Indirect High-Dimensional Linear Regression *

Matt Galloway *University of Minnesota*

This study evaluates the performance of a class of *indirect* regression coefficient estimators designed to perform well in high-dimension regression settings. We demonstrate several simulation studies and compare this class of estimators to other standard methods. This work is heavily based off of Cook, Forzani, and Rothman (2013) and Molstad and Rothman (2016).

Keywords: high dimenison, big data, abundant, graphical lasso, indirect

Introduction

Consider the classical linear regresion model for a univariate response:

$$Y = \mu_y + \beta^T(X - \mu_x) + \epsilon, \tag{1}$$

where $Y \in \mathbb{R}$, $X \in \mathbb{R}^p$, and $\beta \in \mathbb{R}^p$ is a vector of unknown regression coefficients. Unlike a controlled experiment in which we take our explanatory variable X to be fixed, in this project we will assume X is random. Let $X \sim N_p(\mu_x, \Sigma_{xx})$ (take $\Sigma_{xx} > 0$) and $\epsilon \sim N(0, \sigma^2_{y|x})$ so that $(X_1, Y_1), ..., (X_n, Y_n)$ are realizations of the joint multivariate normal distribution (X, Y):

$$(X^T, Y)^T \sim N_{p+1}\left(\begin{pmatrix}\mu_x \\ \mu_y\end{pmatrix}, \begin{pmatrix}\Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{xy}^T & \sigma_y^2\end{pmatrix}\right),$$

The goal throughout this report is to estimate the unknown regression coefficient vector $\beta = \Sigma_{xx}^{-1}\Sigma_{xy}$. When $n > p$, it is standard practice to use the ordinary least squares estimator

$$\hat{\beta}^{OLS} = (\mathbb{X}^T\mathbb{X})^{-1}\mathbb{X}^T\mathbb{Y} = \hat{\Sigma}_{xx}^{-1}\hat{\Sigma}_{xy}, \tag{2}$$

where $\mathbb{X} \in \mathbb{R}^{n\times p}$ has ith row $X_i - \mu_x$, $\mathbb{Y} \in \mathbb{R}^n$ has elements $Y_i - \mu_y$, and $\hat{\Sigma}_{xx}$ and $\hat{\Sigma}_{xy}$ are the sample covariance of X and sample covariance of X and Y, respectively. This estimator is also the maximum likelihood estimator and holds whether we take X to be fixed *or* random.

When $n \leq p$ - the so-called, high dimensional setting - the OLS estimator is no longer identifiable. This is due to the fact that the matrix $\mathbb{X}^T\mathbb{X}$ is rank deficient causing $\hat{\Sigma}_{xx}$ to be singular. To address this issue, shrinkage estimators of β have been proposed that work by penalizing the log-likelihood used to calculate (2) and, in effect, pushing the eigen values of $\hat{\Sigma}_{xx}$ further from zero (making $\hat{\Sigma}_{xx}$ non-singular). A few of the most popular

*December 12, 2017. **Contact**: gall0441@umn.edu.

shrinkage estimators are presented below:

Ridge Penalty:

$$\hat{\beta}^R = \arg\min_{\beta} -l(\beta) + \frac{\lambda}{2}\sum_{j=1}^{p}\beta_j^2,$$

Lasso Penalty:

$$\hat{\beta}^L = \arg\min_{\beta} -l(\beta) + \lambda\sum_{j=1}^{p}|\beta_j|,$$

where $l(\beta)$ is the log-likelihood and λ is a tuning parameter. These methods have proved largely successful by making specific assumptions about β. For instance, we might assume sparsity in our coefficient vector. This assumption would likely favor a lasso penalty due to its non-differentiability that causes a number of entries in β to be equal to zero. In other situations we might assume a non-sparse solution in which case a ridge penalty would be more appropriate. However, these estimators lack the ability to fully exploit the random component in X unique to our design. In the following section we explore *indirect regression estimators* that seek to leverage the joint distribution between the explanatory variables and the response for potential gain.

Indirect Regression Coefficient Estimators

Recall that we assume $X \sim N_p(\mu_x, \Sigma_{xx})$ and $\epsilon \sim N(0, \sigma^2_{y|x})$. This assumption in conjunction with (1) allows us to define the following conditional distributions:

Forward Regression:

$$Y|X = x \sim N\left(\mu_y + \beta^T(X - \mu_x), \sigma^2_{y|x}\right)$$

Inverse Regression:

$$X|Y = y \sim N_p\left(\mu_x + \alpha^T(Y - \mu_y), \Delta\right)$$

where $\beta = \Sigma_{xx}^{-1}\Sigma_{xy}$, $\alpha = \sigma_y^{-2}\Sigma_{xy}^T$, $\sigma^2_{y|x} = \sigma_y^2 - \beta^T\Sigma_{xx}\beta$ and $Var(X|Y) \equiv \Delta = \Sigma_{xx} - \Sigma_{xy}\Sigma_{xy}^T/\sigma_y^2$. We will use the last equivalence to construct our indirect regression coefficient estimator. Using the Woodbury Identity, it is relatively straight forward (details in the appendix) to show that

$$\Sigma_{xx}^{-1} = \Delta^{-1} - \frac{\Delta^{-1}\Sigma_{xy}\Sigma_{xy}^T\Delta^{-1}}{\sigma_y^2 + \Sigma_{xy}^T\Delta^{-1}\Sigma_{xy}^T}$$

We can then plug this formulation into β to show that

$$\beta = \Sigma_{xx}^{-1}\Sigma_{xy} = \Delta^{-1}\Sigma_{xy}\left(1 + \Sigma_{xy}^{T}\Delta^{-1}\Sigma_{xy}/\sigma_y^2\right)^{-1}$$

(assuming a positive definite covariance matrix of (X, Y)) [1]. This work is closely related to Molstad & Rothman (2016), however, they extend this class of estimators to the case when the response is multivariate.

