Evaluating LASSO Performance in High-Dimensional Data: A Comparison Between Pixel and Frequency Domains

Siyang Ren, Nichole E. Carlson, William Lippitt, Yue Wang

Notes

Everything surrounded by [] are my thoughts/questions/to-dos.

1 Introduction

High-dimensional imaging data present unique challenges for predictive modeling and feature selection, largely due to the spatial correlations between neighboring pixels. These correlations can lead to multicollinearity, affecting the stability and efficiency of models. When LASSO is applied to imaging datasets, such spatial dependencies can obscure important features and reduce predictive accuracy if not properly addressed.

This study explores the use of eigendecomposition on the matrix MCM, where M is a centering matrix and C is a spatial adjacency matrix encoding pixel adjacency. The eigenvectors derived from MCM are used to transform imaging data into a "frequency space." While this transformation does not fully decorrelate the data spatially, it reduces spatial dependencies significantly and introduces a structure that enhances interpretability, making MCM a uniquely advantageous choice for such analyses.

To assess the utility of this approach, we simulate imaging datasets with specific sparsity patterns and apply LASSO in both the pixel space and the transformed frequency space. By comparing the performance of these models, we aim to demonstrate the benefits of leveraging **MCM**-based transformations in improving feature selection and predictive performance in high-dimensional imaging data.

2 Methods

This section outlines the methodology used to analyze high-dimensional imaging data, focusing on the transformation of data into a frequency space using spatial adjacency matrices and applying LASSO for feature selection. The narrative progresses from the challenges of spatial correlations in imaging to the construction of the frequency space and evaluation through simulated datasets.

2.1 Spatial Correlations in Imaging Data

Imaging data are often characterized by spatial autocorrelation, where neighboring pixels exhibit similar values due to their proximity. This spatial dependency introduces challenges in predictive modeling, such as multicollinearity, reduced computational efficiency, and obscured spatial signals.

When image data are used as predictors, each pixel serves as a feature to predict a scalar outcome y. Let X represent a dataset of images, where each image x_i is a row vector of p pixels. The predictive model can be expressed as:

$$y_i = x_i.\beta + \epsilon_i,$$

where β is a vector of regression coefficients, and $\epsilon_i \sim N(0, \sigma^2)$ is the error term. In this scenario, the inherent correlations among pixels can lead to multicollinearity, which reduces the stability of coefficient estimation and complicates feature selection.

Alternatively, when image data are treated as outcomes, each pixel becomes a response variable, modeled as a function of predictors. Suppose x_i is a vector of predictors and y_i represents an image outcome with s pixels. The model can be written as:

$$y_i = x_i\beta + \epsilon_i,$$

where β is a $p \times s$ matrix of coefficients and $\epsilon_i \sim \mathcal{N}(0, \Sigma)$ captures the covariance among pixels. A key consideration in this case is the structure of the covariance matrix Σ, which determines how spatial relationships between pixels are modeled. Simplifying Σ improves computational efficiency but may sacrifice accuracy, while a more complex structure captures fine details but increases computational demands.

In both scenarios, the spatial correlations between pixels necessitate strategies to reduce dependencies and improve model performance. Two common approaches are dimension reduction, such as principal component analysis (PCA), and modifying the covariance structure. While PCA effectively reduces dimensionality, it often sacrifices interpretability, as the components are linear combinations of the original features.

This study focuses on modifying the covariance structure by leveraging the eigendecomposition of MCM, where M is a centering matrix and C is a spatial adjacency matrix encoding pixel relationships. The eigenvectors of MCM transform the data into a frequency space, aligning features along orthogonal spatial patterns and reducing spatial dependencies. These eigenvectors are interpretable because they represent spatial patterns inherent to the adjacency structure defined by C. Each eigenvector corresponds to a distinct spatial variation, ordered by its associated eigenvalue, which quantifies the strength of spatial autocorrelation along that pattern. Although the transformation does not fully decorrelate the data, it balances reduced spatial autocorrelation with the preservation of meaningful spatial relationships. This interpretability, combined with computational efficiency, makes MCM-based transformations particularly suitable for high-dimensional imaging analyses with LASSO.

2.2 Frequency Space

In this subection, we firstly introduce Moran's Coefficient (MC), a common measure of spatial autocorrelation, and then introduce a specially constructed spatial adjacency matrix combination, MCM, and then show how the eigenvectors calculated by eigendecomposition on MCM have good explanation in the perspectivity of Moran's Coefficient.

2.2.1 Moran's Coefficient and Eigenvector Spatial Filtering

When dealing with spatially correlated data, like an image where pixel correlations depend on their relative positions, Moran's Coefficient (MC) is a widely used measure of spatial autocorrelation. Suppose a vector $x = (x_1, \ldots, x_p)^T$ represents an image with p pixels, and the spatial relationship between each pair of pixels is represented by a matrix C, where C_{ij} indicates the spatial connection between pixels i and j (with diagonal elements equal to zero). According to **?**, the MC can be computed as follows:

$$MC(x) = \frac{p}{\sum_{i=1}^{p}\sum_{j=1}^{p} c_{ij}} \cdot \frac{\sum_{i=1}^{p}(x_i - \bar{x})\left[\sum_{j=1}^{p} c_{ij}(x_j - \bar{x})\right]}{\sum_{i=1}^{p}(x_i - \bar{x})^2}$$

Write Moran's coefficient in the matrix form To express Moran's coefficient (MC) in matrix form, we begin by defining the centering matrix $M = I - \frac{11^T}{p}$, where I is the identity matrix and 1 is a column vector of ones. This matrix M is used to center the vector x, ensuring it has a mean of zero. Specifically, applying M to x results in $[Mx]_i = x_i - \bar{x}$.

