. URL: http://eudml.org/doc/183624.

[17] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

[18] Theodoros Tsiligkaridis and Alfred O. Hero. "Covariance Estimation in High Dimensions Via Kronecker Product Expansions". In: *IEEE Transactions on Signal Processing* 61.21 (Nov. 2013), pp. 5347–5360. DOI: 10.1109/tsp.2013.2279355. URL: https://doi.org/10.1109%2Ftsp.2013.2279355.

[19] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[20] Yu Wang, Byoungwook Jang, and Alfred Hero. "The Sylvester Graphical Lasso (SyGlasso)". In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. Ed. by Silvia Chiappa and Roberto Calandra. Vol. 108. Proceedings of Machine Learning Research. PMLR, 26–28 Aug 2020, pp. 1943–1953. URL: https://proceedings.mlr.press/v108/wang20d.html.

[21] Jun Ho Yoon and Seyoung Kim. "EiGLasso: Scalable Estimation of Cartesian Product of Sparse Inverse Covariance Matrices". In: *Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI)*. Ed. by Jonas Peters and David Sontag. Vol. 124. Proceedings of Machine Learning Research. PMLR, Mar. 2020, pp. 1248–1257. URL: https://proceedings.mlr.press/v124/ho-yoon20a.html.

[22] L. Zappia, B. Phipson, and A. Oshlack. "Splatter: simulation of single-cell RNA sequencing data". In: *Genome Biol* 18.1 (Sept. 2017), p. 174.

[23] Shuheng Zhou. "Gemini: Graph estimation with matrix variate normal instances". In: *The Annals of Statistics* 42.2 (Apr. 2014). DOI: 10.1214/13-aos1187. URL: https://doi.org/10.1214%2F13-aos1187.

A Supplementary Material

A.1 Hyperparameters

For K-axis tensor data, there are $K + 1$ hyperparameters. The first K are regularization parameters for each axis - as we will see in Section A.6, they can be interpreted as a percent of values to keep, and are thus naturally interpretable. The remaining hyperparameter, b controls the accuracy of the Monte Carlo speedup discussed in Section A.8. In our implementation b is the amount of terms to average over, although one could replace it with a hyperparameter controlling a convergence threshold.

In Figure 7 we can see that when our value of the regularization hyperparameter matches the true value of the edges (on simulated data) we achieve an even balance of precision and recall.

A.2 Simulating Data and Metrics

All experiments on simulated data use data simulated from the same process. We first simulate sparse precision matrices, whose Kronecker sum is used as the parameter for a zero-mean tensor-variate

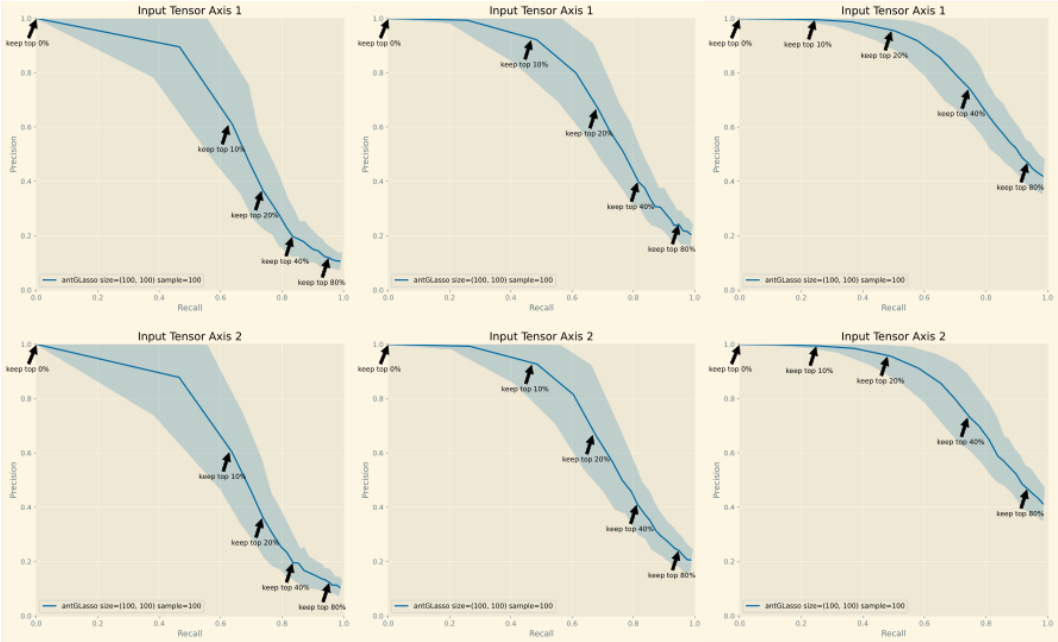

Figure 7: (Left column) Precision recall curves for simulated data with 10% sparsity. (Middle) 20% sparsity. (Right) 40% sparsity. This experiment was performed on two-axis data, with the rows of the figure representing the axes. Arrows indicate value of the regularization hyperparameter.

normal distribution using Corollary 1. For calculating precision and recall, we consider when edges from the true conditional dependency graph match with the estimated dependency graph. To construct our precision/recall plots we set the b hyperparameter to 1000 and varied our sparsity parameters from 0 to 1 to obtain somewhat smooth curves. $b = 1000$ was chosen empirically because in tests values above it did not change the quality of the estimation. In fact a lower value such as $b = 100$ could have been chosen, but the runtime difference is negligible so we selected the higher value.

To simulate sparse precision matrices, we first draw precision matrices from an inverse Wishart distribution. These are not sparse. To sparsify them, we construct a mask in the following manner: construct a vector of i.i.d. Bernoulli variables $\mathbf{x}$, and construct the matrix $\mathbf{M} = c\mathbf{x}\mathbf{x}^T$ where $c \in (0, 1)$ (we chose 0.9). Let $\mathbf{N}$ the diagonals of $\mathbf{M}$ to be 1. By changing the parameter of the Bernoulli distribution we can control the expected sparsity of $\mathbf{N}$. By construction we have guaranteed that it is positive definite,[2] and hence the Hadamard product of it with our precision matrix will also be positive definite.