By exploiting the joint distribution of (X, Y), it is clear that β can be expressed as a function of Δ^{-1}, Σ_{xy}, and σ_y^2 - no longer requiring Σ_{xx} to be invertible. Of course not all issues are solved using this re-formulation: like Σ_{xx} in high-dimensions, the maximum likelihood estimator for $\hat{\Delta}$ will be singular when $n \leq p$.

$$\hat{\Delta}_{MLE} = \hat{\Sigma}_{x|y} = (\mathbb{X} - \mathbb{Y}\hat{\alpha})^T(\mathbb{X} - \mathbb{Y}\hat{\alpha})/n$$

where $\hat{\alpha} = (\mathbb{Y}^T\mathbb{Y})^{-1}\mathbb{Y}^T\mathbb{X}$ is the MLE of the coefficient vector for the regression of X on Y. Similar to previous methods, this issue will be addressed by introducing shrinkage estimators. However, instead of shrinking β we propose shrinking the precision matrix Δ^{-1}. This serves two purposes. The first is that instead of making assumptions about β we can instead make assumptions about the structure of Δ^{-1}. The second purpose is the expectation that shrinking Δ^{-1} will greatly improve our estimates when $p \gg n$. In this setting, a large number of unknowns and low (relative) sample size may result in poor estimates.

Shrinkage Estimators

The shrinkage estimators proposed in this project are analogous to the forward regression case where we use the negative log-likelihood as our loss function. We take $\Theta \equiv \Delta^{-1}$ for convenience.

Ridge Penalty:

$$\hat{\Theta}_{\lambda}^{IR} = \arg\min_{\Theta \in \mathbb{S}_+^p} \left\{ tr(\Theta\hat{\Sigma}_{x|y}) - log\,|\Theta| + \frac{\lambda}{2}\|\Theta\|_F^2 \right\} \tag{3}$$

Lasso Penalty:

$$\hat{\Theta}_{\lambda}^{IL} = \arg\min_{\Theta \in \mathbb{S}_+^p} \left\{ tr(\Theta\hat{\Sigma}_{x|y}) - log\,|\Theta| + \lambda \sum_{i \neq j} |\theta_{ij}| \right\} \tag{4}$$

The first estimator (3) uses a ridge-type penalty in which we penalize the squared sum of all the entries in Θ using the Frobenius norm. Similar to the forward regression, this estimator has a closed form solution and thus can be computed with minimal cost. We show

(in the appendix) that if we decompose $\hat{\Sigma}_{x|y} = VQV^T$ using the spectral decomposition then

$$\hat{\Theta}_\lambda^{IR} = \begin{Bmatrix} \frac{1}{2\lambda} V \left[-Q + (Q^2 + 4\lambda I_p)^{1/2} \right] V^T & \text{if } \lambda > 0 \\ \hat{\Sigma}_{x|y}^{-1} & \text{if } \hat{\Sigma}_{x|y}^{-1} \text{ exists and } \lambda = 0 \end{Bmatrix} \tag{5}$$

The second estimator (4) uses a lasso-type penalty by summing the absolute values of the off-diagonal entries. This penalty encourages sparse solutions of Θ. In our multivariate normal setting, a zero in the precision matrix Θ implies that two elements are conditionally uncorrelated give the other elements - but may still be marginally correlated. We will use the popular *graphical lasso* algorithm for calculation (not presented here).

Using these shrinkage estimators, our proposed indirect estimators are the following:

Indirect Ridge:

$$\hat{\beta}^{IR} = \hat{\Theta}_\lambda^{IR} \hat{\Sigma}_{xy} \left(1 + \hat{\Sigma}_{xy}^T \hat{\Theta}_\lambda^{IR} \hat{\Sigma}_{xy} / \hat{\sigma}_y^2 \right)^{-1} \tag{6}$$

Indirect Lasso:

$$\hat{\beta}^{IL} = \hat{\Theta}_\lambda^{IL} \hat{\Sigma}_{xy} \left(1 + \hat{\Sigma}_{xy}^T \hat{\Theta}_\lambda^{IL} \hat{\Sigma}_{xy} / \hat{\sigma}_y^2 \right)^{-1} \tag{7}$$

where, unlike Molstad & Rothman (2016), $\hat{\Sigma}_{xy}$ and $\hat{\sigma}_y^2$ are the sample estimates (using denominator n). In their previous work, they proposed shrinkage estimators for those as well. We compare the performance of both of these estimators to their forward regression counter-parts in the section that follows.