Claim: Suppose $\vec{a}$ and $\vec{c}$ are column vectors of length n, and B is an $n \times n$ matrix. Then the expression $\vec{a}^T B\vec{c}$ can be written as:

$$\vec{a}^T B\vec{c} = \sum_i \sum_j a_i B_{ij} c_j$$

Proof. The product $B\vec{c}$ results in a column vector of size n, with each entry i being $\sum_j B_{ij}c_j$. Then, by multiplying with $\vec{a}^T$:

$$\vec{a}^T B\vec{c} = \sum_i a_i \left(\sum_j B_{ij}c_j\right) = \sum_i \sum_j a_i B_{ij} c_j.$$

□

Using this, we can express the terms in the MC in matrix form:

$$\vec{1}^T C\vec{1} = \sum_{i=1}^{p}\sum_{j=1}^{p} 1 \cdot c_{ij} \cdot 1 = \sum_{i=1}^{p}\sum_{j=1}^{p} c_{ij}$$

$$\begin{aligned}(Mx)^T C(Mx) &= \sum_{i=1}^{p}\sum_{j=1}^{p}(x_i - \bar{x})c_{ij}(x_j - \bar{x}) \\ &= \sum_{i=1}^{p}(x_i - \bar{x})\left(\sum_{j=1}^{p} c_{ij}(x_j - \bar{x})\right)\end{aligned}$$

$$
\begin{aligned}
x^T Mx &= x^T MMx \\
&= x^T M^T Mx \\
&= (Mx)^T (Mx) \\
&= \sum_i (x_i - \bar{x})^2
\end{aligned}
$$

Thus, MC can be expressed as:

$$
MC(x) = \frac{p}{\vec{1}^T C \vec{1}} \cdot \frac{x^T MCMx}{x^T Mx}
$$

The matrix form can further be reorganized as:

$$
MC(x) = \frac{(Mx)^T C(Mx)}{\vec{1}^T C \vec{1}} / \frac{(Mx)^T I(Mx)}{\vec{1}^T I \vec{1}}
$$

where the numerator represents the covariance of x along the spatial structure C, and the denominator represents the covariance of x assuming an independence structure I.

[Regarding the expectation of MC: I'm unclear on the distinction between "the expected value when there is no correlation in the assumed model" and "the value when there is no correlation in the input." What does "correlation" refer to in each context? According to Wikipedia, the expectation of MC under the null hypothesis of no spatial autocorrelation is $-1/(n-1)$, and I found a proof here which I don't fully understand. In my understanding, no spatial autocorrelation means all elements of C equal zero (which seems to conflict with the $-1/(n-1)$ expectation).]

Eigendecompose MCM and its key properties **?** demonstrated that, when C is symmetric, the maximum and minimum possible values of the Moran coefficient correspond to the largest and smallest eigenvalues of the matrix MCM. Since the Moran coefficient for an eigenvector is a function of its corresponding eigenvalue (which we will prove later), the eigenvector associated with the largest eigenvalue of MCM yields the highest Moran coefficient. This indicates that this vector captures the strongest spatial autocorrelation among all vectors, given the spatial structure defined by C.

Claim: The centering matrix M is idempotent, meaning $M^2 = M$.

Proof.

$$
\begin{aligned}
M^2 = MM &= \left(I - \frac{11^T}{n}\right)\left(I - \frac{11^T}{n}\right) \\
&= I - \frac{11^T}{n} - \frac{11^T}{n} + \frac{11^T 11^T}{n^2} \\
&= I - \frac{2}{n} 11^T + \frac{n 11^T}{n^2} \\
&= I - \frac{11^T}{n} = M
\end{aligned}
$$

□

We can now consider the eigendecomposition of MCM, which can be written as:

$$MCM = E\Lambda E^T$$

where Λ is a diagonal matrix of eigenvalues, and E contains the corresponding eigenvectors as columns. We define λ_i and v_i to be the i-th eigenvalue and eigenvector, respectively.

Since MCM is symmetric, the eigenvectors are orthogonal (i.e., $EE^T = I$). Additionally, the eigenvectors corresponding to non-zero eigenvalues are orthogonal to the vector of ones, $e_i^T 1 = 0$, where e_i is a such a eigenvector. Because $MCM1 = 0$.

Now, we prove that for any eigenvector v_i, $Mv_i = v_i$.

Proof.

$$\begin{aligned} MCMv_i &= \lambda_i v_i \\ M^2CMv_i &= \lambda_i Mv_i \\ MCMv_i &= \lambda_i Mv_i \\ \lambda_i v_i &= \lambda_i Mv_i \end{aligned}$$

For $\lambda_i \neq 0$, this implies $Mv_i = v_i$. □

With this property established, we can show that the Moran coefficient for each eigenvector is proportional to its corresponding eigenvalue. The Moran coefficient for an eigenvector v_i is given by:

$$MC(v_i) = \frac{n}{1^T C 1} \frac{v_i^T MCMv_i}{v_i^T Mv_i}.$$

Using $MCMv_i = \lambda_i v_i$ and $Mv_i = v_i$, this expression simplifies to:

$$MC(v_i) = \frac{n}{1^T C 1} \lambda_i.$$

Thus, the Moran coefficient is directly proportional to the eigenvalue λ_i, as required.

2.2.2 Whitening Transformations

We now explain how the properties we established about the Moran coefficient and the eigenvectors of MCM can be used to reduce data complexity when fitting image data into models.

A common approach to reducing covariance complexity is the whitening transformation, which transforms a vector of random variables (e.g., an image) to yield a diagonal covariance matrix, meaning no correlation between pixels . Multiple whitening (decorrelation) transformations are possible, but orthogonal transformations are popular due to their preservation of vector length.

If we flatten a 2D image into a 1D column vector x, an orthogonal transformation can be viewed as a rotation or reflection of this vector without changing its length.

Proof. The length of x can be expressed as $x^T x$. Applying an orthogonal matrix E, which satisfies $EE^T = I$, to x, the length of the transformed vector $E^T x$ is $(E^T x)^T (E^T x) = x^T x$. □

A common way to find an orthogonal matrix is through eigendecomposition of a symmetric matrix, which provides a diagonal matrix of eigenvalues and an orthogonal matrix of eigenvectors. Here, we perform eigendecomposition on the centered adjacency matrix MCM as described earlier. This approach is particularly useful for transforming data with spatial autocorrelation due to its interpretability. In this case, the eigenvector associated with the n-th largest eigenvalue captures the n-th strongest spatial autocorrelation direction, as defined by C. This orthogonal transformation can thus be viewed as rotating or reflecting the original data to align with these spatial directions [I'm trying to picture how an image would appear along the direction of strongest spatial autocorrelation]. Since this decomposition is not based on the actual covariance of the dataset, it may not fully eliminate pixel correlation. However, if the adjacency matrix C is reasonable, we expect the transformed data to have sufficiently reduced pixel correlations, improving computational efficiency.

The next question is how to select an appropriate adjacency matrix C for an image. For an image with s pixels, C will be an $s \times s$ matrix, where each element C_{ij} represents the adjacency between pixels i and j. There are several ways to define adjacency between pixels.

