antGLasso

People in Shoes

June 2022

Contents

1 antGLasso

1.1 Notation

1.1.1 Tensor-Specific Notation

We will consider K-dimensional tensors $\mathcal{Y}$ of size $d_1, ..., d_K$. $m_\ell = \frac{(\prod_i^K d_i)}{d_\ell}$ amount of data in one index of the ℓth dimension.

For an introduction to tensor mathematics, Kolda and Bader (2009) is an invaluable resource. The mode-ℓ 'matricization' of a tensor $\mathcal{Y}$ is denoted $\mathcal{Y}_\ell$ is the matrix whose columns are the mode-ℓ fibers of the original tensor. A mode-ℓ fiber of a matrix is obtained by fixing all except the ℓth index, which varies over all allowed values. The mode-1 fibers of a matrix would be the rows, and the mode-2 fibers would be the columns. For higher order tensors, there is not a standard way to order the fibers in the mode-ℓ matricization, but as long as one is consistent it will not matter.

The last tensor-specific notation is that of mode-ℓ multiplication between a tensor and a matrix, denoted $\mathcal{Y}_{i_1,...,i_K} \times_\ell \mathbf{V}$ (where $\mathbf{V}$ has dimensions i_ℓ by j). Intuitively, this is matrix multiplication 'along' the ℓth dimension. The $(i_1, ..., i_{\ell-1}, j, i_{\ell+1}, ..., i_K)$th element of $\mathcal{Y} \times_\ell \mathbf{V}$ is $\sum_{i_\ell} \mathcal{Y}_{(i_1,...,i_K)} \mathbf{V}_{i_\ell, j}$. Note that when $\mathcal{Y}$ is a matrix $\mathbf{Y}$, then $\mathbf{Y} \times_1 \mathbf{V} = \mathbf{V}^T\mathbf{Y}$ and $\mathbf{Y} \times_2 \mathbf{V} = \mathbf{YV}$. One particularly useful property of mode-ℓ multiplication and matricizations is the following: $(\mathcal{Y} \times_1 \mathbf{V}_1 \times_2 ... \times_K \mathbf{V}_K)_{(\ell)} = \mathbf{V}_\ell \mathcal{Y}_{(\ell)} (\mathbf{V}_K \otimes ... \otimes \mathbf{V}_{\ell+1} \otimes \mathbf{V}_{\ell-1} \otimes ... \otimes \mathbf{V}_1)^T$.

1.1.2 Graphical-Lasso-Specific Notation

Suppose we have a set of m samples of tensor data $\{\mathcal{Y}^{(n)}\}$. Then the empirical covariance matrix over dimension ℓ can be computed as $\mathbf{S}_\ell = \frac{1}{mm_\ell} \sum_n \mathcal{Y}_{(\ell)}^{(n)} \mathcal{Y}_{(\ell)}^{(n)^T}$. We wish to estimate the precision matrices $\{\mathbf{\Psi}_\ell\}$, which have eigenvectors $\mathbf{V}_\ell$ and eigenvalues $\mathbf{\Lambda}_\ell$. $\text{tr}_x[\mathbf{X}]$ is the blockwise trace obtained by partitioning the matrix into x by x blocks and creating a matrix of the traces of these blocks. $\oplus$ is the Kronecker Sum, defined as $\mathbf{X} \oplus \mathbf{Y} = \mathbf{X} \otimes \mathbf{I} + \mathbf{I} \otimes \mathbf{Y}$.

1.2 Structure of Proof

First, we will manipulate the fixed point found by Kalaitzis et al. (2013) to find the estimated eigenvectors of the precision matrices. We will then show that using these eigenvectors you can 'diagonalize' the distribution such that we can reinterpret the data as being drawn from diagonal precision matrices. A linear transformation of the reinterpreted data produces the eigenvalues. The naive way to do this requires non-optimal asymptotic memory - the rest of the proof is devoted to making this memory-optimal and efficient.

1.3 Theorem 1

Theorem 1. $\mathbf{V}_\ell$ *are the eigenvectors of* $\mathbf{S}_\ell \circ \mathbf{K}_\ell^{2m_\ell - 1}$.