A.3 Pseudocode

We give the pseudocode for antGLasso in Algorithm 1 (with the Monte Carlo speedup described in Section A.8).

A.4 Proofs

Let s be the number of samples of tensors $(\mathcal{Y}_1, ...\mathcal{Y}_s)$, each with shape $(d_1, ..., d_K)$. Each of these tensors is assumed to be i.i.d from our tensor-variate Kronecker sum normal distribution for which we want to estimate the per-axis precision matrices. It is not uncommon to have $n = 1$. It will be helpful to define $m_\ell = \frac{\prod_{i=1}^{K} d_i}{d_\ell}$. Let $\mathbf{\Psi}_\ell$ be the precision matrix of the ℓth axis, and $\mathbf{S}_\ell$ be the Gram matrix for the ℓth axis. These can either be obtained via the Nonparanormal Skeptic, or by the

[2] $\mathbf{N} = \mathbf{M} + (1 - c)\mathbf{I}$, and hence the eigenvalues of $\mathbf{N}$ must all be $(1 - c)$ larger than their corresponding eigenvalue of $\mathbf{M}$. As $\mathbf{M}$ is positive semidefinite, $\mathbf{N}$ is positive definite.

Algorithm 1 Analytic Tensor Graphical Lasso (antGLasso)

Input: $\{\mathcal{Y}^{(n)}_{d_1\times\ldots\times d_K}\}, \{\beta_\ell\}, b$
Output: $\{\mathbf{\Psi}_\ell\}$
  for $1 \le \ell \le n$
    $\mathbf{S}_\ell \leftarrow \frac{1}{nm_\ell}\sum_i^n \mathbf{Y}^{(i)}_{(\ell)}\mathbf{Y}^{(i)T}_{(\ell)}$
    $\mathbf{V}_\ell \leftarrow \text{eigenvectors}[\mathbf{S}_\ell \circ \mathbf{K}^{2m_\ell-1}_{m_\ell}]$
  end for
  for $1 \le p \le m$
    $\mathcal{X}^{(p)} \leftarrow \mathcal{Y}^{(p)} \times_1 \mathbf{V}_1 \times_2 \ldots \times_K \mathbf{V}_K$
  end for
  for $\vec{i} \in [1,\ldots,d_1]\times\ldots\times[1,\ldots,d_K]$
    $\mathbf{a}_{\sum_\ell i_\ell d_{1:(\ell-1)}} \leftarrow \frac{n}{\sum_p^n (x^{(p)}_{i_1,\ldots,i_K})^2}$
  end for
  $\mathbf{\Lambda} \leftarrow \mathbf{0}$
  Construct $\tilde{\mathbf{B}}^{-1}$ as described in Section A.8.
  for b iterations of different tuples $\vec{i}$
    Let $\mathbf{P}^{\vec{i}}_1, \mathbf{P}^{\vec{i}}_2$ be the matrices representing the permutations that capture the process described in Section A.8.
    If first iteration, set each element in $\lambda^{1:(K-1)}_{\vec{i}}$ to 1
    $\mathbf{\Lambda} \leftarrow \mathbf{\Lambda} + \mathbf{P}^{\vec{i}}_1\tilde{\mathbf{B}}^{-1}\mathbf{P}^{\vec{i}}_2\begin{bmatrix}\mathbf{a}\\ \lambda^{1:(K-1)}_{\vec{i}}\end{bmatrix}$
  end for
  $\mathbf{\Lambda} \leftarrow \mathbf{\Lambda}/b$
  for $1 \le \ell \le K$
    $\mathbf{\Psi}_\ell \leftarrow \mathbf{V}_\ell\mathbf{\Lambda}_\ell\mathbf{U}^T_\ell$
    for ith row $\mathbf{r}$ in $\mathbf{\Psi}_\ell$
      $\mathbf{r}_{\backslash i} \leftarrow \text{sign}(\mathbf{r}_{\backslash i}) \circ \min(0, |\mathbf{r}_{\backslash i}| - \frac{\beta_\ell}{2})$
    end for
  end for

formula $\frac{1}{nm_\ell}\sum_{i=1}^n \mathbf{Y}_{i,(\ell)}\mathbf{Y}^T_{i,(\ell)}$, where $\mathbf{A}_{(\ell)}$ is the 'matricization' of the ℓth axis.[3] Another piece of tensor-specific notation we use is the ℓ-mode product, $\mathcal{Y}\times_\ell \mathbf{M}$, which intuitively is multiplying a tensor by a matrix along its ℓth dimension. When the input is a two-dimensional tensor (a matrix), $\times_1$ and $\times_2$ correspond to left-multiplying by the transpose and right-multiplying, respectively. We often encounter the case $\mathcal{Y}\times_1\mathbf{V}_1\times_2\mathbf{V}_2\times_3\ldots$, which is abbreviated as $\mathcal{Y};\{\mathbf{V}_\ell\}$. For an overview of tensor notation, we direct the reader to the comprehensive report by Kolda and Bader [9]. We use $\circ$ to represent the Hadamard product, and $\text{tr}_n[\mathbf{M}]$ to be the blockwise trace obtained as follows: partition $\mathbf{M}$ into $n\times n$ blocks, and then take the trace of each block. The output is the matrix of these traces.

A.5 Analytic Solution

Theorem 1. *Let $\mathbf{V}_\ell\mathbf{\Lambda}_\ell\mathbf{V}^T_\ell$ be the eigendecomposition of $\mathbf{\Psi}_\ell$. $\mathbf{V}_\ell$ are the eigenvectors of $\mathbf{S}_\ell\circ\mathbf{K}^{2m_\ell-1}_{m_\ell}$.*