One approach is data-independent and unweighted, where only 0 or 1 values are assigned in C: 1 for adjacent pixels and 0 otherwise. A common example is the "2-neighbor adjacency matrix," where two pixels are considered adjacent if they are direct neighbors or share a direct neighbor. All other values, including the diagonal, are set to 0.

Another approach is data-independent but weighted, assigning adjacency values based on the distance between two pixels. Suppose the distance between pixels i and j is d_{ij}, calculated using their Euclidean distance in the image; then the adjacency $C_{ij} = -\exp(d_{ij})$. Both methods define adjacency based on relative pixel positions rather than their actual values.

After choosing an adjacency matrix C, we can apply the eigendecomposition of MCM as we discussed above, and get the E contains the eigenvectors as columns. The eigenvector v_i, corresponding to the i-th largest eigenvalue, represents the i-th strongest spatial autocorrelation that could be captured by a vector among all possible vectors. Suppose X represents a $n \times p$ matrix, representing n images, with each row corresponding to one image follow the spatial adjacency we just specified. Transforming images X to the frequency space is achieved by projecting it onto the matrix of eigenvectors:

$$X_{\text{freq}} = X \cdot E$$

2.3 Models

When we use images as predictors to predict binary outcomes each image belongs to, we can use the Logistic regression. Logistic regression models the probability of an event occurring (e.g. $y = 1$) as a function of the predictors (X). Suppose x_i is a vector of predictor variables for the i-th observation, and $y_i \in \{0, 1\}$ is the binary outcome, and β is the vector of coefficients to be estimated. The model predicts the probability as $p(y_i = 1) =$

$1/(1+\exp(-x_i^T\beta))$. The goal is to estimate β by minimzing the negative log-likelihood of the observed data.

$$y_i \sim \text{Bernoulli}(p_i)$$
$$\text{logit}(p_i) = x_i^T \cdot \beta$$

The goal is find the optimal β such that when $y_i = 1$, the corresponding p_i could be as large as possible, while when $y_i = 0$, $(1-p_i)$ could be as large as possible. We can thus the define the likelihood function as:

$$L = \prod_i p_i^{y_i}(1-p_i)^{1-y_i}$$

The goal is to maximize this function. To simplify the derivatives and improve numerical stability, we usually not directly maximize this function, but instead, minizmize the negative log-likelihood:

$$\begin{aligned} -\ell = -\ln(L) &= -\sum_i^N (y_i \ln(p_i) + (1-y_i)\ln(1-p_i)) \\ &= \sum_i^N \left[\ln\left(1+\exp(x_i^T\beta)\right) + y_i(x_i^T\beta)\right] \end{aligned}$$

When there are a lot of coefficient values equal to zero, in another word, a lot of sparsity in coefficient vector, we add L1 penalty to the negative log-likehood function:

$$\min_\beta -\ell + \sum_j^p \lambda|\beta_j|$$

Here p represents the number of features, and λ is a positive hyperparameter represents the penalty strength we apply for having too many non-zero coefficients. To minimize this function, β_js with small absolute values, representing not important features, will be shrink to zero. The larger λ is, the more coefficients will be shrink to zero. This model is well suited for the scenario that we know there are sparsity in the features.

The goal of this study is to evaluate when we need to do high-dimensional feature selection on imaging data for binary classification problem, whether the spatial correlation between pixels could impede the performance of the LASSO model, and whether fitting LASSO model on the data after whitening transformation we proposed could provide a better performance. This could be achieved by simulating some image data X, and design a coefficient vector β, with some of the values as non-zero values and others as zeros. The pixels corresponding to non-zero coefficients are the ones that will affect the classification of the image. We can calculate the probability of $y = 1$ for each image using $p_i = 1/(1+\exp(-x_{i\cdot}\beta))$, then fit the outcome vector y on X to get the estimated coefficient vector $\hat{\beta}$, see how many of the pixels corresponding to non-zero coefficients are correctly selected.

To compare its performance with whitening transformed data. We choose a spatial adjacency matrix C_{adj} (since we simulated data with the known covariance matrix C_{cov}, we can let they equal, just replace the 1 in the diagonal of latter to 0 for the former), perform the transformation and get X_{freq}. The coefficient vector of the pixel space can be transformed as well with the following equation:

$$X \cdot \beta = X_{\text{freq}} \cdot b = (X \cdot E) \cdot b = X \cdot (E \cdot b)$$

so $\beta = E \cdot b$. We than fit y on X_{freq}, get the estimated coefficients $\hat{b}$, and compare it with the true coefficients b. Comparing the performance of models on pixel space and frequency space can we get an idea of whether whitening transformation improve the model performance in such scenario.

We can also simulate data and coefficients directly on the frequency space, and generate the outcomes. Then the simulated data and coefficients can be transformed back to the pixel space using the inversion of the transformation we provided above. Then the LASSO model will be fit on both X and X_{freq}, and a similar comparison between true coefficients and estimated coefficients could be achieved. This scenario we are aiming to simulate when the sparsity is in the freqeuency space, meaning the image pattern is dominated by a few frequencies.

2.4 Simulations

The first two simulations, one simulate data on the pixel space, and another simulate data on the frequency space, will assume the estimated spatial adjaceny matrix reflects the true covariance in the data. The first simulation we simulate X as a 1000×256 matrix, representing 1000 images, each with 16×6 pixels. They were simulated from a multivariate normal distribution with mean zero and a 256×256 covariance matrix $C_{\text{cov}} - \exp(\text{dist}(x_i, x_j))$, where $\text{dist}(x_i, x_j)$ represents the Euclidean distance between any two pixels i and j in a 2D image. We simulate 1000 such images, flatten each image into a 1D sequence of length 256, and thus create X as a 1000×256 matrix, representing 1000 images, and each pixel will be used as a predictor. β as the coefficient vector, with each value indicating whether the corresponding pixel is important in deciding the classification. We use 0 for the pixels outside of the central 8×8 area, and use a constant non-zero value β for the central area. The value of β will be decided aiming to make the distribution of $p = 1/(1 + \exp(-X\beta))$ evenly distributed around 0 and 1. Since a too large β will make the probabilities either near 0 or near 1, make the task too simple, and a too small β makes all probabilities around 0.5, make the classification too hard. And y is the outcome vector with values of 0 and 1. The outcome vector will be simulated from binomial distributions for each position independently. The probability of $y_i = 1$ will be decided by p_i.