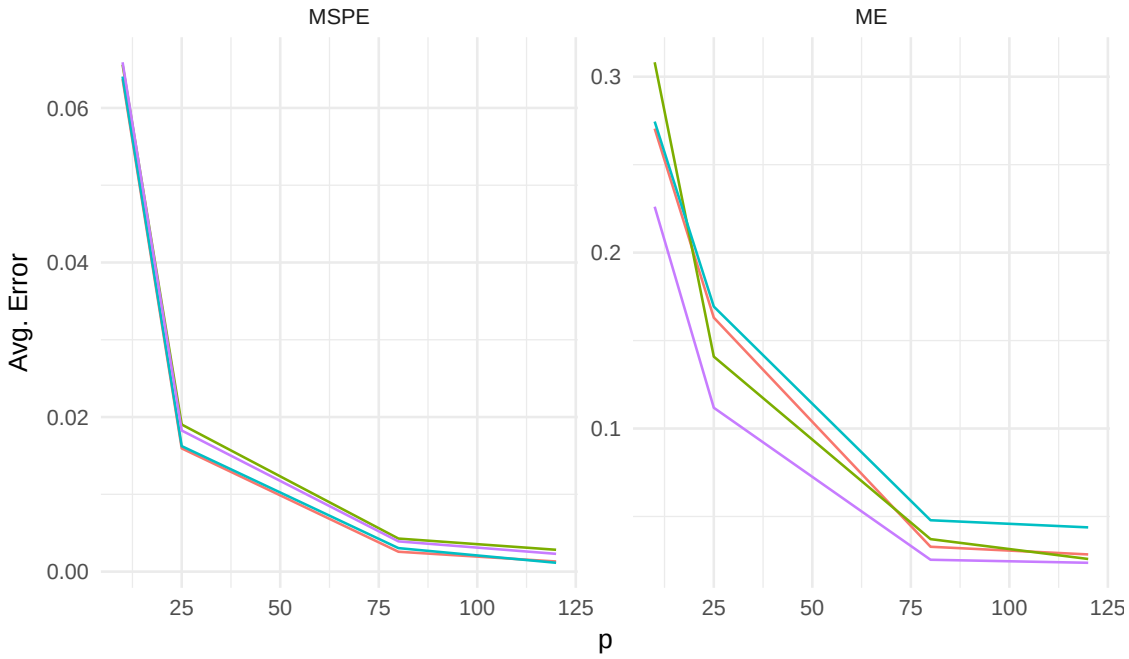
Simulations

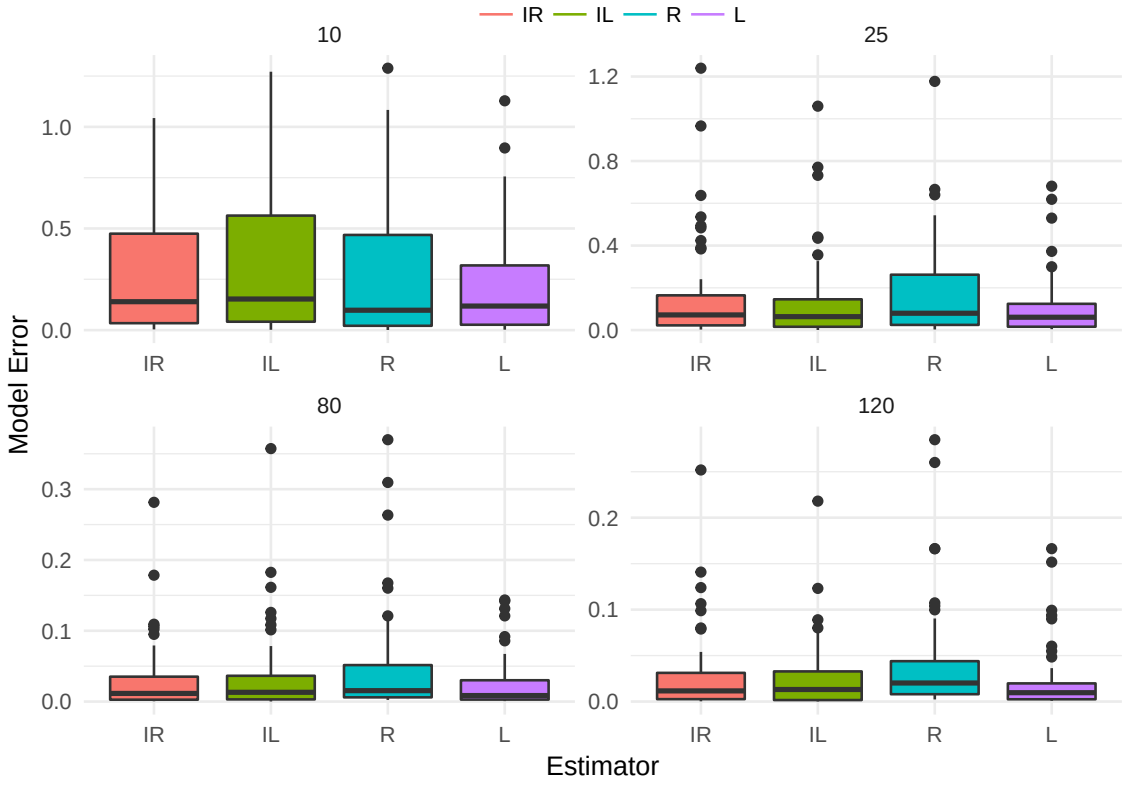
We generate realizations of $n = 100$ independent copies of $(X^T, Y)^T$ where $Y \sim N(0, \sigma_y^2)$ and $X|Y = y \sim N_p(\alpha^T Y, \Delta)$. Following Molstad & Rothman, the inverse regression coefficient vector α was chosen so that $\alpha = Z \circ B$ where Z is a vector of standard normal entries and B is a vector of values drawn from a Bernoulli distribution with probability b. In addition, Δ is calculated so that all of the off-diagonal entries are equal to 0.9 and 1 otherwise. A few of the parameters have multiple candidate values: $p \in (10, 25, 80, 120)$, $\sigma_y^2 \in (0.3, 0.7)$, and $b \in (0.3, 0.9)$. The simulations are constructed so that each scenario is replicated a total of 50 times.

$$ME(\beta, \hat{\beta}) = tr\left\{ (\hat{\beta} - \beta)^T \Sigma_{xx} (\hat{\beta} - \beta) \right\}$$

$$MSPE(\hat{\beta}) = \frac{1}{n} \|Y - X\hat{\beta}\|^2$$

For each replication, the model error (ME) and mean squared prediction error (MSPE) are recorded for each of the estimators $\hat{\beta}^{IR}, \hat{\beta}^{IL}, \hat{\beta}^{R}, \hat{\beta}^{L}$. The tuning parameters λ were chosen from the set $(10^{-4}, 10^{-3.5}, ..., 10^{7.5}, 10^{8})$ using three-fold crossvalidation.



