antGLasso: An Efficient Tensor Graphical Lasso Algorithm
Supplementary Material

Bailey Andrew
School Computing
University of Leeds
Leeds, UK LS2 9JT
sceba@leeds.ac.uk

David R. Westhead
Faculty of Biological Sciences
University of Leeds
Leeds, UK LS2 9JT
D.R.Westhead@leeds.ac.uk

Luisa Cutillo
School of Mathematics
University of Leeds
Leeds, UK LS2 9JT
L.Cutillo@leeds.ac.uk

Contents

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 4.2, 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 6. 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 1 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.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

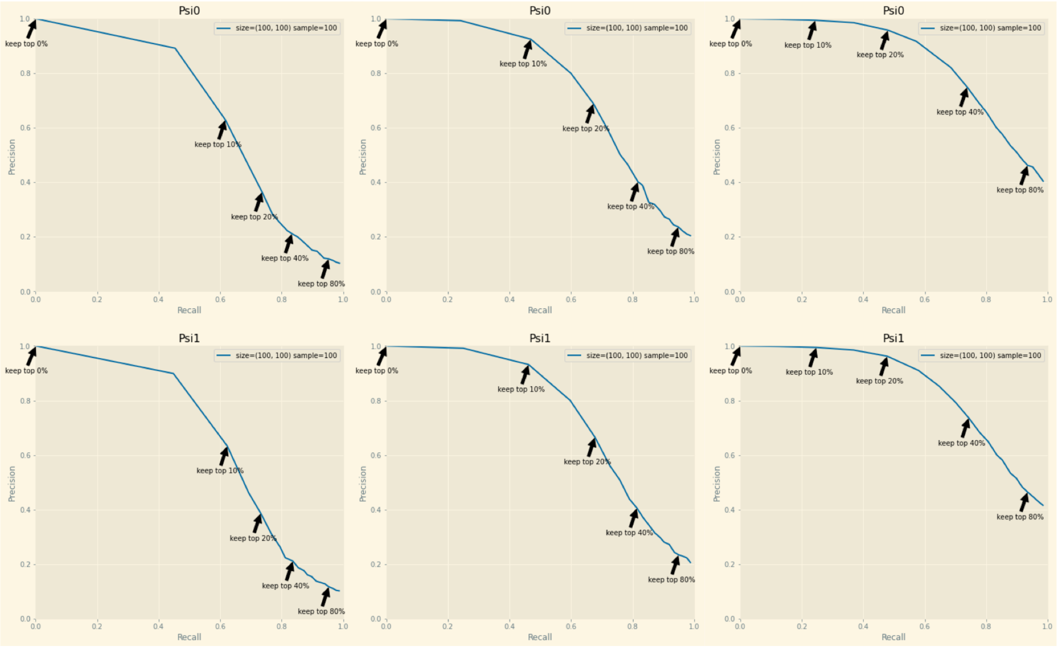

Figure 1: (Left column) Precision recall curves for simulated data with 10% sparsity. (Middle) 20% sparsity. (Right) 40% sparsity. This experiment was performed on two-axis data, with the rows of the figure representing the axes. Arrows indicate value of the regularization hyperparameter.

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 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,[1] and hence the Hadamard product of it with our precision matrix will also be positive definite.

3 Pseudocode

We give the pseudocode for antGLasso in Algorithm 1 (with the Monte Carlo speedup described in Section 6).