After generating the data, we will simulate the step of assuming a spatial adjacency matrix C_{adj} for the images, performing eigendecomposition on MCM, and use the eigenvectors to transform the original images to the frequency space, which will be called X_{freq}. Simultaneously, we will transform the coefficient vector β into the frequency space as well, labeled as b. This way, we are able to fit Logistic models with L1 penalty on the original images X, and on the transformed data X_{freq} can compare if the coefficient estimation is improved after whitening transformation. In this first scenario, we assume the estimated spatial adjacency matrix C_{adj} correctly reflects the true underlying covariance matrix $C_{textcosv}$, the only difference will be that the diagonal elements in the spatial adjacency matrix are 0s while they are 1s in the covariance matrix.

The above simulation covers the scenario that there is sparsity in the coefficients in the original space. We also want to test how the two models perform when there is sparsity in the coefficients in the frequency space. It represents when the pattern of the image is dominated by a few spatial patterns. We will start from simulating X_{freq} this time. We

expect our whitening transformation could perform similarly to the whitening transformation using eigencomposition of the true covariance matrix. Thus, to simplify the simulation, we randomly draw X_{freq} with 1000 observations from a multivariate normal distribution with mean zero, and a diagonal 256×256 covariance matrix, with the diagonal values decreasing with a constant step size. To simulate the sparsity in the coefficient in the frequency space, we create b as a vector of length 256, with 10% items randomly assigned a non-zero constant value b, while all other values equal 0. Similar to the previous simulation, the value of b will be decided aiming to make the probability of $y = 1$ evenly along the range from 0 to 1. Since the transformation between X and X_{freq} solely depends on our assumption of the spatial adjacency matrix C_{adj}, not the actual covariance C_{cov}, here we still use the same matrix of eigenvectors to transform X_{freq} to its original space: $X = X_{\text{freq}} \cdot E^T$, and transform b back to the coefficients in the original space β.

Each simulation will be repeated for 500 times.

2.5 Analyses

We will introduce what analyses we performed on the simulated data, which include visualization to check the simulatation quality, and then the model metrics of the LASSO models.

2.5.1 Group mean Difference

To visualize the performance of the data we simulated, in other words, whether the β value we chose for the pixel space, or the b value we chose for the frequency space could properly reflect the criteria we imaged for classification, we calculate the group mean differences for each simulation. Since we performed 500 iterations of simulation, only the data from the first iteration will be used for visualization. For data in the pixel space, we calculated the average value at each pixel for all images belong to $y = 1$, and similarly for all images belong to $y = 0$. Then the difference between two groups will be subtracted and visualized by a heatmap. If the simulation works well, since we set β to have positive non-zero values only for the pixels corresponding to the central 8×8 area, images with larger values in that area will have a larger probability of getting the corresponding y equals 1. Thus the group mean difference heatmap should also have positive values around that area, while the difference in the other areas should be around 0.

Similarly, we calculated the group mean difference between two classes in the frequency space. Instead of showing them as 2D images (as it makes no sense to frequencies), we show the group mean difference as scatterplot with x-axis as the frequencies and y-axis as the corresponding group mean difference. When the covariance between pixels depends on their spatial relationship, and the estimated spatial adjacency matrix correctly reflect such structure, then the eigenvectors of MCM corresponding to the largest eigenvalue should reflect the direction that with the largest spatial variance. Thus the frequencies transposed by those eigenvectors should have larger variance, and the group mean difference at those frequencies should be more obvious than the others, either negative or positive (depends on the value in b, which is transposed from β.

The same visualization was generated for the second simulation as well. Since this time we simulated data in the frequency space with a decreasing diagonal values on the covariance matrix, we expect to see frequencies with a larger diagonal value will have more obvious group mean difference than the frequencies with a smaller diagonal value. And we don't expect recoganizable patterns in the heatmap showing in the pixel space. The group mean difference in the pixel space should be dominated by the values in the transposed β.

2.5.2 LASSO cross-validation

To compare the performance of LASSO in both the pixel space and frequency space, we fit two models: one using covariates in the pixel space and another using covariates in the frequency space. The optimal regularization parameter λ is selected via cross-validation, using the binomial deviance as the performance metric. The dataset is split into training (80%) and test (20%) sets, and the cross-validation is performed using 10 folds.

Two values of λ are considered:

- `lambda.min`, which minimizes the cross-validated error.
- `lambda.1se`, the largest λ within one standard error of the minimum.

To balance between achieving a log-likelihood as large as possible, and have as few non-zero coefficients as possible, we need to choose an optimal λ. This step is usually produced by doing cross-validation. Using 5-fold cross-validation as an example, the dataset will be randomly split into 5 batches, with almost identity number of samples in each. The model we specified will be fit on 4-folds, and then predict on the other fold. Some performance metric will be used to evaluate the model performance on that fold. This process will be repeated for 4 more times, using each of the other 4 folds as the evaluation set. The model performance will be averaged across the 5 iterations. We do this process under several options of λ values, and the λ provides the best model performance will be used. WE split the dataset into 80% training set (used for cross-validation and select optimal parameter), and the chosen parameter will be used to fit model and evaluate on the 20% test set.

2.5.3 LASSO metrics

There are several metrics that could used to evaluate the model performance on the test set. After we get the predicted probabilities of $y = 1$, the simplest way to decide classes is to assign observations with $p_i > 0.5$ as $y_i = 1$, and others as $y_i = 0$. Then we can compare the predicted classes with the actual classes to calculate the accuracy. Another commonly used measurement is called Area Under Curve (AUC). It measures the area under the receiver operating characteristic (ROC curve), which marks the false positive rate in the x-axis along with the true positive rate on the y-axis as the threshold changes. A larger AUC value means the classifier could achieve a relative high true positive rate while maintain a relative low false positive rate. The last metric we consider to use is the p-values for covaraites. Though Logistic regression with L1 penalty does not provide the calculation of p-values naturally, we can use the `hdi` package to calculate it [Need to provide more details].

We will calculated the estimated coefficients for all models and repeated it for 500 times. When simulating data in the pixel space, we will transform the data and coefficients simulated

into the frequency space, and fit model using the data on both spaces. The true coefficients in both space, as well as the mean estimated coefficients across iterations, will be visualized. For coefficients in the pixel space, we will use the 16×16 heatmap, while for coefficients in the frequency space, we will order them by the corresponding of eigenvalues, from smallest to the largested, using a scatterplot. Both the true coefficient and estimated coefficients will be transformed into another space and visualize in the same way. For example, for Simulation 1, we simulate data in the pixel space, then we will show the true coefficient used and estimated coefficients by fitting directly on data in the pixel space, estimated coefficients by fitting data in the frequency space and transposed the estimated coefficients back from the frequency space. We will also show the transposed true coefficients in the frequency space, the transposed estimated coefficients by fitting model in the pixel space, the estimated coefficients by fitting model in the frequency space. Since we tried two options of λ, this will be 10 images in total. And the same number of images will be show for simulation 2.