Proof. Greenewald, Zhou, and Hero [4] present a maximum likelihood estimator for the tensor variate case: $-\log\left|\bigoplus_{\ell=1}^K\mathbf{\Psi}_\ell\right| + \sum_{\ell=1}^K m_k\mathbf{S}^T_\ell\mathbf{\Psi}_\ell$. Using the exact same argument as Kalaitzis et al. [7], we can derive the fixed point $\mathbf{S}_\ell - \frac{1}{2m_\ell}\mathbf{S}_\ell\circ\mathbf{I} = \frac{1}{m_\ell}\text{tr}_{m_\ell}[(\bigoplus_i\mathbf{\Psi}_i)^{-1}] - \frac{1}{2m_\ell}\text{tr}_{m_\ell}[(\bigoplus_i\mathbf{\Psi}_i)^{-1}]\circ\mathbf{I}$. The order in which we compute $\bigoplus_i\mathbf{\Psi}_i$ matters - we'll assume here that we've transposed the data such that $\bigoplus_i\mathbf{\Psi}_i = \mathbf{\Psi}_\ell\oplus\bigoplus_{i\neq\ell}\mathbf{\Psi}_i$. It is only important that $\mathbf{\Psi}_\ell$ goes first, the rest of the order does not matter. Let $\mathbf{\Psi}_{\backslash\ell} = \bigoplus_{i\neq\ell}\mathbf{\Psi}_i$ so that $\bigoplus_i\mathbf{\Psi}_i = \mathbf{\Psi}_\ell\oplus\mathbf{\Psi}_{\backslash\ell}$

[3] This is similar to vectorization, except instead of stacking all axes into a column vector, we preserve the ℓth axis, reducing our tensor to a matrix instead. Further details available in [9].

$$\mathbf{S}_\ell - \frac{1}{2m_\ell}\mathbf{S}_\ell \circ \mathbf{I} = \frac{1}{m_\ell}\mathrm{tr}_{m_\ell}[(\mathbf{\Psi}_\ell \oplus \mathbf{\Psi}_{\backslash\ell})^{-1}] - \frac{1}{m_\ell}\mathrm{tr}_{m_\ell}[(\mathbf{\Psi}_\ell \oplus \mathbf{\Psi}_{\backslash\ell})^{-1}] \circ \mathbf{I} \tag{1}$$

$$\mathbf{S}_\ell \circ \mathbf{K}_1^{\frac{2m_\ell-1}{2m_\ell}} = \mathrm{tr}_{m_\ell}[(\mathbf{\Psi}_\ell \oplus \mathbf{\Psi}_{\backslash\ell})^{-1}] \circ \mathbf{K}_{\frac{1}{m_\ell}}^{\frac{1}{2m_\ell}} \tag{2}$$

$$\mathbf{S}_\ell \circ \mathbf{K}_{m_\ell}^{2m_\ell-1} = \mathrm{tr}_{m_\ell}[(\mathbf{\Psi}_\ell \oplus \mathbf{\Psi}_{\backslash\ell})^{-1}] \tag{3}$$

$$\mathbf{S}_\ell \circ \mathbf{K}_{m_\ell}^{2m_\ell-1} = \mathrm{tr}_{m_\ell}[(\mathbf{V}_\ell \otimes \mathbf{V}_{\backslash\ell})(\mathbf{\Lambda}_\ell \oplus \mathbf{\Lambda}_{\backslash\ell})^{-1}(\mathbf{V}_\ell^T \otimes \mathbf{V}_{\backslash\ell}^T)] \tag{4}$$

$$\mathbf{S}_\ell \circ \mathbf{K}_{m_\ell}^{2m_\ell-1} = \mathrm{tr}_{m_\ell}[(\mathbf{V}_\ell \otimes \mathbf{I})(\mathbf{\Lambda}_\ell \oplus \mathbf{\Lambda}_{\backslash\ell})^{-1}(\mathbf{V}_\ell^T \otimes \mathbf{I})] \tag{5}$$

$$\mathbf{S}_\ell \circ \mathbf{K}_{m_\ell}^{2m_\ell-1} = \mathbf{V}_\ell \mathrm{tr}_{m_\ell}[(\mathbf{\Lambda}_\ell \oplus \mathbf{\Lambda}_{\backslash\ell})^{-1}]\mathbf{V}_\ell^T \tag{6}$$

The penultimate step uses Proposition 3.1 of Li et al. [10], and the last step requires Lemma 2 of Dahl et al. [2]. As $\mathrm{tr}_{m_\ell}[(\mathbf{\Lambda}_\ell \oplus \mathbf{\Lambda}_{\backslash\ell})^{-1}]$ is diagonal, we can observe that $\mathbf{V}_\ell$ must be the eigenvectors of $\mathbf{S}_\ell \circ \mathbf{K}_{m_\ell}^{2m_\ell-1}$.

□

Lemma 1. *Suppose $\mathcal{Y} \sim \mathcal{N}(0, \bigoplus\{\mathbf{\Psi}_i\}^{-1})$. Then we can diagonalize the precision matrix as follows: $\mathcal{X} = \mathcal{Y} \times_1 \mathbf{V}_1^T \times_2 ... \times_K \mathbf{V}_K^T \sim \mathcal{N}(\mathbf{0}, \bigoplus\{\mathbf{\Lambda}_i\}^{-1})$.*

Proof. We will show that the probability density function of $\mathcal{X}$ is that of a Kronecker sum distribution with the desired parameters. It will rely on the following useful property of ℓ-mode matrix multiplication: $(\mathcal{Y} \times_1 \mathbf{V}_1 \times_2 ... \times_K \mathbf{V}_K)_{(\ell)} = \mathbf{V}_\ell \mathcal{Y}_{(\ell)}(\mathbf{V}_K \otimes ... \otimes \mathbf{V}_{\ell+1} \otimes \mathbf{V}_{\ell-1} \otimes ... \otimes \mathbf{V}_1)^T$.