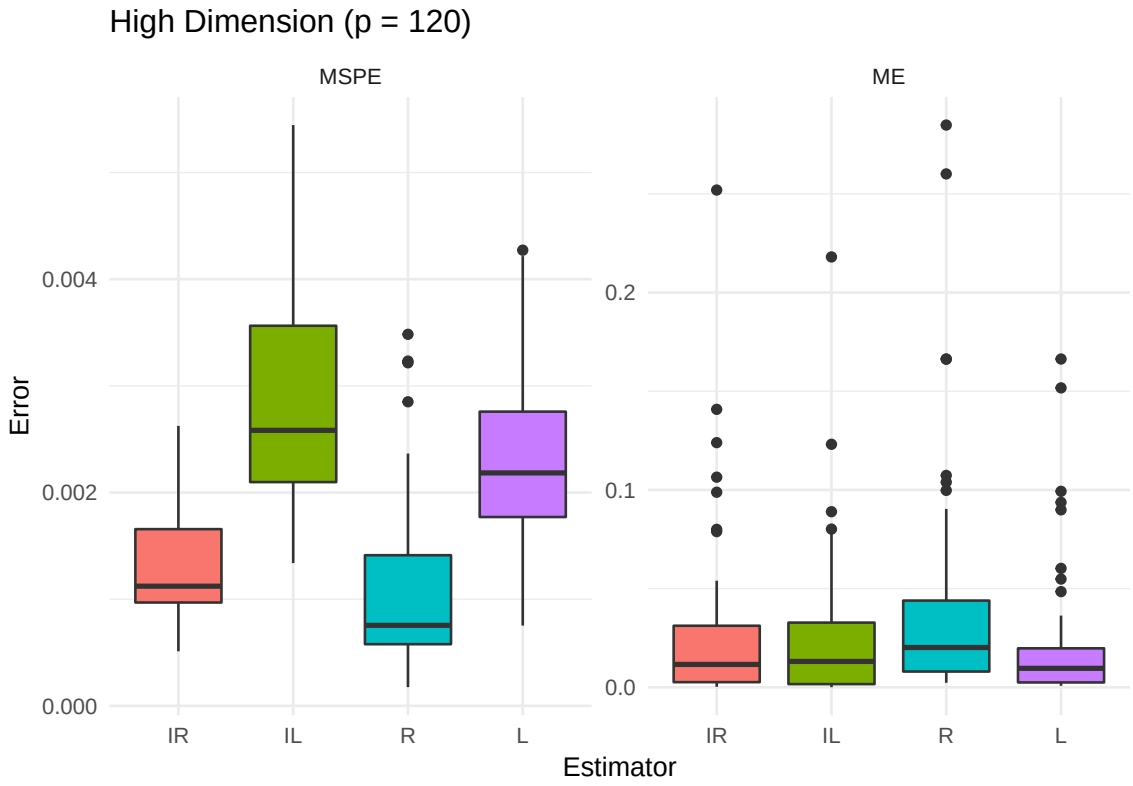


We can see from the simulations that performances under the MSPE metric were all relatively comparable. Each estimator's performance increased as the sample size increased and appeared to plateau as p exceeded 100. This general trend was also true for the model error except for the forward regression lasso estimator, $\hat{\beta}^L$, noticeably outperforming the others. The worst estimator when $p = 10$ was the indirect lasso estimator but it recovered by being the second best when $p = 120$ ($\sigma_y^2 = 0.3, b = 0.3$).

Table 1: Model error by estimator and dimension.

	IR		IL		R		L	
P	mean	sd	mean	sd	mean	sd	mean	sd
10	**0.270**	0.302	**0.308**	0.333	**0.274**	0.344	**0.226**	0.270
25	**0.163**	0.252	**0.141**	0.216	**0.169**	0.229	**0.112**	0.154
80	**0.033**	0.052	**0.037**	0.063	**0.048**	0.079	**0.025**	0.038
120	**0.028**	0.047	**0.026**	0.039	**0.044**	0.061	**0.024**	0.037
All	**0.124**	0.222	**0.128**	0.230	**0.134**	0.232	**0.097**	0.177

Focusing in on the high dimension case, we note that the ridge estimators appear to be the clear winners if we are focused on MSPE. Not only did their average error outperform the other methods but their standard errors appear to be smaller as well. Interestingly, the forward regression ridge estimator, $\hat{\beta}^R$, was the worst in terms of model error.



As we vary the sparsity of our inverse regression coefficient vector, α, and the variance of Y, we can see that the indirect lasso estimator performs best when the sparsity level is low ($b = 0.9$) and the variance of Y is low ($\sigma_y^2 = 0.3$) - though the difference in performance is minimal when p is large.