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

[1] $\mathbf{N} = \mathbf{M} + (1 - c)\mathbf{I}$, 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: ${\mathcal{Y}^{(n)}_{d_1 \times ... \times d_K}}, {\beta_\ell}, b$
Output: ${\mathbf{\Psi}_\ell}$
  for $1 \leq \ell \leq n$
    $\mathbf{S}_\ell \leftarrow \frac{1}{nm_\ell} \sum_i^n \mathbf{Y}^{(i)}_{(\ell)} \mathbf{Y}^{(i)T}_{(\ell)}$
    $\mathbf{V}_\ell \leftarrow$ eigenvectors$[\mathbf{S}_\ell \circ \mathbf{K}^{2m_\ell - 1}_{m_\ell}]$
  end for
  for $1 \leq p \leq m$
    $\mathcal{X}^{(p)} \leftarrow \mathcal{Y}^{(p)} \times_1 \mathbf{V}_1 \times_2 ... \times_K \mathbf{V}_K$
  end for
  for $\vec{i} \in [1, ..., d_1] \times ... \times [1, ..., d_K]$
    $\mathbf{a}_{\sum_\ell i_\ell d_{1:(\ell-1)}} \leftarrow \frac{n}{\sum_p^n (x^{(p)}_{i_1, \dots, i_K})^2}$
  end for
  $\mathbf{\Lambda} \leftarrow \mathbf{0}$
  Construct $\tilde{\mathbf{B}}^{-1}$ as described in Section 6.
  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 6.
    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 $1 \leq \ell \leq K$
    $\mathbf{\Psi}_\ell \leftarrow \mathbf{V}_\ell \mathbf{\Lambda}_\ell \mathbf{U}^T_\ell$
    for $i$th row $\mathbf{r}$ in $\mathbf{\Psi}_\ell$
      $\mathbf{r}_{\backslash i} \leftarrow \mathrm{sign}(\mathbf{r}_{\backslash i}) \circ \min(0, |\mathbf{r}_{\backslash i}| - \frac{\beta_\ell}{2})$
    end for
  end for
```

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 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[2]. 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 $\mathcal{Y} \times_1 \mathbf{V}_1 \times_2 \mathbf{V}_2 \times_3 ...$, 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 [4]. We use $\circ$ to represent the Hadamard product, and $tr_n[\mathbf{M}]$ 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.

4.1 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 [2] 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. [3], we can derive the fixed point $\mathbf{S}_\ell - \frac{1}{2m_\ell} \mathbf{S}_\ell \circ \mathbf{I} = \frac{1}{m_\ell} \mathrm{tr}_{m_\ell}[(\bigoplus_i \mathbf{\Psi}_i)^{-1}] - \frac{1}{2m_\ell} \mathrm{tr}_{m_\ell}[(\bigoplus_i \mathbf{\Psi}_i)^{-1}] \circ \mathbf{I}$. The order in which we compute $\bigoplus_i \mathbf{\Psi}_i$ matters - we'll assume here that we've transposed the data such

[2] 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 [4].

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 $\mathbf{\Psi}_{\setminus\ell} = \bigoplus_{i\neq\ell} \mathbf{\Psi}_i$ so that $\bigoplus_i \mathbf{\Psi}_i = \mathbf{\Psi}_\ell \oplus \mathbf{\Psi}_{\setminus\ell}$

$$
\begin{aligned}
\mathbf{S}_\ell - \frac{1}{2m_\ell}\mathbf{S}_\ell \circ \mathbf{I} &= \frac{1}{m_\ell}\mathrm{tr}_{m_\ell}[(\mathbf{\Psi}_\ell \oplus \mathbf{\Psi}_{\setminus\ell})^{-1}] - \frac{1}{m_\ell}\mathrm{tr}_{m_\ell}[(\mathbf{\Psi}_\ell \oplus \mathbf{\Psi}_{\setminus\ell})^{-1}] \circ \mathbf{I} && (1)\\
\mathbf{S}_\ell \circ \mathbf{K}_1^{\frac{2m_\ell - 1}{2m_\ell}} &= \mathrm{tr}_{m_\ell}[(\mathbf{\Psi}_\ell \oplus \mathbf{\Psi}_{\setminus\ell})^{-1}] \circ \mathbf{K}_{\frac{1}{m_\ell}}^{\frac{1}{2m_\ell}} && (2)\\
\mathbf{S}_\ell \circ \mathbf{K}_{m_\ell}^{2m_\ell - 1} &= \mathrm{tr}_{m_\ell}[(\mathbf{\Psi}_\ell \oplus \mathbf{\Psi}_{\setminus\ell})^{-1}] && (3)\\
\mathbf{S}_\ell \circ \mathbf{K}_{m_\ell}^{2m_\ell - 1} &= \mathrm{tr}_{m_\ell}[(\mathbf{V}_\ell \otimes \mathbf{V}_{\setminus\ell})(\mathbf{\Lambda}_\ell \oplus \mathbf{\Lambda}_{\setminus\ell})^{-1}(\mathbf{V}_\ell^T \otimes \mathbf{V}_{\setminus\ell}^T)] && (4)\\
\mathbf{S}_\ell \circ \mathbf{K}_{m_\ell}^{2m_\ell - 1} &= \mathrm{tr}_{m_\ell}[(\mathbf{V}_\ell \otimes \mathbf{I})(\mathbf{\Lambda}_\ell \oplus \mathbf{\Lambda}_{\setminus\ell})^{-1}(\mathbf{V}_\ell^T \otimes \mathbf{I})] && (5)\\
\mathbf{S}_\ell \circ \mathbf{K}_{m_\ell}^{2m_\ell - 1} &= \mathbf{V}_\ell \mathrm{tr}_{m_\ell}[(\mathbf{\Lambda}_\ell \oplus \mathbf{\Lambda}_{\setminus\ell})^{-1}]\mathbf{V}_\ell^T && (6)
\end{aligned}
$$