$$\mathrm{pdf}(\mathcal{X}) = \mathrm{pdf}(\mathcal{Y}) \tag{7}$$

$$= (2\pi)^{\frac{-\prod_i d_i}{2}} \sqrt{\left|\bigoplus_i \mathbf{\Psi}_i\right|} e^{\frac{-1}{2}\sum_\ell^K \mathrm{tr}[\mathbf{\Psi}_\ell \mathcal{Y}_{(\ell)}\mathcal{Y}_{(\ell)}^T]} \tag{8}$$

$$\left|\bigoplus_i \mathbf{\Psi}_i\right| = \left|\bigotimes_i \mathbf{V}_i\right| \left|\bigoplus_i \mathbf{\Lambda}_i\right| \left|\bigotimes_i \mathbf{V}_i^T\right| \tag{9}$$

$$= \left|\bigoplus_i \mathbf{\Lambda}_i\right| \tag{10}$$

$$\mathrm{tr}[\mathbf{\Psi}_\ell \mathcal{Y}_{(\ell)}\mathcal{Y}_{(\ell)}^T] = \mathrm{tr}[\mathbf{\Psi}_\ell (\mathcal{X} \times_1 \mathbf{V}_1 \times_2 ... \times_K \mathbf{V}_K)_{(\ell)}(\mathcal{X} \times_1 \mathbf{V}_1 \times_2 ... \times_K \mathbf{V}_K)_{(\ell)}^T] \tag{11}$$

$$= \mathrm{tr}[\mathbf{\Psi}_\ell \mathbf{V}_\ell \mathcal{X}_{(\ell)}(\bigotimes_{i\neq\ell} \mathbf{V}_i)^T(\bigotimes_{i\neq\ell} \mathbf{V}_i)\mathcal{X}_{(\ell)}^T \mathbf{V}_\ell^T] \tag{12}$$

$$= \mathrm{tr}[\mathbf{V}_\ell^T \mathbf{\Psi}_\ell \mathbf{V}_\ell \mathcal{X}_{(\ell)}\mathcal{X}_{(\ell)}^T] \tag{13}$$

$$= \mathrm{tr}[\mathbf{\Lambda}_\ell \mathcal{X}_{(\ell)}\mathcal{X}_{(\ell)}^T] \tag{14}$$

□

The following corollary provides a convenient way to sample from tensor-variate Kronecker sum distributions without high dimensionality or expensive matrix inverses/decompositions.

Corollary 1. *Suppose you have $\prod_i d_i$ samples of $\mathcal{N}(0,1)$ $\mathbf{z}$, then we have that $\mathrm{vec}^{-1}[(\bigoplus_i \mathbf{\Lambda}_i)^{-1/2}\mathbf{z}] \times_1 \mathbf{V}_1 \times_2 ... \times_K \mathbf{V}_K \sim \mathcal{N}_{KS}(\mathbf{0}, \{\mathbf{V}_i \mathbf{\Lambda}_i \mathbf{V}_i^T\})$*

Proof.

$$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{15}$$

$$(\bigoplus_i \mathbf{\Lambda}_i)^{-1/2}\mathbf{z} \sim \mathcal{N}(\mathbf{0}, (\bigoplus_i \mathbf{\Lambda}_i)^{-1}) \tag{16}$$

$$\text{vec}^{-1}[(\bigoplus_i \mathbf{\Lambda}_i)^{-1/2}\mathbf{z}] \sim \mathcal{N}_{KS}(\mathbf{0}, \{\mathbf{\Lambda}_i\}) \tag{17}$$

$$\text{vec}^{-1}\left[(\bigoplus_i \mathbf{\Lambda}_i)^{-1/2}\mathbf{z}\right] \times_1 \mathbf{V}_1 \times_2 ... \times_K \mathbf{V}_K \sim \mathcal{N}_{KS}(\mathbf{0}, \{\mathbf{V}_i\mathbf{\Lambda}_i\mathbf{V}_i^T\}) \tag{18}$$

□

Lemma 2. $\frac{M}{\sum_n^m (x^{(n)}_{i_1 \dots i_K})^2} \approx \sum_\ell^K \lambda^\ell_{i_\ell}$.

Proof. Let $d_{1:n} = \prod_{\ell=1}^n d_\ell$. Defining $d_{1:0}$ as 1, we can observe that:

$$x_{ij} = \text{vec}[\mathcal{X}]_{\sum_\ell i_\ell d_{1:(\ell-1)}} \tag{19}$$

$$\frac{1}{M}\sum_n^m (x^{(n)}_{i_1 \dots i_K})^2 \approx \text{var}[x_{i_1 \dots i_K}] \tag{20}$$

$$= ((\bigoplus_\ell \mathbf{\Lambda}_\ell)^{-1})_{\sum_\ell i_\ell d_{1:(\ell-1)}, \sum_\ell i_\ell d_{1:(\ell-1)}} \tag{21}$$

$$= \frac{1}{\sum_\ell^K \lambda^\ell_{i_\ell}} \tag{22}$$

$$\frac{M}{\sum_n^m (x^{(n)}_{i_1 \dots i_K})^2} \approx \sum_\ell^K \lambda^\ell_{i_\ell} \tag{23}$$

□

Theorem 2. *We can obtain the eigenvalues from the variances via a linear system.*

Proof. It is easy to see that the approximation of Lemma 2 defines a linear system of the form $\mathbf{a} = \mathbf{B}\mathbf{\Lambda}$, where $\mathbf{a}$ is a vector whose elements are $\frac{M}{\sum_n^m (x^{(n)}_{i_1 \dots i_K})^2}$ and $\mathbf{\Lambda}$ is a vector made from stacking the eigenvalues of each axis. Since the RHS of Lemma 2 is a linear combination of eigenvalues, it is easy to construct a matrix $\mathbf{B}$ relating the two. It is not invertible: the eigenvalues are underdetermined.[4] However, we can select the least squares solution by multiplyling both sides by the pseudo-inverse of $\mathbf{B}$: $\mathbf{B}^\dagger\mathbf{a} \approx \mathbf{\Lambda}$.

The matrix $\mathbf{B}$ will be much larger than necessary - in Section A.8 we will see how to reduce its size. The exact form of $\mathbf{B}$ is easy to understand: the first d_1 columns will represent the eigenvectors of axis 1, the next d_2 columns will represent the eigenvectors of axis 2, and so on. Each row will contain exactly one 1 in the first d_1 columns, exactly one 1 in the next d_2 columns, and so on. The rest is zeros. The matrix contains all possible rows under that restriction. The exact ordering of the rows of the matrix will depend on how you have vectorized your tensor. □

A.6 Regularization

The original BiGLasso performs lasso independently on each row of the precision matrices at each iteration. The natural analogy in this case would be to perform row-wise lasso at the end of antGLasso.

[4]This system is both overdetermined and undertermined, in the sense that it is an inconsistent rectangular matrix (overdetermined) whose rank is not maximal (undetermined). Its rank is always $K - 1$ less than maximal, where K is the number of tensor dimensions of the input.