Proof. Greenewald et al. (2017) present a maximum likelihood estimator for the tensor variate case: $-\log \left| \bigoplus_{\ell=1}^{K} \mathbf{\Psi}_\ell \right| + \sum_{\ell=1}^{K} m_k \mathbf{S}_\ell^T \mathbf{\Psi}_\ell$. Using the exact same argument as Kalaitzis et al. (2013), 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 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 $\bigoplus_i \mathbf{\Psi}_i = \mathbf{\Psi}_\ell \oplus \mathbf{\Psi}_{\setminus \ell}$

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

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

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

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

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

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

The penultimate step uses Proposition 3.1 of Li et al. (2022), and the last step requires Lemma 2 of Dahl et al. (2013). 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 $\mathbf{S}_\ell \circ \mathbf{K}_{m_\ell}^{2m_\ell - 1}$.

□

1.4 Lemma 1

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

Proof. We will show that the density function of $\mathcal{X}$ is that of a Kronecker sum distribution with the desired parameters.

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

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

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

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

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

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

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

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

□

1.4.1 Corollary 1

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

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

Proof.

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

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

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

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

□

1.5 Lemma 2

Lemma 2. $\frac{1}{\mathbf{\Sigma}^{(i_1 \dots i_{K-1})}_{i_K, i_K}} = \sum_{\ell}^{K} \lambda_{i_\ell}^{\ell}$.

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

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

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

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

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

$$\frac{M}{\sum_{n}^{m} (x^{(n)}_{i_1 \dots i_K})^2} = \sum_{\ell}^{K} \lambda_{i_\ell}^{\ell} \tag{23}$$

□

1.6 Theorem 2

This theorem is intimidating to formalize, but is actually very intuitive if you follow along with pen and paper - we give a worked example in the Supplementary Material.

Theorem 2. *Let* $\mathbf{a}$ *be the vector of length* $\prod_\ell^K d_\ell$ *whose* $(\sum_\ell i_\ell d_{1:(\ell-1)})$*th element is* $\frac{1}{\mathbf{\Sigma}^{(i_1 \cdots i_{K-1})}_{i_K, i_K}}$*. Then* $\mathbf{\Lambda} \approx \frac{1}{b} \sum_{\vec{i}}^{b} \mathbf{P}_1^{\vec{i}} \tilde{\mathbf{B}}^{-1} \mathbf{P}_2^{\vec{i}} \begin{bmatrix} \mathbf{a}^{\vec{i}} \\ \lambda_{\vec{i}}^{1:(K-1)} \end{bmatrix}$ *where* $\lambda_{\vec{i}}^{1:(K-1)}$ *is the vector of length* $K-1$ *whose* ℓ*th element is* $\lambda_{i_\ell}^{\ell}$ *and the sum is over* b *randomly chosen tuples of indices* $\vec{i}$*.*

Proof. Lemma 2 gives us a straightforward way to express $\mathbf{\Lambda}$ as a system of equations with corresponding matrix $\mathbf{B}$. This equation is heavily overdetermined, but there is an easy way to reduce it to a system describable by a $(\sum_\ell^K d_\ell) - K + 1$ by $\sum_\ell^K d_\ell$ matrix. Suppose we know $\lambda_{\vec{i}}^{1:(K-1)}$ (this accounts for the $K-1$ degrees of freedom). In what follows, we'll use $\_$ as a placeholder for a specific index of $\mathbf{a}$, i.e. $\mathbf{a}_\_$. We do this because the specific index will not matter in our proof, and would have a complicated form if given explicitly which would distract from the topic at hand.

We can create our smaller matrix as follows: choose the first row, i.e. the one corresponding to $\sum_\ell^K \lambda_{i_\ell}^\ell = \mathbf{a}_\_$. We know all values except for λ_1^K, and hence we can solve for it. We can then take all rows corresponding to a sum of the form $(\sum_{\ell \neq 1}^K \lambda_1^\ell) + \lambda_i^1 = \mathbf{a}_x$, which we can use to deduce all λ_j^1. We can do a similar process for each λ_j^ℓ, each time adding $d_\ell - 1$ rows to our matrix. At the end, we will be able to solve for every value of λ. We'll let $\mathbf{a}^{\vec{i}}$ be the sub-vector of $\mathbf{a}$ containing the analogous rows to those picked for our system of equations.

Consider the resulting matrix. Move the $(i_\ell + \sum_j^{\ell-1} d_j)$th columns (for every ℓ) to the rightmost column of the matrix (so the i_ℓth column, which corresponded to $\lambda_{i_\ell}^\ell$, will become the $(\ell + \sum_j^K d_j)$th column). Next, move the first row to the bottom. This matrix is now non-square upper triangular. We can make it square by appending $K-1$ rows to the bottom with all-zeros except a 1 corresponding to $\lambda_{i_\ell}^\ell$ (i.e. the $(\ell + \sum_j^K d_j)$th column). These rows correspond to the fact that we made initial guesses for $\lambda_{i_\ell}^\ell$.