The penultimate step uses Proposition 3.1 of Li et al. [5], and the last step requires Lemma 2 of Dahl et al. [1]. As $\mathrm{tr}_{m_\ell}[(\mathbf{\Lambda}_\ell \oplus \mathbf{\Lambda}_{\setminus\ell})^{-1}]$ 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 $\mathcal{Y} \sim \mathcal{N}(0, \zeta\{\mathbf{\Psi}_i\}^{-1})$. Then we can diagonalize the precision matrix as follows: $\mathcal{X} = \mathcal{Y} \times_1 \mathbf{V}_1^T \times_2 ... \times_K \mathbf{V}_K^T \sim \mathcal{N}(\mathbf{0}, \zeta\{\mathbf{\Lambda}_i\}^{-1})$.*

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: $(\mathcal{Y} \times_1 \mathbf{V}_1 \times_2 ... \times_K \mathbf{V}_K)_{(\ell)} = \mathbf{V}_\ell \mathcal{Y}_{(\ell)}(\mathbf{V}_K \otimes ... \otimes \mathbf{V}_{\ell+1} \otimes \mathbf{V}_{\ell-1} \otimes ... \otimes \mathbf{V}_1)^T$.

$$
\begin{aligned}
\mathrm{pdf}(\mathcal{X}) &= \mathrm{pdf}(\mathcal{Y}) && (7)\\
&= (2\pi)^{\frac{-\prod_i d_i}{2}} \sqrt{\left|\bigoplus_i \mathbf{\Psi}_i\right|} e^{\frac{-1}{2}\sum_\ell^K \mathrm{tr}[\mathbf{\Psi}_\ell \mathcal{Y}_{(\ell)} \mathcal{Y}_{(\ell)}^T]} && (8)\\
\left|\bigoplus_i \mathbf{\Psi}_i\right| &= \left|\bigotimes_i \mathbf{V}_i\right| \left|\bigoplus_i \mathbf{\Lambda}_i\right| \left|\bigotimes_i \mathbf{V}_i^T\right| && (9)\\
&= \left|\bigoplus_i \mathbf{\Lambda}_i\right| && (10)\\
\mathrm{tr}[\mathbf{\Psi}_\ell \mathcal{Y}_{(\ell)} \mathcal{Y}_{(\ell)}^T] &= \mathrm{tr}[\mathbf{\Psi}_\ell (\mathcal{X} \times_1 \mathbf{V}_1 \times_2 ... \times_K \mathbf{V}_K)_{(\ell)} (\mathcal{X} \times_1 \mathbf{V}_1 \times_2 ... \times_K \mathbf{V}_K)_{(\ell)}^T] && (11)\\
&= \mathrm{tr}[\mathbf{\Psi}_\ell \mathbf{V}_\ell \mathcal{X}_{(\ell)} (\bigotimes_{i\neq\ell} \mathbf{V}_i)^T (\bigotimes_{i\neq\ell} \mathbf{V}_i) \mathcal{X}_{(\ell)}^T \mathbf{V}_\ell^T] && (12)\\
&= \mathrm{tr}[\mathbf{V}_\ell^T \mathbf{\Psi}_\ell \mathbf{V}_\ell \mathcal{X}_{(\ell)} \mathcal{X}_{(\ell)}^T] && (13)\\
&= \mathrm{tr}[\mathbf{\Lambda}_\ell \mathcal{X}_{(\ell)} \mathcal{X}_{(\ell)}^T] && (14)
\end{aligned}
$$

□

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. $\frac{M}{\sum_n^m (x_{i_1 \dots i_K}^{(n)})^2} \approx \sum_\ell^K \lambda_{i_\ell}^\ell$.

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

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

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

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

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

$$\frac{M}{\sum_n^m (x_{i_1 \dots i_K}^{(n)})^2} \approx \sum_\ell^K \lambda_{i_\ell}^\ell \tag{23}$$

□

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 $\frac{M}{\sum_n^m (x_{i_1 \dots i_K}^{(n)})^2}$ 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[3]. However, we can select the least squares solution by multiplying 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 6 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. □

4.2 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.

[3] This system is both overdetermined and undertermined, in the sense that it is a 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.

5 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 vulnerable to 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[6] which would allow us to generalize to non-Gaussian data *à la* Li et al. [5].

Our goal is to find the variance of each element of $\mathcal{X}$, or in other words[4] $\underset{\mathcal{X}}{\mathrm{var}}[\mathcal{X}_{i_1\ldots i_K}]$. The idea of the heuristic is to fix one index, i_n, and consider the remaining indices as random variables $\vec{i}_{\backslash n}$. That is, we wish to find $\underset{\mathcal{X},\vec{i}_{\backslash n}}{\mathrm{var}}[\mathcal{X}_{i_1\ldots i_K}]$.

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

$$\begin{aligned}
\underset{\mathcal{X},\vec{i}_{\setminus n}}{\operatorname{var}}[\mathcal{X}_{i_1 \dots i_K}] &= \underset{\mathcal{Y},\vec{i}_{\setminus n}}{\operatorname{var}}[[\![\mathcal{Y}; \{\mathbf{V}_\ell^T\}]\!]_{i_1 \dots i_K}] && (29)\\
&= \underset{\mathcal{Y},\vec{i}_{\setminus n}}{\operatorname{var}}[[\![\mathcal{Y}; \{\Delta_{i_\ell}\mathbf{V}_\ell^T\}]\!]] && (30)\\
&= \underset{\mathcal{Y},\vec{i}_{\setminus n}}{\operatorname{var}}[\Delta_{i_n}\mathbf{V}_n^T\mathcal{Y}_{(n)} \bigotimes_{\substack{\ell\in 1..K\setminus n\\ \textit{descending}}} \{\mathbf{V}_\ell\Delta_{i_\ell}^T\}] && (31)\\