The function to minimize is, for a row $\hat{r}$, $f(r) = (r - \hat{r})^T (r - \hat{r}) + \beta \, ||r||_1$. If r and $\hat{r}$ were restricted to be nonnegative, then this would be differentiable. Suppose for now that that is the case.

$$\frac{\partial}{\partial r_i} f(r) = \frac{\partial}{\partial r_i} (r - \hat{r})^T (r - \hat{r}) + \beta \, ||r||_1 \tag{24}$$

$$= \frac{\partial}{\partial r_i} \sum_i (r_i - \hat{r_i})^2 + \beta r_i \tag{25}$$

$$= 2(r_i - \hat{r_i}) + \beta \tag{26}$$

$$-\frac{\beta}{2} = r_i - \hat{r_i} \tag{27}$$

$$\hat{r_i} - \frac{\beta}{2} = r_i \tag{28}$$

Since the domain of $\hat{r}_i$ is nonnegative, and the function is monotonic, if $\hat{r}_i - \frac{\beta}{2} < 0$ then the minimum on the domain occurs at $r_i = 0$. We can easily enforce a nonnegative domain by performing our regularization after taking the absolute value of the row. This gives us a regularizer $\mathrm{shrink}(\hat{r}) = \mathrm{sign}(\hat{r}) \circ \min(0, |\hat{r}| - \frac{\beta}{2})$ that is equivalent to performing row-wise Lasso on the output of our algorithm. Note that this allows us to reframe the regularization as a thresholding. In fact, the threshold does not depend on the row, but is rather a global constraint. Thus, instead of using β as an argument, we could give the algorithm a percent of edge connections to keep. When framed this way, our hyperparameters become easily interpretable! We found that when setting the threshold percent to be the true percent of connections in simulated data, the result had roughly equal precision and recall.

A.7 Heuristic

The vanilla variant of antGLasso performs much faster, but noticeably worse than, current state-of-the-art BiGLasso algorithms. This gap closes as the number of samples increases. The part of the algorithm that appears most responsible for this error is the estimation of the variance of $\mathcal{X}$, which is used to calculate the eigenvalues. In small sample cases, such as having only one sample, our variance estimate will necessarily be quite poor. Thus it would be beneficial to devise a heuristic to estimate these variances in a different way. As a side effect, our heuristic will happen to be framed in terms of the empirical covariance matrices, making antGLasso compatible with the Nonparanormal Skeptic method[11] which would allow us to generalize to non-Gaussian data *à la* Li et al. [10].

Our goal is to find the variance of each element of $\mathcal{X}$, or in other words[5] $\underset{\mathcal{X}}{\mathrm{var}}[\mathcal{X}_{i_1 \ldots i_K}]$. The idea of the heuristic is to fix one index, i_n, and consider the remaining indices as random variables $\vec{i}_{\setminus n}$. That is, we wish to find $\underset{\mathcal{X}, \vec{i}_{\setminus n}}{\mathrm{var}} [\mathcal{X}_{i_1 \ldots i_K}]$.

[5] Here we use a subscript under the variance to indicate which variables are to be considered random variables.

$$
\begin{aligned}
\operatorname*{var}_{\mathcal{X},\vec{i}_{\backslash n}}[\mathcal{X}_{i_1\dots i_K}] &= \operatorname*{var}_{\mathcal{Y},\vec{i}_{\backslash n}}[\mathcal{Y};\{\mathbf{V}_\ell^T\}_{i_1\dots i_K}] && (29)\\
&= \operatorname*{var}_{\mathcal{Y},\vec{i}_{\backslash n}}[\mathcal{Y};\{\Delta_{i_\ell}\mathbf{V}_\ell^T\}] && (30)\\
&= \operatorname*{var}_{\mathcal{Y},\vec{i}_{\backslash n}}[\Delta_{i_n}\mathbf{V}_n^T\mathcal{Y}_{(n)}\bigotimes_{\substack{\ell\in 1..K\backslash n\\ \textit{descending}}}\{\mathbf{V}_\ell\Delta_{i_\ell}^T\}] && (31)\\
&= \Delta_{i_n}\mathbf{V}_n^T\operatorname*{var}_{\mathcal{Y},\vec{i}_{\backslash n}}[\mathcal{Y}_{(n)}\bigotimes_{\substack{\ell\in 1..K\backslash n\\ \textit{descending}}}\{\mathbf{V}_\ell\Delta_{i_\ell}^T\}]\mathbf{V}_n\Delta_{i_n}^T && (32)\\
&= \Delta_{i_n}\mathbf{V}_n^T\operatorname*{var}_{\mathcal{Y},\vec{i}_{\backslash n}}[\mathcal{Y}_{(n)}\bigotimes_{\substack{\ell\in 1..K\backslash n\\ \textit{descending}}}\{\vec{v}_{i_\ell}^{(\ell)}\}]\mathbf{V}_n\Delta_{i_n}^T && (33)\\
&= \Delta_{i_n}\mathbf{V}_n^T\operatorname*{\mathbb{E}}_{\mathcal{Y},\vec{i}_{\backslash n}}[\mathcal{Y}_{(n)}\left[\bigotimes_{\substack{\ell\in 1..K\backslash n\\ \textit{descending}}}\{\vec{v}_{i_\ell}^{(\ell)}\vec{v}_{i_\ell}^{(\ell)T}\}\right]\mathcal{Y}_{(n)}^T]\mathbf{V}_n\Delta_{i_n}^T && (34)\\
&= \Delta_{i_n}\mathbf{V}_n^T\operatorname*{\mathbb{E}}_{\mathcal{Y}}\left[\mathcal{Y}_{(n)}\operatorname*{\mathbb{E}}_{\vec{i}_{\backslash n}}\left[\bigotimes_{\substack{\ell\in 1..K\backslash n\\ \textit{descending}}}\{\vec{v}_{i_\ell}^{(\ell)}\vec{v}_{i_\ell}^{(\ell)T}\}\right]\mathcal{Y}_{(n)}^T\right]\mathbf{V}_n\Delta_{i_n}^T && (35)\\
&= \Delta_{i_n}\mathbf{V}_n^T\operatorname*{\mathbb{E}}_{\mathcal{Y}}\left[\mathcal{Y}_{(n)}\left[\bigotimes_{\substack{\ell\in 1..K\backslash n\\ \textit{descending}}}\operatorname*{\mathbb{E}}_{i_\ell}\left[\vec{v}_{i_\ell}^{(\ell)}\vec{v}_{i_\ell}^{(\ell)T}\right]\right]\mathcal{Y}_{(n)}^T\right]\mathbf{V}_n\Delta_{i_n}^T && \text{(Multilinearity of }\textstyle\bigotimes\text{)}\\
& && (36)\\
&= \Delta_{i_n}\mathbf{V}_n^T\operatorname*{\mathbb{E}}_{\mathcal{Y}}\left[\mathcal{Y}_{(n)}\left[\bigotimes_{\substack{\ell\in 1..K\backslash n\\ \textit{descending}}}\frac{1}{d_\ell}\sum_{i_\ell}\left[\vec{v}_{i_\ell}^{(\ell)}\vec{v}_{i_\ell}^{(\ell)T}\right]\right]\mathcal{Y}_{(n)}^T\right]\mathbf{V}_n\Delta_{i_n}^T && (37)\\
&= \Delta_{i_n}\mathbf{V}_n^T\operatorname*{\mathbb{E}}_{\mathcal{Y}}\left[\mathcal{Y}_{(n)}\frac{1}{m_n}\left[\bigotimes_{\substack{\ell\in 1..K\backslash n\\ \textit{descending}}}\mathbf{V}_\ell^T\mathbf{I}\mathbf{V}_\ell\right]\mathcal{Y}_{(n)}^T\right]\mathbf{V}_n\Delta_{i_n}^T && \text{(Sum notation of eigendecomposition)}\\
& && (38)\\
&= \Delta_{i_n}\mathbf{V}_n^T\frac{1}{m_n}\operatorname*{\mathbb{E}}_{\mathcal{Y}}\left[\mathcal{Y}_{(n)}\mathcal{Y}_{(n)}^T\right]\mathbf{V}_n\Delta_{i_n}^T && (39)\\
&\approx \Delta_{i_n}\mathbf{V}_n^T\mathbf{S}_n\mathbf{V}_n\Delta_{i_n}^T && (40)
\end{aligned}
$$