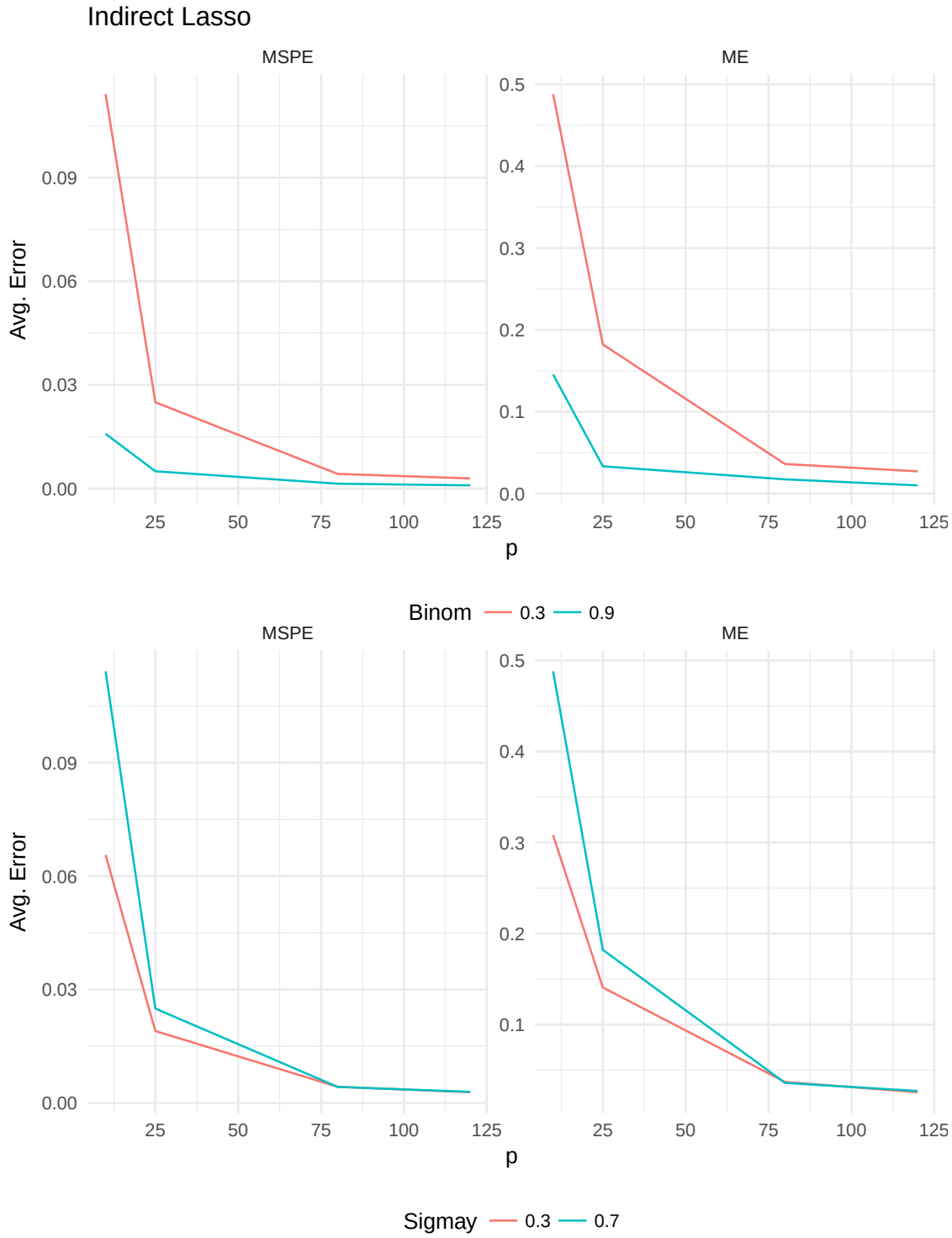


Table 2: Model error for indirect lasso by b and dimension.

	b0.3		b0.9	
P	mean	sd	mean	sd
10	**0.488**	0.736	**0.146**	0.183
25	**0.182**	0.296	**0.033**	0.049
80	**0.036**	0.045	**0.017**	0.020
120	**0.027**	0.043	**0.010**	0.011

Discussion

Indirect regression estimators are yet another tool that statisticians can use in problematic environments such as the case the $p > n$. We illustrated the fact that both of the new estimators proposed offer comparable results to the more standard methods when the joint distribution of $(X^T, Y)^T$ is known and Δ^{-1} is non-sparse.

Future work is needed to explore the case when Δ^{-1} is sparse and/or $p \gg n$. Our simulations tease the fact that the indirect estimators, specifically $\hat{\beta}^{IL}$, might outperorm the forward regression methods when p is a magnitue much larger than n - however, due to time constraints we were unable to investigate further.

References

[1] Cook, R. Dennis, Liliana Forzani, and Adam J. Rothman. "Prediction in abundant high-dimensional linear regression." Electronic Journal of Statistics 7 (2013): 3059-3088.

[2] Molstad, Aaron J., and Adam J. Rothman. "Indirect multivariate response linear regression." Biometrika 103.3 (2016): 595-607.

Appendix

Proof of OLS estimator for β

Consider the log-likelihood of the joint distribution of X and Y:

$$\log g(X, Y; \beta) = \log f(Y|X, \beta)h(X) = \log f(Y|X, \beta) + \log h(X)$$

where $\log f(Y|X, \beta)$ can be simplified to the following form:

$$\begin{aligned}
\log f(Y|X, \beta) &= \log \prod_{i=1}^{n} f(Y_i|X_i, \beta) \\
&= \log \prod_{i=1}^{n} (2\pi\sigma^2_{y|x})^{-1/2} \exp\left\{ -\frac{1}{2\sigma^2_{y|x}} \left(Y_i - \mu_Y - \beta^T(X_i - \mu_X) \right)^2 \right\} \\
&= \log (2\pi\sigma^2_{y|x})^{-n/2} \exp\left\{ -\frac{1}{2\sigma^2_{y|x}} \sum_{i=1}^{n} \left(Y_i - \mu_Y - \beta^T(X_i - \mu_X) \right)^2 \right\} \\
&= -\frac{n}{2} \log(2\pi\sigma^2_{y|x}) - \frac{1}{2\sigma^2_{y|x}} \sum_{i=1}^{n} \left(Y_i - \mu_Y - \beta^T(X_i - \mu_X) \right)^2 \\
&= const. - \frac{1}{2\sigma^2_{y|x}} \sum_{i=1}^{n} \left(Y_i - \mu_Y - \beta^T(X_i - \mu_X) \right)^2
\end{aligned}$$

Because we are taking the gradient with respect to β and $\log h(X)$ does not depend on β, it can be ignored in further computation.