&= \Delta_{i_n}\mathbf{V}_n^T \underset{\mathcal{Y},\vec{i}_{\setminus n}}{\operatorname{var}}[\mathcal{Y}_{(n)} \bigotimes_{\substack{\ell\in 1..K\setminus 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 \underset{\mathcal{Y},\vec{i}_{\setminus n}}{\operatorname{var}}[\mathcal{Y}_{(n)} \bigotimes_{\substack{\ell\in 1..K\setminus n\\ \textit{descending}}} \{\vec{v}_{i_\ell}^{(\ell)}\}]\mathbf{V}_n\Delta_{i_n}^T && (33)\\
&= \Delta_{i_n}\mathbf{V}_n^T \underset{\mathcal{Y},\vec{i}_{\setminus n}}{\mathbb{E}}[\mathcal{Y}_{(n)} \left[\bigotimes_{\substack{\ell\in 1..K\setminus 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 \underset{\mathcal{Y}}{\mathbb{E}}\left[\mathcal{Y}_{(n)} \underset{\vec{i}_{\setminus n}}{\mathbb{E}}\left[\bigotimes_{\substack{\ell\in 1..K\setminus 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 \underset{\mathcal{Y}}{\mathbb{E}}\left[\mathcal{Y}_{(n)} \left[\bigotimes_{\substack{\ell\in 1..K\setminus n\\ \textit{descending}}} \underset{i_\ell}{\mathbb{E}}\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 } \bigotimes\text{)}\\
& && (36)\\
&= \Delta_{i_n}\mathbf{V}_n^T \underset{\mathcal{Y}}{\mathbb{E}}\left[\mathcal{Y}_{(n)} \left[\bigotimes_{\substack{\ell\in 1..K\setminus 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 \underset{\mathcal{Y}}{\mathbb{E}}\left[\mathcal{Y}_{(n)} \frac{1}{m_n}\left[\bigotimes_{\substack{\ell\in 1..K\setminus 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}\underset{\mathcal{Y}}{\mathbb{E}}\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.

$$\begin{aligned}
\underset{\mathcal{X}}{\operatorname{var}}[\mathcal{X}_{i_1 \dots i_K}] &\approx \underset{\mathcal{X},\vec{i}_{\setminus n}}{\operatorname{var}}[\mathcal{X}_{i_1 \dots i_K}] && (41)\\
&\approx \Delta_{i_n}\mathbf{V}_n^T\frac{1}{m_n}\mathbf{S}_n\mathbf{V}_n\Delta_{i_n}^T && (42)
\end{aligned}$$

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[5]. To get around this, we run the calculation multiple times, once for each dimension.

[5] 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.

6 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[6] 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 -

[6] 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:

$$\begin{array}{c|c} \mathbf{B} & \mathbf{a} \\ \hline \mathbf{\Lambda}^T & \end{array} = \begin{array}{ccccc:cccc|c} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & a_{17} \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & a_{20} \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & a_{14} \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & a_{22} \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & a_6 \\ \hdashline 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & a_{18} \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \lambda_1^2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \lambda_2^3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \lambda_2^4 \\ \hline \lambda_1^1 & \lambda_2^2 & \lambda_1^3 & \lambda_3^3 & \lambda_1^4 & \lambda_2^1 & \lambda_1^2 & \lambda_2^3 & \lambda_2^4 & \end{array} \tag{47}$$

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.

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

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

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

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[7] before the computation time becomes noticeable. We can compute this product easily using block matrix multiplication.

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

$$-\mathbf{MN}^{-1} = -\left[\begin{array}{c:ccc} 0 & 1 & 1 & 1 \\ \hdashline 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{array}\right] \left[\begin{array}{c:ccc} 1 & -1 & -1 & -1 \\ \hdashline 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right] \tag{51}$$

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

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

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

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 $-\mathbf{MN}^{-1}$ 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 43, 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.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

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

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