Using this, we define our heuristic as follows.

$$
\begin{aligned}
\operatorname*{var}_{\mathcal{X}}[\mathcal{X}_{i_1\dots i_K}] &\approx \operatorname*{var}_{\mathcal{X},\vec{i}_{\backslash n}}[\mathcal{X}_{i_1\dots i_K}] && (41)\\
&\approx \Delta_{i_n}\mathbf{V}_n^T\mathbf{S}_n\mathbf{V}_n\Delta_{i_n}^T && (42)
\end{aligned}
$$

The heuristic only depends on the first dimension's eigenvectors, which means that the eigenvalues for the other dimensions won't typically be in the same order as their corresponding eigenvectors.[6] To get around this, we run the calculation multiple times, once for each dimension.

[6]Nor would we necessarily expect the eigenvalues to be good approximations even if they were in the correct order, as a lot of the information on the other axes is lost in our heuristic.

A.8 Monte Carlo Speedup

As the speedup is technical, it is helpful to see a worked example for the (2, 2, 3, 2) tensor case. We have $\mathbf{B}\mathbf{\Lambda} = \mathbf{a}$ and we want to find a smaller matrix $\tilde{\mathbf{B}}$ which preserves the system. The idea is to guess the value of one eigenvalue for each axis, as doing so will fix a unique solution for the rest of them. A series of algebraic manipulations will make finding such a unique solution easy. In doing so we only look at a small subset of $\mathbf{B}$ ($\tilde{\mathbf{B}}$) and hence need to make several guesses and average out the result so that our solution is not overly affected by any individual guess.

We'll choose $(\lambda_2^1 = 1, \lambda_1^2 = 1, \lambda_2^3 = 1, \lambda_2^4 = 1)$ as our initial guesses for this worked example. In fact, we do not need to make a guess for one of the axes (the rank of the matrix is $K - 1$ less than maximal, so we only need $K - 1$ guesses). However it's simpler to describe if we ignore this for now. Note that if we simultaneously swap the i, jth columns of $\mathbf{B}$ and the i, jth rows of $\mathbf{\Lambda}$, the system is preserved. Likewise for the i, jth rows of $\mathbf{B}$ and the i, jth rows of $\mathbf{a}$. In light of that, we will express the system as:

$$\begin{array}{c|c} \mathbf{B} & \mathbf{a} \\ \hline \mathbf{\Lambda}^T & \end{array}$$

and perform a series of row and column swaps to simplify it. We'll say that two systems of equations are equal if they can be arranged into each other through a series of permutations and removal of redundant[7] rows/columns.

$$\begin{array}{c|c} \mathbf{B} & \mathbf{a} \\ \hline \mathbf{\Lambda}^T & \end{array} = \begin{array}{cc:cc:ccc:cc|c}
1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & a_1 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & a_2 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & a_3 \\
0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & a_4 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & a_5 \\
0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & a_6 \\
1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & a_7 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & a_8 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & a_9 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & a_{10} \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & a_{11} \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & a_{12} \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & a_{13} \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & a_{14} \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & a_{15} \\
0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & a_{16} \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & a_{17} \\
0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & a_{18} \\
1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & a_{19} \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & a_{20} \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & a_{21} \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & a_{22} \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & a_{23} \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & a_{24} \\
\hline
\lambda_1^1 & \lambda_2^1 & \lambda_1^2 & \lambda_2^2 & \lambda_1^3 & \lambda_2^3 & \lambda_3^3 & \lambda_1^4 & \lambda_2^4 &
\end{array} \tag{43}$$

We want to re-order this system so that it behaves as if we had chosen $(\lambda_1^1, \lambda_1^2, \lambda_1^3, \lambda_1^4)$ as our initial guesses (i.e. guessing the first eigenvalue of each axis). Note that if we swapped the first and second columns, and then swapped every odd column with every even row, the matrix would look exactly the same but now the λ^1 we guessed is the first column rather than the second. A similar line of reasoning applies to the other guesses we made, although instead of swapping individual rows we need to swap chunks of rows. The size of the chunks would be 2 for the second guess, 4 for the third guess, and 12 for the fourth guess. This is because we want to preserve the structure from the previous guesses -

[7] In practice, these equations are not consistent, so there is overdeterminedness that is not actually redundant - hence the need to run multiple iterations of this and average the result.

the size of the tensor is (2, 2, 3, 2) and hence the size of the chunks are $(1, 2, 2 \times 2, 2 \times 2 \times 3)$. In general, the size of the chunks will be $(1, d_1, \prod_{\ell=1}^{2} d_\ell, \prod_{\ell=1}^{3} d_\ell, ..., \prod_{\ell=1}^{K-1} d_\ell)$.