$$\begin{aligned}
\nabla_\beta \{\log g(X, Y; \beta)\} &= \nabla_\beta \{\log f(Y|X, \beta)\} \\
&= \nabla_\beta \left\{ -\frac{1}{2\sigma^2_{y|x}} \sum_{i=1}^{n} \left(Y_i - \mu_Y - \beta^T(X_i - \mu_X) \right)^2 \right\} \\
&= \nabla_\beta \left\{ -\frac{1}{2\sigma^2_{y|x}} \|\mathbb{Y} - \mathbb{X}\beta\|^2 \right\} \\
&= \nabla_\beta \left\{ -\frac{1}{2\sigma^2_{y|x}} (\mathbb{Y}^T\mathbb{Y} - 2\beta^T\mathbb{X}^T\mathbb{Y} + \beta^T\mathbb{X}^T\mathbb{X}\beta) \right\} \\
&= \frac{1}{\sigma^2_{y|x}} \mathbb{X}^T\mathbb{Y} - \frac{1}{\sigma^2_{y|x}} \mathbb{X}^T\mathbb{X}\beta
\end{aligned}$$

where $\mathbb{X} \in \mathbb{R}^{n \times p}$ with rows $X_i - \mu_X$ and $\mathbb{Y} \in \mathbb{R}^n$ with elements $Y_i - \mu_Y$.
Setting the gradient equal to zero, it follows that $\hat{\beta}^{MLE} = \hat{\beta}^{OLS} = (\mathbb{X}^T\mathbb{X})^{-1}\mathbb{X}^T\mathbb{Y}$.

Proof of indirect regression coefficient β

We stated as fact that $\Sigma_{xx} = \Delta + \Sigma_{xy}\Sigma_{xy}^T/\sigma_y^2$. Using the Woodbury Identity, it follows that

$$\begin{aligned}
\Sigma_{xx}^{-1} &= \left(\Delta + \Sigma_{xy}\Sigma_{xy}^T/\sigma_y^2\right)^{-1} \\
&= \Delta^{-1} - \Delta^{-1}\Sigma_{xy}\left(\sigma_y^2 + \Sigma_{xy}^T\Delta^{-1}\Sigma_{xy}\right)^{-1}\Sigma_{xy}^T\Delta^{-1} \\
&= \Delta^{-1} - \frac{\Delta^{-1}\Sigma_{xy}\Sigma_{xy}^T\Delta^{-1}}{\sigma_y^2 + \Sigma_{xy}^T\Delta^{-1}\Sigma_{xy}^T}
\end{aligned}$$

This directly implies that β is of the following form:

$$\begin{aligned}
\beta = \Sigma_{xx}^{-1}\Sigma_{xy} &= \Delta^{-1}\Sigma_{xy} - \frac{\Delta^{-1}\Sigma_{xy}\Sigma_{xy}^T\Delta^{-1}\Sigma_{xy}}{\sigma_y^2 + \Sigma_{xy}^T\Delta^{-1}\Sigma_{xy}^T} \\
&= \Delta^{-1}\Sigma_{xy}\left(1 - \frac{\Sigma_{xy}^T\Delta^{-1}\Sigma_{xy}}{\sigma_y^2 + \Sigma_{xy}^T\Delta^{-1}\Sigma_{xy}}\right) \\
&= \Delta^{-1}\Sigma_{xy}\left(\frac{\sigma_y^2}{\sigma_y^2 + \Sigma_{xy}^T\Delta^{-1}\Sigma_{xy}}\right) \\
&= \Delta^{-1}\Sigma_{xy}\left(1 + \Sigma_{xy}^T\Delta^{-1}\Sigma_{xy}/\sigma_y^2\right)^{-1}
\end{aligned}$$

Proof of MLE for Δ^{-1}

Recall that

$$X|Y = y \sim N_p\left(\mu_x + \alpha^T(Y - \mu_y), \Delta\right)$$

The likelihood and maximum likelihood estimators can be simplified to the following (using the same notation defined previously):

$$\begin{aligned}
l(\alpha, \Delta^{-1}) &= \sum_{i=1}^{n} \log \phi(X_i; \mu_x + \alpha^T(Y_i - \mu_y), \Delta^{-1}) \\
&= -\frac{np}{2}\log(2\pi) + \frac{n}{2}\log\left|\Delta^{-1}\right| - \frac{1}{2}\sum_{i=1}^{n}(X_i - \mu_x - \alpha^T(Y_i - \mu_y))^T\Delta^{-1}(X_i - \mu_x - \alpha^T(Y_i - \mu_y)) \\
&= const. + \frac{n}{2}\log\left|\Delta^{-1}\right| - \frac{n}{2}tr\left\{\frac{1}{n}\sum_{i=1}^{n}(X_i - \mu_x - \alpha^T(Y_i - \mu_y))(X_i - \mu_x - \alpha^T(Y_i - \mu_y))^T\Delta^{-1}\right\} \\
&= const. + \frac{n}{2}\log\left|\Delta^{-1}\right| - \frac{n}{2}tr\left\{\frac{1}{n}(\mathbb{X} - \mathbb{Y}\alpha)^T(\mathbb{X} - \mathbb{Y}\alpha)\Delta^{-1}\right\}
\end{aligned}$$

$$\begin{aligned}\nabla_\alpha l(\alpha, \Delta^{-1}) &= -\nabla_\alpha \frac{n}{2} tr\left\{\frac{1}{n}(\mathbb{X} - \mathbb{Y}\alpha)^T(\mathbb{X} - \mathbb{Y}\alpha)\Delta^{-1}\right\} \\ &= -\frac{1}{2}\nabla_\alpha tr\left\{-2\alpha^T\mathbb{Y}^T\mathbb{X}\Delta^{-1} + \alpha^T\mathbb{Y}^T\mathbb{Y}\alpha\Delta^{-1}\right\} \\ &= \mathbb{Y}^T\mathbb{X}\Delta^{-1} - \mathbb{Y}^T\mathbb{Y}\alpha\Delta^{-1}\end{aligned}$$