We will also report the p-values computed using the `hdi` package [need to provide further details on how the package computes p-values]. For each simulation, we will report the percantage of $p < 0.05$ for each predictor when fitting model on the pixel space as well as fitting model on the frequency space. For the percentages for model in the pixel space, they will be shown in a similar way as we talked above, as a heatmap; for percentages in the frequency space, we will also use a similarly scatterplot.

3 Results

3.1 Effect Size Determination

In Simulation 1, we evaluated the distribution of the success probability $\mathbf{p}$ at different non-zero values of β (0.01, 0.05, 0.1, 0.2, and 1). As shown in Figure 1, a value of 0.1 produced the most uniform distribution of $\mathbf{p}$, making it the optimal choice for model fitting in this scenario.

Similarly, in Simulation 2, we assessed the distribution of $\mathbf{p}$ at various non-zero values for $\mathbf{b}$ (0.1, 0.15, 0.2, 0.25, and 0.3). As shown in Figure 2, the value of 0.2 resulted in the most uniform distribution of $\mathbf{p}$, making it the best option for this simulation.

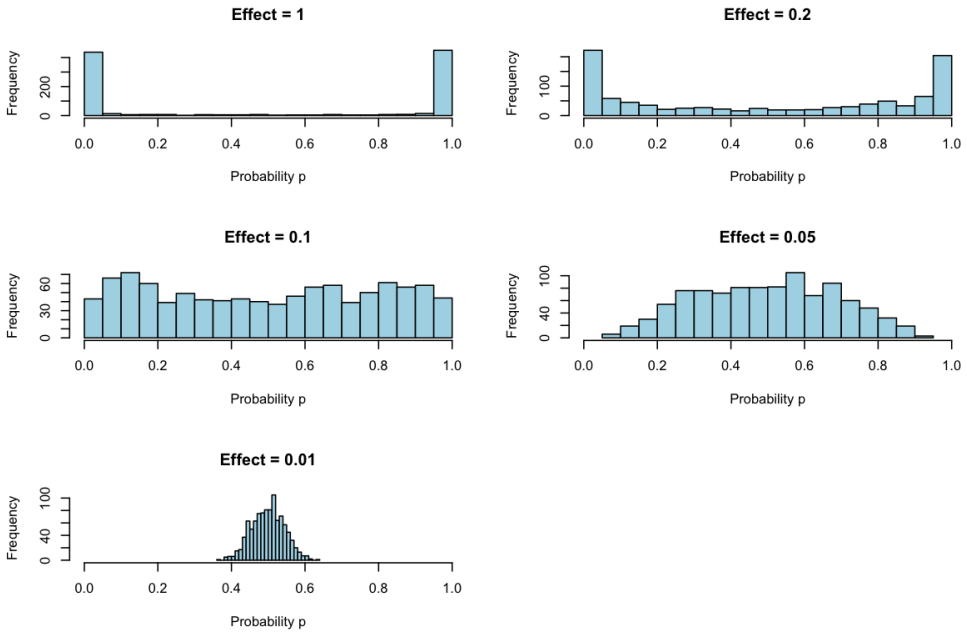

Figure 1: Distribution of success probability $\mathbf{p}$ at different non-zero values in Simulation 1.

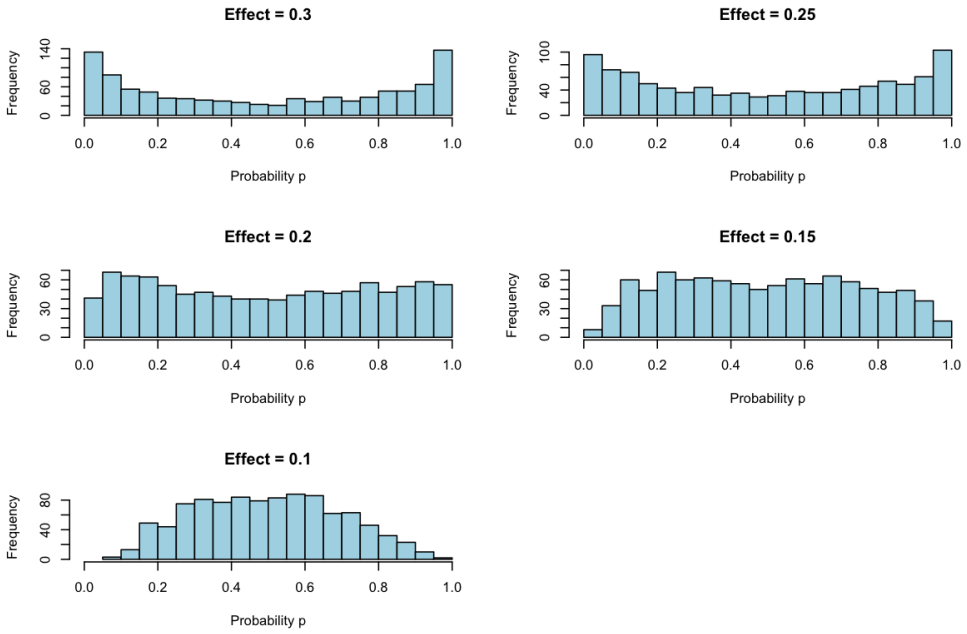

Figure 2: Distribution of success probability $\mathbf{p}$ at different non-zero values in Simulation 2.

Group Mean Difference

In this subsection, we examine the group mean differences in covariate values between instances where $y = 1$ and $y = 0$ for both Simulation 1 and Simulation 2.

Figure 3 presents the group mean differences for Simulation 1, with the heatmap on the

left showing that regions corresponding to non-zero coefficients in β exhibit larger mean differences between $y = 1$ and $y = 0$, as larger covariate values in these locations have higher probabilities of being assigned to $y = 1$. The scatterplot on the right displays the group mean differences in the frequency domain, where each point represents a frequency component; frequencies associated with larger eigenvalues tend to have larger mean differences. Figure 4 shows the actual coefficients used in Simulation 1, where non-zero coefficients in β are localized to specific pixels, corresponding to the areas with larger mean differences in the group comparison.