After doing this, you'll get a system looking like this:

$$\begin{array}{c|c} \mathbf{B} & \mathbf{a} \\ \hline \mathbf{\Lambda}^T & \end{array} = \begin{array}{cc:cc:ccc:cc|c}
1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & a_{18} \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & a_{17} \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & a_{20} \\
0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & a_{19} \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & a_{14} \\
0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & a_{13} \\
1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & a_{16} \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & a_{15} \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & a_{22} \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & a_{21} \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & a_{24} \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & a_{23} \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & a_{6} \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & a_{5} \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & a_{8} \\
0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & a_{7} \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & a_{2} \\
0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & a_{1} \\
1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & a_{4} \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & a_{3} \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & a_{10} \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & a_{9} \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & a_{12} \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & a_{11} \\
\hline
\lambda_2^1 & \lambda_1^1 & \lambda_1^2 & \lambda_2^2 & \lambda_2^3 & \lambda_1^3 & \lambda_3^3 & \lambda_2^4 & \lambda_1^4 &
\end{array} \tag{44}$$

Note that the $\mathbf{B}$ component did not change, as expected, but the order of $\mathbf{a}$ and $\mathbf{\Lambda}$ did. We can then shrink this matrix by grabbing the first row, and every row that differs from it by the position of exactly one of the 1s.

$$\begin{array}{c|c} \mathbf{B} & \mathbf{a} \\ \hline \mathbf{\Lambda}^T & \end{array} = \begin{array}{cc:cc:ccc:cc|c}
1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & a_{18} \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & a_{17} \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & a_{20} \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & a_{14} \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & a_{22} \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & a_{6} \\
\hline
\lambda_2^1 & \lambda_1^1 & \lambda_1^2 & \lambda_2^2 & \lambda_2^3 & \lambda_1^3 & \lambda_3^3 & \lambda_2^4 & \lambda_1^4 &
\end{array} \tag{45}$$

The system of equations is no longer overdetermined but is still a subset of the old system. Our goal is now to reshape this into an upper triangular matrix. The columns corresponding to the values we guessed are mostly 1s, except for $d_\ell - 1$ zeros. The other columns form a zero-padded identity matrix when taken together. Move all of our guess columns to the end, and then the top row to the bottom:

$$\begin{array}{c|c} \mathbf{B} & \mathbf{a} \\ \hline \mathbf{\Lambda}^T & \end{array} = \begin{array}{ccccc:cccc|c}
1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & a_{17} \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & a_{20} \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & a_{14} \\
0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & a_{22} \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & a_{6} \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & a_{18} \\
\hline
\lambda_1^1 & \lambda_2^2 & \lambda_1^3 & \lambda_3^3 & \lambda_1^4 & \lambda_2^1 & \lambda_1^2 & \lambda_2^3 & \lambda_2^4 &
\end{array} \tag{46}$$

We can treat our guesses as a linear equation: if we add these equations to our matrix then it will become square:

$$
\begin{array}{c|c}
\mathbf{B} & \mathbf{a} \\
\hline
\mathbf{\Lambda}^T &
\end{array}
=
\left[\begin{array}{ccccc|cccc|c}
1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & a_{17} \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & a_{20} \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & a_{14} \\
0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & a_{22} \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & a_{6} \\
\hline
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & a_{18} \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \lambda_1^2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \lambda_2^3 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \lambda_2^4 \\
\hline
\lambda_1^1 & \lambda_2^2 & \lambda_1^3 & \lambda_3^3 & \lambda_1^4 & \lambda_2^1 & \lambda_1^2 & \lambda_2^3 & \lambda_2^4 &
\end{array}\right]
\tag{47}
$$

Thus, we never actually need to create $\mathbf{B}$. We can directly create $\tilde{\mathbf{B}}$, and perform the demonstrated permutations on $\mathbf{a}$ and $\mathbf{\Lambda}$. As mentioned earlier, we don't actually need a guess for the first dimension (λ^1). To solve the system for $\mathbf{\Lambda}$, we need to find the inverse of $\tilde{\mathbf{B}}$. We've partitioned the matrix into four blocks (indicated by the dashed lines) - we can then invert this matrix using the block matrix inversion formula. Because of the simple forms of the submatrices involved, this is a cheap operation.

$$
\tilde{\mathbf{B}}^{-1} = \left[\begin{array}{ccccc|cccc}
1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right]^{-1}
\tag{48}
$$

$$
= \left[\begin{array}{c|c}
\mathbf{I} & \mathbf{M} \\
\hline
\mathbf{0} & \mathbf{N}
\end{array}\right]^{-1}
\tag{49}
$$

$$
= \left[\begin{array}{c|c}
\mathbf{I} & -\mathbf{M}\mathbf{N}^{-1} \\
\hline
\mathbf{0} & \mathbf{N}^{-1}
\end{array}\right]
\tag{50}
$$

The last step follows from the application of the block matrix inverse formula. Note that we have reduced the computation of $\tilde{\mathbf{B}}^{-1}$ to the inverse of a $K \times K$ matrix ($\mathbf{N}$) and one matrix multiplication.

We can see that $\mathbf{N}$ will always have the form it does (zeros except for ones at the diagonal and first row). The ones on the diagonal are because we added the guesses at the end, and the ones on the first row are because we had a 1 at the top of every 'guess' column. It is not hard to see that the inverse of such a matrix will have 1s on the diagonal, -1 on the off-diagonal first row, and 0s elsewhere. Hence, we can just construct its inverse directly.