Setting the gradient equal to zero, it follows that $\hat{\alpha}^{MLE} = (\mathbb{Y}^T\mathbb{Y})^{-1}\mathbb{Y}^T\mathbb{X}$ – which we know is identifiabile because $\mathbb{Y}^T\mathbb{Y}$ is a scalar. Now we take the gradient with respect to Δ^{-1}:

$$\begin{aligned}\nabla_{\Delta^{-1}} l(\alpha, \Delta^{-1}) &= \nabla_{\Delta^{-1}}\left[\frac{n}{2}\log\left|\Delta^{-1}\right| - \frac{n}{2} tr\left\{\frac{1}{n}(\mathbb{X} - \mathbb{Y}\alpha)^T(\mathbb{X} - \mathbb{Y}\alpha)\Delta^{-1}\right\}\right] \\ &= \nabla_{\Delta^{-1}}\left[\frac{n}{2}\log\left|\Delta^{-1}\right| - \frac{n}{2} tr\left\{\hat{\Sigma}_{x|y}\Delta^{-1}\right\}\right] \\ &= \frac{n}{2}\Delta - \frac{n}{2}\hat{\Sigma}_{x|y}\end{aligned}$$

where $\hat{\Sigma}_{x|y} = (\mathbb{X} - \mathbb{Y}\alpha)^T(\mathbb{X} - \mathbb{Y}\alpha)/n$. This is the residual sample variance for the regression X on Y (denominator n). Setting the gradient equal to zero, it follows that $\hat{\Delta}^{-1}_{MLE} = \hat{\Sigma}^{-1}_{x|y}$ (if it exists) where α is replaced with $\hat{\alpha}^{MLE} = (\mathbb{Y}^T\mathbb{Y})^{-1}\mathbb{Y}^T\mathbb{X}$.

Proof of ridge penalized Δ^{-1}

$$\hat{\Delta}^{-1}_\lambda = \arg\min_{\Theta \in \mathbb{S}^p_+}\left\{tr(\Theta\hat{\Sigma}_{x|y}) - log\,|\Theta| + \frac{\lambda}{2}\|\Theta\|^2_F\right\}$$

Let g be the objective function in the previous equation.

$$\begin{aligned}\nabla_\Theta g(\Theta) &= \nabla_\Theta\left\{tr(\Theta\hat{\Sigma}_{x|y}) - log\,|\Theta| + \frac{\lambda}{2}\|\Theta\|^2_F\right\} \\ &= \hat{\Sigma}_{x|y} - \Theta^{-1} + \lambda\Theta\end{aligned}$$

Setting the gradient equal to zero…

$$\begin{aligned}\hat{\Sigma}_{x|y} &= \hat{\Theta}^{-1} - \lambda\hat{\Theta} \\ &= (VDV^T)^{-1} - \lambda VDV^T \\ &= V(D^{-1} - \lambda D)V^T\end{aligned}$$

using the spectral decomposition $\hat{\Theta} = VDV^T$ where D is a diagonal matrix with diagonal elements being the eigen values of Θ and V is matrix with the corresponding eigen vectors as columns. This structure implies that

$$\phi_j(\hat{\Sigma}_{x|y}) = \frac{1}{\phi_j(\hat{\Theta})} - \lambda\phi_j(\hat{\Theta})$$

where $\phi_j(\cdot)$ is the jth eigen value.

$$\begin{aligned}
&\Rightarrow \lambda\phi_j(\hat{\Theta}) + \phi_j(\hat{\Sigma}_{x|y})\phi_j(\hat{\Theta}) - 1 = 0 \\
&\Rightarrow \phi_j(\hat{\Theta}) = \frac{-\phi_j(\hat{\Sigma}_{x|y}) \pm \sqrt{\phi_j^2(\hat{\Sigma}_{x|y}) + 4\lambda}}{2\lambda}
\end{aligned}$$

In summary, if we decompose $\hat{\Sigma}_{x|y} = VQV^T$ then

$$\hat{\Theta}_\lambda = \begin{cases} \frac{1}{2\lambda} V \left[-Q + (Q^2 + 4\lambda I_p)^{1/2} \right] V^T & \text{if } \lambda > 0 \\ \hat{\Sigma}_{x|y}^{-1} & \text{if } \hat{\Sigma}_{x|y}^{-1} \text{ exists and } \lambda = 0 \end{cases}$$

(proof taken from Adam Rothman's STAT 8931 lecture notes.)