We will now explicitly define our permutation matrices. We currently have a system of equations $\begin{bmatrix} \mathbf{a}^{\vec{i}} \\ \lambda_{\vec{i}}^{1:(K-1)} \end{bmatrix} = (\mathbf{P}_2^{\vec{i}})^{-1} \tilde{\mathbf{B}} (\mathbf{P}_1^{\vec{i}})^{-1} \hat{\mathbf{\Lambda}}$. $(\mathbf{P}_1^{\vec{i}})^{-1}$ corresponds to the column permutations - thus, $(\mathbf{P}_1^{\vec{i}})^{-1}$ is the permutation matrix which moves the $(i_\ell + \sum_j^{\ell-1} d_j)$th column to the $(\ell + \sum_j^K d_j)$ column. This is not a swap - intuitively, we remove every $(i_\ell + \sum_j^{\ell-1} d_j)$th column at once, and then append them directly to the end of the matrix. $(\mathbf{P}_2^{\vec{i}})^{-1}$ is the permutation matrix for the rows. We moved the top row to the bottom, but then latter added $K-1$ new rows - hence $(\mathbf{P}_2^{\vec{i}})^{-1}$ is the permutation matrix which moves row 1 to row $(\sum_\ell^K d_\ell) - (K-1)$. Like before, this is not a swap - intuitively, we remove the

first row and then insert it between what was originally the last row and the added rows.

$\tilde{\mathbf{B}}$ is triangular with a unit diagonal, and is thus invertible. We can now solve for $\mathbf{\Lambda}$: $\hat{\mathbf{\Lambda}} = \mathbf{P}_1^{\vec{i}} \tilde{\mathbf{B}}^{-1} \mathbf{P}_2^{\vec{i}} \begin{bmatrix} \mathbf{a}^{\vec{i}} \\ \lambda_{\vec{i}}^{1:(K-1)} \end{bmatrix}$. In practice, we want to average our approximation over multiple random selections of $\vec{i}$, as the 'overdeterminedness' of our original equation contains some useful information in finding the best solution to the problem given all the data. In the first iteration, we can set our initial guesses for $\lambda_{\vec{i}}$ to 1 - but for subsequent iterations, it is sensible to feed in our previous iterations' results as the guess. Note that $\tilde{\mathbf{B}}^{-1}$ is the same regardless of $\vec{i}$, and thus can be computed before iterating over b if desired. Additionally, it is sparse: it only requires $O(\sum_\ell^K d_\ell)$ space. $\square$

1.7 Regularization

Bigraphical Lasso 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. 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 $\text{shrink}(\hat{r}) = \text{sign}(\hat{r}) \circ \min(0, |\hat{r}| - \frac{\beta}{2})$ 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. Thus, instead of using β as an argument, we could instead give the algorithm a percent of edge connections to keep. When framed this way, our hyperparameters become easily interpretable!

1.8 Algorithm

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 $1 \leq \ell \leq n$
    $\mathbf{S}_\ell \leftarrow \frac{1}{nm_\ell} \sum_i^n \mathbf{Y}^{(i)}_{(\ell)} \mathbf{Y}^{(i)T}_{(\ell)}$
    $\mathbf{V}_\ell \leftarrow \text{eigenvectors}[\mathbf{S}_\ell \circ \mathbf{K}^{2m_\ell - 1}_{m_\ell}]$
  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 $\vec{i} \in [1, ..., d_1] \times ... \times [1, ..., d_K]$
    $\mathbf{a}_{\sum_\ell i_\ell d_{1:(\ell-1)}} \leftarrow \frac{n}{\sum_p^n (x^{(p)}_{i_1,...,i_K})^2}$
  end for
  $\mathbf{\Lambda} \leftarrow \mathbf{0}$
  Construct $\tilde{\mathbf{B}}^{-1}$ as in Theorem 2
  for b iterations of different tuples $\vec{i}$
    Construct $\mathbf{a}^{\vec{i}}, \mathbf{P}_1^{\vec{i}}, \mathbf{P}_2^{\vec{i}}$ as in Theorem 2
    If first iteration, set each element in $\lambda_{\vec{i}}^{1:(K-1)}$ to 1
    $\mathbf{\Lambda} \leftarrow \mathbf{\Lambda} + \mathbf{P}_1^{\vec{i}} \tilde{\mathbf{B}}^{-1} \mathbf{P}_2^{\vec{i}} \begin{bmatrix} \mathbf{a}^{\vec{i}} \\ \lambda_{\vec{i}}^{1:(K-1)} \end{bmatrix}$
  end for
  $\mathbf{\Lambda} \leftarrow \mathbf{\Lambda}/b$
  for $1 \leq \ell \leq K$
    $\mathbf{\Psi}_\ell \leftarrow \mathbf{V}_\ell \mathbf{\Lambda}_\ell \mathbf{U}_\ell^T$
    for ith row $\mathbf{r}$ in $\mathbf{\Psi}_\ell$
      $\mathbf{r}_{\setminus i} \leftarrow \text{sign}(\mathbf{r}_{\setminus i}) \circ \min(0, |\mathbf{r}_{\setminus i}| - \frac{\beta_\ell}{2})$
    end for
  end for