We are interested in constructing $-\mathbf{M}\mathbf{N}^{-1}$ directly too. For a small number of iterations b, this is not a bottleneck - however, by computing it directly we can leverage sparse matrix multiplication routines in a manner such that we can easily run this computation a thousand times[8] before the computation time becomes noticeable. We can compute this product easily using block matrix multiplication.

[8]Which in practice is more than enough for our Monte Carlo approximation to be as accurate as the non-approximative version of the algorithm.

$$
-\mathbf{M}\mathbf{N}^{-1} = -\left[\begin{array}{c|ccc} 0 & 1 & 1 & 1 \\ \hline 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{array}\right] \left[\begin{array}{c|ccc} 1 & -1 & -1 & -1 \\ \hline 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right] \tag{51}
$$

$$
= -\left[\begin{array}{c|c} 0 + \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} & 0 \begin{bmatrix} -1 & -1 & -1 \end{bmatrix} + \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \mathbf{I} \\ \hline \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + 0 & \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & -1 & -1 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \end{array}\right] \tag{52}
$$

$$
= -\left[\begin{array}{c|c} 0 & \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \\ \hline \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} & -\mathbf{J} + \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \end{array}\right] \tag{53}
$$

$$
= \left[\begin{array}{c|c} 0 & \begin{bmatrix} -1 & -1 & -1 \end{bmatrix} \\ \hline \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{array}\right] \tag{54}
$$

In terms of generalizing this beyond the example, it is easy to check that the upper block of $\mathbf{M}$ should have $d_1 - 1$ rows (but it will be the same row, repeated). This translates into $-\mathbf{M}\mathbf{N}^{-1}$ also having the upper block repeated $d_1 - 1$ times. The lower right-hand block is always the same as in $\mathbf{M}$, but with the 1s and 0s swapped: this is due to the outer product of a vector of 1s and a vector of -1s being $-\mathbf{J}$, in line 52, regardless of the size of the inputs.

We have now reduced a very large matrix inversion and multiplication problem into a series of small, sparse matrix multiplications with a pre-computed matrix. This pushes the runtime bottleneck of our algorithm solely onto the eigendecomposition used in Theorem 1.

The most expensive part of this reduction is the permutations we perform on $\mathbf{a}$. It's not noticeable for 2-axis tensor inputs, but can be noticeable for 3-axis inputs. Since after permuting $\mathbf{a}$ we select a subset of it, we can achieve further speedups by directly working out the indices of the desired elements of $\mathbf{a}$, rather than permuting them. For details on this final speedup, we refer the reader to our Python implementation of antGLasso.

A.9 Asymptotic Memory Complexity

Given an input tensor with d_ℓ elements along the ℓth axis, the output graphs are of size d_ℓ^2. Thus it is not possible to have a better memory complexity than $O(\sum_\ell d_\ell^2)$. Note that S_ℓ, V_ℓ have size d_ℓ^2.

The matrix $\tilde{B}^{-1}$ from the Monte Carlo speedup has size K^2 (where $K = \sum_\ell d_\ell$), but is sparse. As seen in Section A.8, $\tilde{B}^{-1}$ can be divided into four blocks, with the top left block being zero, the top right block having $O(d_1 K)$ ones, the bottom left block having $O(K)$ zeros, and the bottom right block having $O(K)$ ones. We can always reshape the input so that the first index is the smallest, giving that $O(d_1 K) \subset O(\sum_\ell \max(d_\ell, d_1)) = O(\sum_\ell d_\ell^2)$.

The cause of non-optimal memory complexity is the step in which we estimate the variances, as there are $\prod_\ell d_\ell$ variances to estimate. This is the same as the size of the input. Thus the vanilla variant of antGLasso is not necessarily memory-optimal in the sense of the amount of new space it takes up ($\prod_\ell d_\ell$), but it is still bounded by the size of the input. Heuristic antGLasso only estimates d_ℓ variances at a time, and hence maintains optimality.

In the many-axis case, concerns over memory usage are not so important, as the size of the input grows exponentially in the number of dimensions.

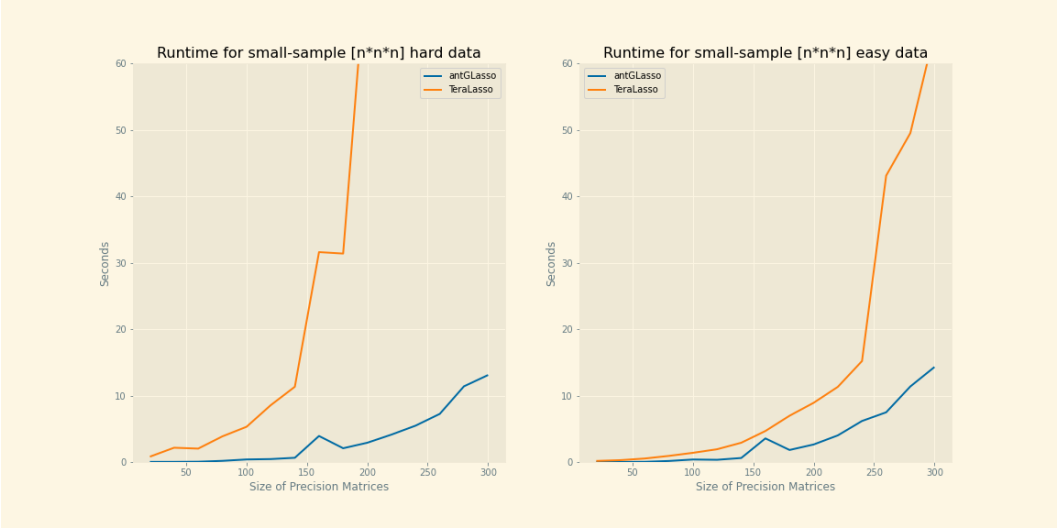

Figure 8: Runtime for 3-axis tensor data, comparing antGLasso with TeraLasso.

A.10 Runtimes for Tensor Data

Figure 8 shows a runtime comparison for antGLasso and TeraLasso. In the three-axis case, antGLasso is still substantially faster than alternatives.