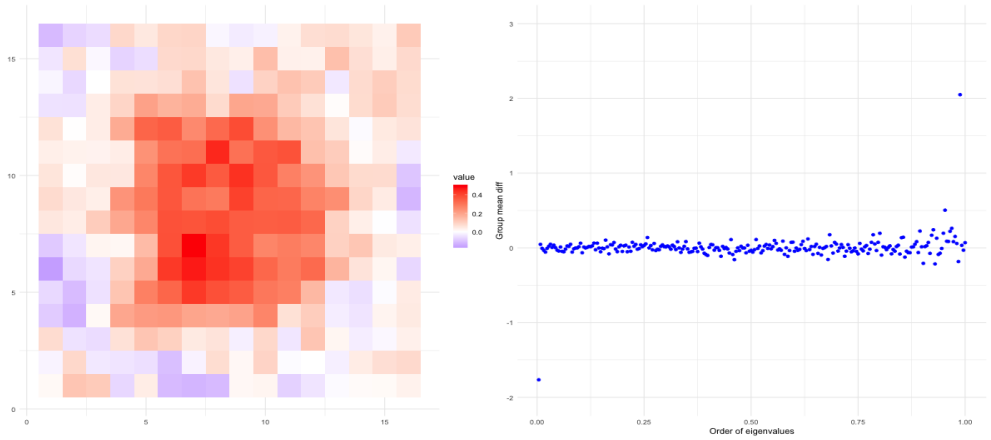

Figure 3: Group mean difference in covariate values between instances where $y = 1$ and $y = 0$ in Simulation 1, shown for both the pixel space (left) and frequency space (right).

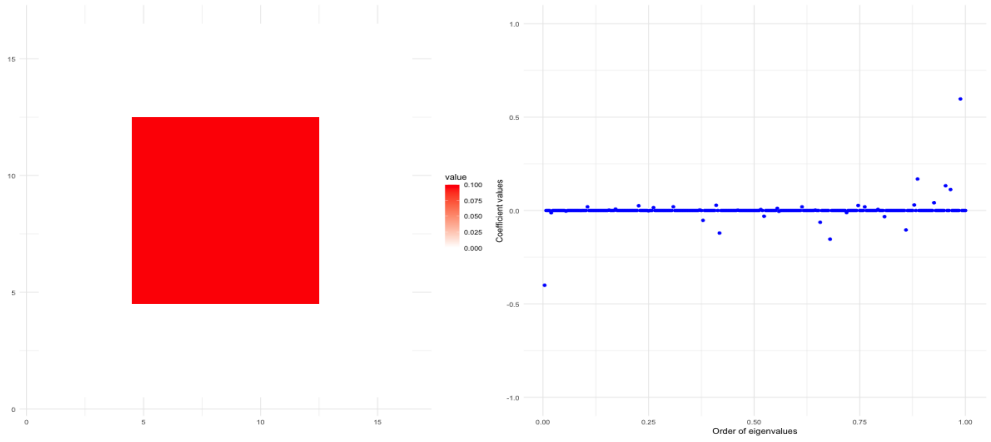

Figure 4: Actual coefficients in Simulation 1 for the pixel space (left) and frequency space (right).

Figure 5 shows the group mean differences for Simulation 2, while Figure 6 displays the actual coefficients. The non-zero coefficients in $\mathbf{b}$ are uniformly set to 0.2. However, the scatterplot in the frequency space does not clearly highlight the non-zero components, with increasing variance observed for larger eigenvalues. This variance pattern is consistent with the diagonal covariance matrix used in the simulation. The difficulty in identifying the

non-zero components suggests that the effect size may be too small relative to the variance, making detection challenging. [Further adjustments to either the effect size or the covariance matrix could improve the detectability of these non-zero coefficients in future analyses.]

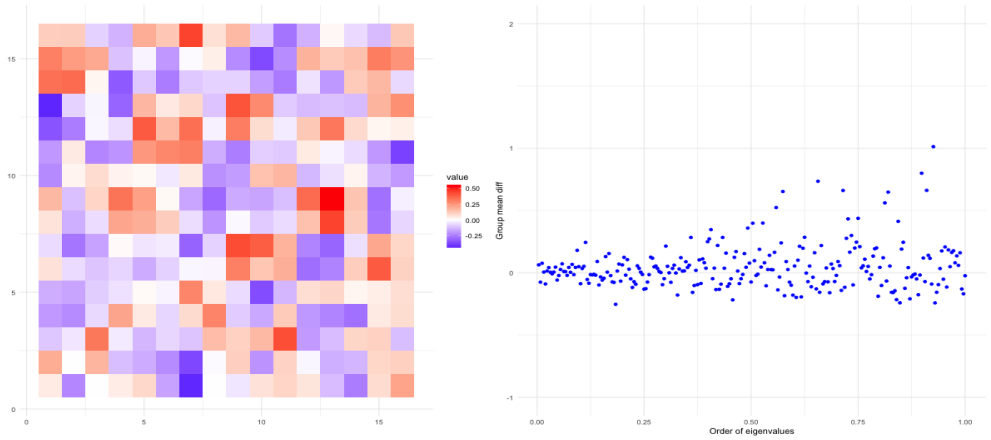

Figure 5: Group mean difference in covariate values between instances where $y = 1$ and $y = 0$ in Simulation 2.

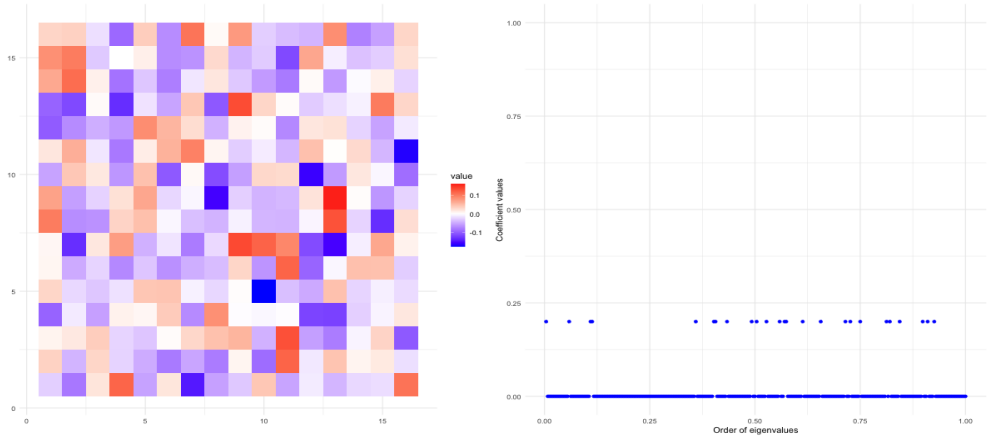

Figure 6: Actual coefficients in Simulation 2 for the pixel space (left) and frequency space (right).