2 Methods and Results

2.1 Dependencies

The implementation is available on GitHub. It relies on NumPy (Harris et al. (2020)), SciPy (Virtanen et al. (2020)), Sklearn (Pedregosa et al. (2011)), and Matplotlib (Hunter (2007)). We also found CVXPY (Diamond and Boyd (2016), Agrawal et al. (2018)) helpful in prototyping. To interface with Matlab code written by others, we used the Matlab Engine package (citation needed).

2.2 Data Generation Methods

Given precision matrices, we generate gaussian matrix data using the matrix-variate normal with Kronecker Sum structure according to Corollary 1. The precision matrices are generated using a base positive definite matrix variate distribution (specifically the Inverse Wishart), and then 'sparsifying' them by Hadamard-producting them with some positive definite masking matrix represented as the outer product of an i.i.d. vector of bernoulli variables with itself (and then manually setting the diagonals to 1).

2.3 Asymptotic Complexity

Name of Step	Time Complexity	Space Complexity
Input $\{\mathcal{Y}^{(n)}_{d_1\times\ldots\times d_K}\}$	N/A	$O(n\prod_\ell^K d_\ell)$
Creation of $\{\mathbf{S}_\ell\}$	$O(n(\prod_\ell^K d_\ell)(\sum_\ell^K d_\ell))$	$O(\sum_\ell^K d_\ell^2)$
Eigendecomposition of $\{\mathbf{S}_\ell\}$	$O(\sum_\ell^K d_\ell^3)$	$O(\sum_\ell^K d_\ell^2)$ (sizes of $\{\mathbf{V}_\ell\}$)
Creation of $\{\mathcal{X}^{(n)}\}$	$O(n(\prod_\ell^K d_\ell)(\sum_\ell^K d_\ell))$	$O(n\prod_\ell^K d_\ell)$
Multiply by sparse triangular $\tilde{\mathbf{B}}^{-1}$	$O(b\sum_\ell^K d_\ell)$	$O(\sum_\ell^K d_\ell)$
Creation of $\{\Psi_\ell\}$	$O(\sum_\ell^K d_\ell^3)$	$O(\sum_\ell^K d_\ell^2)$
Shrink $\{\Psi_\ell\}$	$O(\sum_\ell^K d_\ell^2)$	In-place modification of $\{\Psi_\ell\}$

Table 1: Time and Space Complexity breakdowns of antGLasso algorithm.

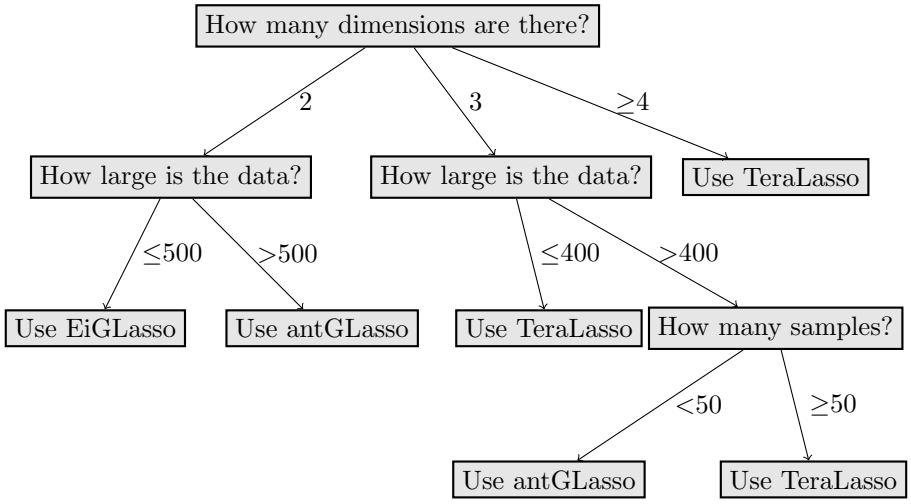
The full space complexity is $O(n\prod_\ell^K d_\ell + \sum_\ell^K d_\ell^2)$, which is the size of the inputs and outputs. In the bigraphical case it is $O(nd_1d_2 + d_1^2 + d_2^2)$. Supposing all dimensions are the same size d, then it is $O(d^K)$, i.e. exponential in the number of dimensions.