AUC and Accuracy

Table 1 summarizes the average AUCs and accuracies over 500 iterations. In both Simulation 1 (pixel space sparsity) and Simulation 2 (frequency space sparsity), models fitted in the frequency space consistently outperformed those fitted in the pixel space. For example, in Simulation 1, using `lambda.min` as the regularization parameter, models trained with pixel space covariates achieved an AUC of 0.803 (SE = 0.031) and an accuracy of 72.6% (SE = 0.032). In contrast, models trained with frequency space covariates produced a slightly higher AUC of 0.826 (SE = 0.028) and a higher accuracy of 74.5% (SE = 0.030). A similar trend

was observed in Simulation 2, with frequency space models showing superior performance regardless of the regularization parameter used.

Table 1: Comparison of AUC and accuracy between models fitted in the pixel space and frequency space across 500 iterations for Simulation 1 and Simulation 2.

Simulation	**Model in Pixel Space**		**Model in Frequency Space**	
	AUC (SE)	**Accuracy (SE)**	**AUC (SE)**	**Accuracy (SE)**
Simulation 1				
`lambda.min`	0.803 (0.031)	0.726 (0.032)	0.826 (0.028)	0.745 (0.030)
`lambda.1se`	0.800 (0.032)	0.722 (0.032)	0.826 (0.029)	0.745 (0.031)
Simulation 2				
`lambda.min`	0.755 (0.036)	0.684 (0.034)	0.812 (0.030)	0.732 (0.032)
`lambda.1se`	0.735 (0.039)	0.669 (0.038)	0.812 (0.031)	0.732 (0.032)

Coefficients Estimation

Figure 8 presents the mean estimated b values plotted against the order of eigenvalues. The order of eigenvalues are calculated the same way as above. For Simulation 1, `lambda.1se` shrinks the estimated coefficients more than `lambda.min`, as it provides a larger panelty on it. For Simulation 2, even though it is not obvious, I feel the estimated values has an increase trend as the eigenvalues increase. [Still, I am wondering whether this is related to the covariance matrix, the decreased diagonal values, consider math proof?].

The mean estimated coefficients across iterations were calculated, and Figure 7 displays the mean estimated β values. Two key observations can be made: (1) There is no significant difference in the estimated coefficients when using `lambda.min` versus `lambda.1se`, and (2) the estimated values align well with the actual values, indicating that the model is accurately identifying the relevant features.

Figure 8 shows the mean estimated $\mathbf{b}$ values plotted against the order of eigenvalues. The eigenvalue ordering is consistent with earlier calculations. In Simulation 1, `lambda.1se` applies a stronger regularization penalty, shrinking the estimated coefficients more than `lambda.min`. For Simulation 2, although the trend is less clear, there seems to be an upward trend in the estimated values as the eigenvalues increase. This trend may be related to the structure of the covariance matrix, specifically its decreasing diagonal values [consider math proof?].

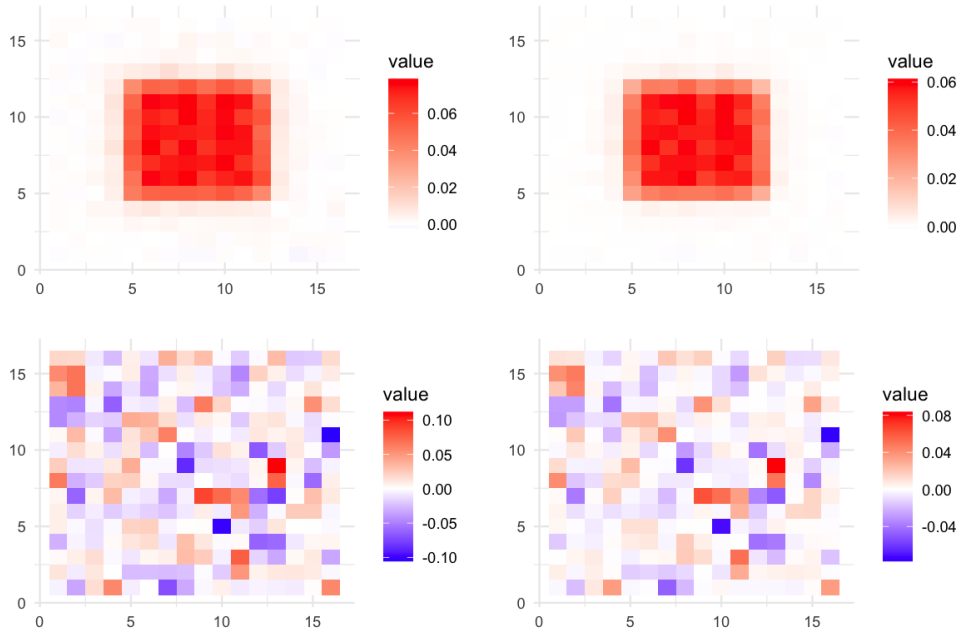

Figure 7: Mean estimated β values across simulations, with models fitted using `lambda.min` (left) and `lambda.1se` (right). The top row shows results for Simulation 1, while the bottom row shows results for Simulation 2.