The full time complexity (assuming b is constant) is $O(n(\prod_\ell^K d_\ell)(\sum_\ell^K d_\ell) + \sum_\ell^K d_\ell^3)$. In the bigraphical case it is $O(nd_1^2d_2 + nd_1d_2^2 + d_1^3 + d_2^3)$. If all dimensions are the same size d, then it is $O(nd^{K+1})$.

2.4 Results

tba

2.5 Recommendations



In short, antGLasso excels at large problems, but if a problem is small enough to be solved by other algorithms then it would be best to use them instead. When the amount of tensor dimensions are high, the most expensive operation is the calculation of the empirical covariance matrices, as this grows exponentially in dimension. The speed gains of antGLasso become irrelevant, and thus for dimension ≥ 4 (and large-sample dimension 3) TeraLasso should be preferred due to its accuracy.

The exact cutoff point between when to use each algorithm will depend on the machine they run on and the patience of the user.

3 Supplementary Material

3.1 Walkthrough of Theorem 2

Since Theorem 2 is technical, it is helpful to see a worked example for the (2, 2, 3, 2) tensor case, where $\mathbf{B}\boldsymbol{\Lambda} = \mathbf{a}$ and we want to find a smaller matrix $\tilde{\mathbf{B}}$ which preserves the system. We'll choose $(\lambda_2^1, \lambda_1^2, \lambda_2^3, \lambda_2^4)$ as our initial guesses. Note that if we simultaneously swap the i, jth columns of $\mathbf{B}$ and the i, jth rows of $\boldsymbol{\Lambda}$, 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:

$\mathbf{B}$	$\mathbf{a}$
$\boldsymbol{\Lambda}^T$	

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[1] rows/columns.

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

$$
\begin{array}{c|c}
\mathbf{B} & \mathbf{a} \\
\hline
\mathbf{\Lambda}^T &
\end{array}
=
\left[\begin{array}{cc:cc:cccc:cc|c}
1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & a_1 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & a_2 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & a_3 \\
0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & a_4 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & a_5 \\
0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & a_6 \\
1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & a_7 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & a_8 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & a_9 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & a_{10} \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & a_{11} \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & a_{12} \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & a_{13} \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & a_{14} \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & a_{15} \\
0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & a_{16} \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & a_{17} \\
0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & a_{18} \\
1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & a_{19} \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & a_{20} \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & a_{21} \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & a_{22} \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & a_{23} \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & a_{24} \\
\hline
\lambda_1^1 & \lambda_2^1 & \lambda_1^2 & \lambda_2^2 & \lambda_1^3 & \lambda_2^3 & \lambda_3^3 & \lambda_1^4 & \lambda_2^4 &
\end{array}\right]
\tag{29}
$$

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. 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 - the size of the tensor is (2, 2, 3, 2) and hence the size of the chunks are $(1, 2, 2 \times 2, 2 \times 2 \times 3)$.

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

$$\begin{array}{c|c} \mathbf{B} & \mathbf{a} \\ \hline \mathbf{\Lambda}^T & \end{array} = \begin{array}{ccc:cc:ccc:cc|c} 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & a_{18} \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & a_{17} \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & a_{20} \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & a_{19} \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & a_{14} \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & a_{13} \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & a_{16} \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & a_{15} \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & a_{22} \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & a_{21} \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & a_{24} \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & a_{23} \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & a_{6} \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & a_{5} \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & a_{8} \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & a_{7} \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & a_{2} \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & a_{1} \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & a_{4} \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & a_{3} \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & a_{10} \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & a_{9} \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & a_{12} \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & a_{11} \\ \hline \lambda_2^1 & \lambda_1^1 & \lambda_1^2 & \lambda_2^2 & \lambda_2^3 & \lambda_1^3 & \lambda_3^3 & \lambda_2^4 & \lambda_1^4 & \end{array} \tag{30}$$