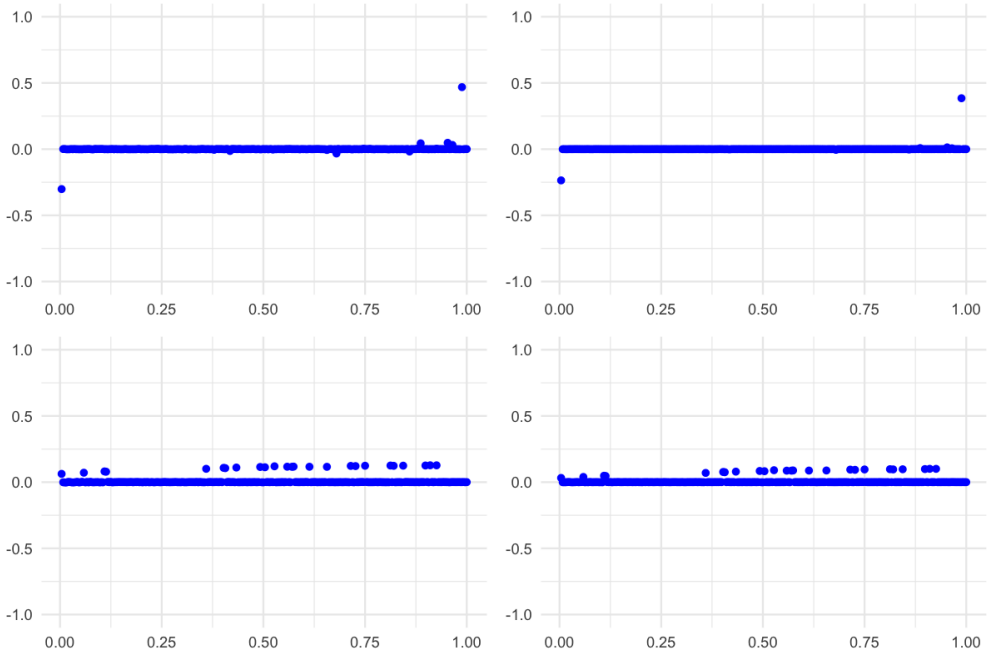

Figure 8: Mean estimated b values across simulations, plotted against ordered eigenvalues. Models fitted using `lambda.min` are on the left and models fitted with `lambda.1se` on the right. The top row shows results for Simulation 1, while the bottom row shows results for Simulation 2.

Significant P-values

It is interesting to observe that, although the heatmap for significance of β in Simulation 1 follows the pattern of the actual non-zero values, the percentage of significance is relatively low (Figure 9 left). In contrast, the non-zero values of $\mathbf{b}$ (Figure 10 left) show a much higher percentage of significance, reaching as high as 100% across iterations.

Another observation is that, although the non-zero effect size for $\mathbf{b}$ is constant in Simulation 2, the percentage of significant p-values increases as the eigenvalues grow (Figure 10 right). [I am considering creating a plot to visualize the actual β values in Simulation 2 and the actual b values in Simulation 1, where the non-zero values vary, and compare them with the corresponding percentage of significant p-values. The goal is to examine whether the size of the actual non-zero values correlates with the percentage of significance. I suspect that a larger absolute effect size should result in a higher percentage of significance, but this doesn't seem to be the case for b in Simulation 2, so I want to investigate other factors as well.]

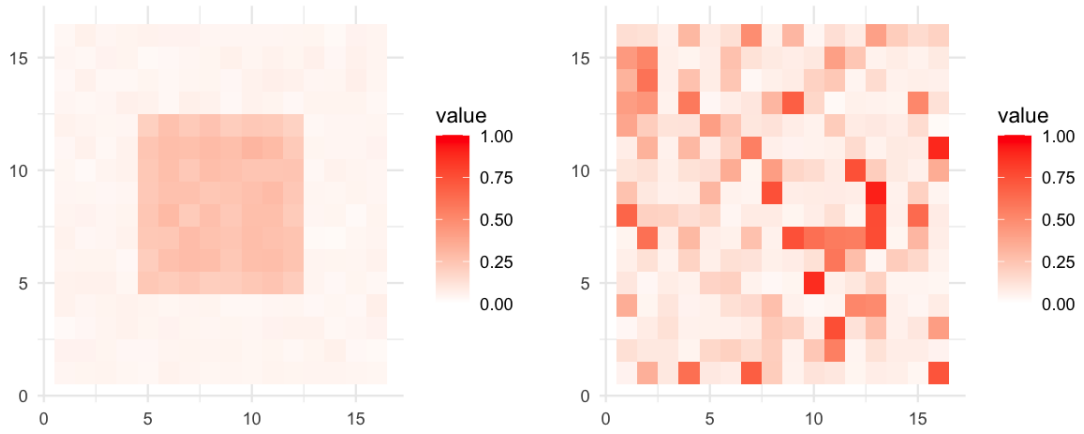

Figure 9: Percentage of significant p-values for elements of β when fitting models in the pixel space in Simulation 1 (left) and Simulation 2 (right).

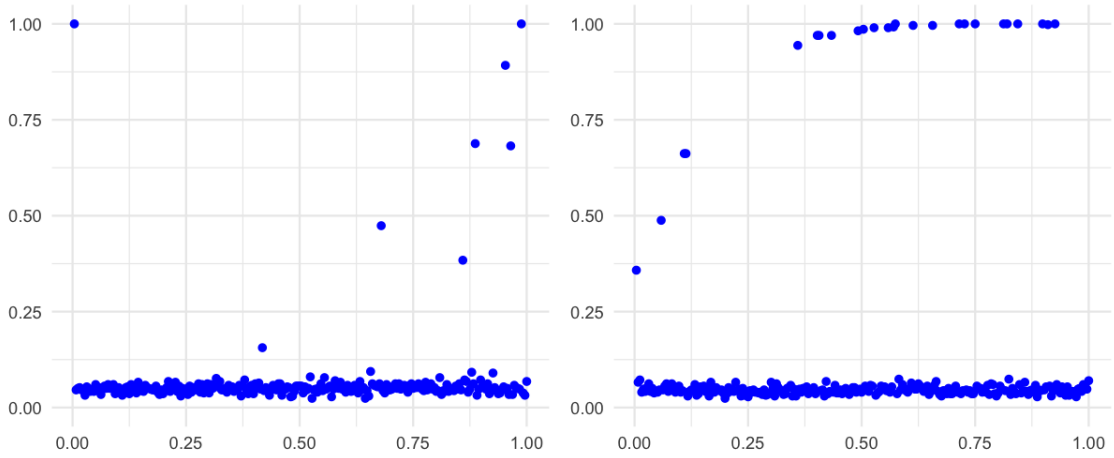

Figure 10: Percentage of significant p-values for elements of b across ordered eigenvalues in both simulations.

Future Work

- Adding details about how `hdi` package calculated p-values and why my permutation test didn't work.
- Increase b effect size (how to keep p evenly distributed in the same time?) see whether the pattern of coefficient estimates disappear or relieve.
- What is the next step in higher level?

References

Peter de Jong, C Sprenger, and Frans van Veen. On extreme values of moran's i and geary's c. *Geographical Analysis*, 16(1):17–24, 1984.

Daniel Griffith and Yongwan Chun. Spatial autocorrelation and spatial filtering. *Handbook of regional science*, pages 1477–1507, 2014.