Note that the $\mathbf{B}$ component did not change, 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.

$$\begin{array}{c|c} \mathbf{B} & \mathbf{a} \\ \hline \mathbf{\Lambda}^T & \end{array} = \begin{array}{ccc:cc:ccc:cc|c} 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & a_{18} \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & a_{17} \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & a_{20} \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & a_{14} \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & a_{22} \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & a_{6} \\ \hline \lambda_2^1 & \lambda_1^1 & \lambda_1^2 & \lambda_2^2 & \lambda_2^3 & \lambda_1^3 & \lambda_3^3 & \lambda_2^4 & \lambda_1^4 & \end{array} \tag{31}$$

The system of equations is no longer overdetermined. Our goal is now to reshape this into an upper triangular matrix. Move all of our guess columns to the end, and then the top row to the bottom:

$$\begin{array}{c|c} \mathbf{B} & \mathbf{a} \\ \hline \mathbf{\Lambda}^T & \end{array} = \left[\begin{array}{ccccc:cccc|c} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & a_{17} \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & a_{20} \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & a_{14} \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & a_{22} \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & a_{6} \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & a_{18} \\ \hline \lambda_1^1 & \lambda_2^2 & \lambda_1^3 & \lambda_3^3 & \lambda_1^4 & \lambda_2^1 & \lambda_1^2 & \lambda_2^3 & \lambda_2^4 & \end{array}\right] \tag{32}$$

We then add our guesses in to make the matrix square:

$$\begin{array}{c|c} \mathbf{B} & \mathbf{a} \\ \hline \mathbf{\Lambda}^T & \end{array} = \left[\begin{array}{ccccc:cccc|c} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & a_{17} \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & a_{20} \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & a_{14} \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & a_{22} \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & a_{6} \\ \hdashline 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & a_{18} \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \lambda_1^2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \lambda_2^3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \lambda_2^4 \\ \hline \lambda_1^1 & \lambda_2^2 & \lambda_1^3 & \lambda_3^3 & \lambda_1^4 & \lambda_2^1 & \lambda_1^2 & \lambda_2^3 & \lambda_2^4 & \end{array}\right] \tag{33}$$

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 $\mathbf{\Lambda}$. 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.

$$\tilde{\mathbf{B}}^{-1} = \left[\begin{array}{ccccc:cccc} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ \hdashline 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]^{-1} \tag{34}$$

$$= \left[\begin{array}{c:c} \mathbf{I} & \mathbf{M} \\ \hdashline \mathbf{0} & \mathbf{N} \end{array}\right]^{-1} \tag{35}$$

$$= \left[\begin{array}{c:c} \mathbf{I} & -\mathbf{M}\mathbf{N}^{-1} \\ \hdashline \mathbf{0} & \mathbf{N}^{-1} \end{array}\right] \tag{36}$$

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 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. There is likely a way to construct $-\mathbf{M}\mathbf{N}^{-1}$ directly too, but this is not a bottleneck in the algorithm so we do not.

References

Agrawal, A., Verschueren, R., Diamond, S., & Boyd, S. (2018). A rewriting system for convex optimization problems. *Journal of Control and Decision*, *5*(1), 42–60.

Dahl, A., Hore, V., Iotchkova, V., & Marchini, J. (2013). Network inference in matrix-variate gaussian models with non-independent noise. https://doi.org/10.48550/ARXIV.1312.1622

Diamond, S., & Boyd, S. (2016). CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, *17*(83), 1–5.

Greenewald, K., Zhou, S., & Hero, A. (2017). Tensor graphical lasso (teralasso). https://doi.org/10.48550/ARXIV.1705.03983

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., ... Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Kalaitzis, A., Lafferty, J., Lawrence, N. D., & Zhou, S. (2013). The bigraphical lasso. In S. Dasgupta & D. McAllester (Eds.), *Proceedings of the 30th international conference on machine learning* (pp. 1229–1237). PMLR. https://proceedings.mlr.press/v28/kalaitzis13.html

Kolda, T. G., & Bader, B. W. (2009). Tensor decompositions and applications. *SIAM review*, *51*(3), 455–500.

Li, S., López-García, M., Lawrence, N. D., & Cutillo, L. (2022). Scalable bigraphical lasso: Two-way sparse network inference for count data. https://doi.org/10.48550/ARXIV.2203.07912

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E.

(2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., ... SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2