Code

```
# All code Code taken and/or augmented
# from Adam Rothman's STAT 8931 course

# libraries
library(glmnet)
library(glasso)

# define sigma ridge function
sigma_ridge = function(S, lam) {

    # dimensions
    p = dim(S)[1]

    # gather eigen values of S (spectral
    # decomposition)
    e.out = eigen(S, symmetric = TRUE)

    # augment eigen values for omega hat
    new.evs = (-e.out$val + sqrt(e.out$val^2 +
        4 * lam))/(2 * lam)

    # compute omega hat for lambda (zero
    # gradient equation)
    omega = tcrossprod(e.out$vec * rep(new.evs,
        each = p), e.out$vec)

    return(omega)
}


# define betaI function
betaI = function(delta, Sxy, Syy) {

    # betaI
    delta %*% Sxy/as.numeric(1 + t(Sxy) %*%
        delta %*% Sxy/as.numeric(Syy))

}


# define betaMP function
betaMP = function(X, Y) {
```

```
    # betaMP
    MASS::ginv(t(X) %*% X) %*% (cov(X, Y) *
        (nrow(X) - 1)/nrow(X))
}


# define betaR function
betaR = function(X, Y, lam) {

    # betaR
    m = glmnet(X, Y, alpha = 0, lambda = lam,
        intercept = F)
    predict(m, type = "coefficients")[-1,
        ]

}


# define betaL function
betaL = function(X, Y, lam) {

    # betaL
    m = glmnet(X, Y, lambda = lam, intercept = F)
    predict(m, type = "coefficients")[-1,
        ]

}


# define model error function
ME = function(beta_hat, beta, Sxx) {

    # ME
    t(beta_hat - beta) %*% Sxx %*% (beta_hat -
        beta)

}

# define mean squared error function
MSE = function(beta, X.valid, Y.valid) {

    # loss
    mean((X.valid %*% beta - Y.valid)^2)
```

```
}


# define CV function
CV = function(X, Y, lam, ind = NULL, K = 5,
    quiet = TRUE, crit = NULL) {

    # dimensions of data
    n = dim(X)[1]
    p = dim(X)[2]

    # if the user did not specify a
    # permutation of 1,..,n, then randomly
    # permute the sequence:
    if (is.null(ind))
        ind = sample(n)

    # allocate the memory for the loss matrix
    # (rows correspond to values of the
    # tuning paramter) (columns correspond to
    # folds)
    cv.loss = array(0, c(length(lam), 4,
        K))

    for (k in 1:K) {

        leave.out = ind[(1 + floor((k - 1) *
            n/K)):floor(k * n/K)]

        # training set
        X.train = X[-leave.out, , drop = FALSE]
        X_bar = apply(X.train, 2, mean)
        X.train = scale(X.train, center = X_bar,
            scale = FALSE)

        Y.train = Y[-leave.out, , drop = FALSE]
        Y_bar = apply(Y.train, 2, mean)
        Y.train = scale(Y.train, center = Y_bar,
            scale = FALSE)

        # validation set
        X.valid = X[leave.out, , drop = FALSE]
        X.valid = scale(X.valid, center = X_bar,
            scale = FALSE)
```

```
Y.valid = Y[leave.out, , drop = FALSE]
Y.valid = scale(Y.valid, center = Y_bar,
    scale = FALSE)

# sample covariances
Sxx = crossprod(X.train)/nrow(X.train)
Sxy = crossprod(X.train, Y.train)/nrow(X.train)
Syy = crossprod(Y.train)/nrow(Y.train)

m = lm.fit(Y.train, X.train)
Sx.y = crossprod(m$residuals)/nrow(X.train)
Sx.y.valid = crossprod(X.valid -
    Y.valid %*% m$coefficients)/nrow(X.valid)

# glasso
out = glassopath(s = Sx.y, rholist = lam,
    penalize.diagonal = FALSE, trace = 0,
    thr = 0.001, maxit = 3)

# loop over all lambda values
for (i in 1:length(lam)) {

    # lambda
    lam. = lam[i]

    # loss for betaIR
    deltaIR = sigma_ridge(Sx.y, lam.)
    lossIR = sum(deltaIR * Sx.y.valid) -
        determinant(deltaIR, logarithm = TRUE)$modulus[1]
    betaIR = betaI(deltaIR, Sxy,
        Syy)

    # loss for betaIL
    deltaIL = out$wi[, , i]
    lossIL = sum(deltaIL * Sx.y.valid) -
        determinant(deltaIL, logarithm = TRUE)$modulus[1]
    betaIL = betaI(deltaIL, Sxy,
        Syy)

    # loss betaR
    betaR. = betaR(X.train, Y.train,
        lam.)
    lossR = MSE(betaR., X.valid,
        Y.valid)
```

```
        # loss betaL
        betaL. = betaL(X.train, Y.train,
            lam.)
        lossL = MSE(betaL., X.valid,
            Y.valid)

        # if criteria not NULL, use prediction
        # MSE as criteria for deciding lambda for
        # betaIR and betaIL
        if (!is.null(crit)) {

            lossIR = MSE(betaIR, X.valid,
              Y.valid)
            lossIL = MSE(betaIL, X.valid,
              Y.valid)

        }

        # designate loss
        cv.loss[i, , k] = c(lossIR, lossIL,
            lossR, lossL)

    }

    # if not quiet, then print progress fold
    if (!quiet)
        cat("Finished fold", k, "\n")

}

# accumulate the error over the folds
cv.err = apply(cv.loss, c(1, 2), sum)

# find the best tuning parameter values
best.loc = apply(cv.err, 2, which.min)
best.lam = lam[best.loc]

# best betas
Sxy = crossprod(X, Y)/nrow(X)
Syy = crossprod(Y)/nrow(Y)

m = lm.fit(Y, X)
Sx.y = crossprod(m$residuals)/nrow(X)
```

```
    betaIR = sigma_ridge(Sx.y, best.lam[1]) %>%
        betaI(Sxy, Syy)
    betaIL = out$wi[, , best.loc[2]] %>%
        betaI(Sxy, Syy)
    betaR. = betaR(X, Y, best.lam[3])
    betaL. = betaL(X, Y, best.lam[4])

    # compute final estimate at the best
    # tuning parameter value
    beta_hat = matrix(c(betaIR, betaIL, betaR.,
        betaL.), ncol = 4)
    colnames(beta_hat) = c("BIR", "BIL",
        "BR", "BL")

    return(list(beta_hat = beta_hat, best.lam = best.lam,
        cv.err = cv.err, lam = lam))
}



## SIMULATION

# initialize values
lam = 10^seq(-4, 8, 0.5)
reps = 50
N = 100
# P = c(10, 25, 80, 120)
P = 120
# Syy = c(0.3, 0.7)
Syy = 0.7
# Bin = c(0.3, 0.9)
Bin = 0.3

# allocate memory
sim = array(0, c(reps, 5, length(N), length(P),
    3, length(Syy), length(Bin)), dimnames = list(Reps = c(1:reps),
    Beta = c("IR", "IL", "R", "L", "MP"),
    N = c(N), P = c(P), criteria = c("MSE",
        "ME", "Boundary"), Sigmay = c(Syy),
    Binom = c(Bin)))


# lots of loops
for (n in 1:length(N)) {
```

```
  for (p in 1:length(P)) {
      for (s in 1:length(Syy)) {
          for (b in 1:length(Bin)) {
              for (r in 1:reps) {

                  # initialize values set variance for Y
                  syy = Syy[s]

                  # Y ~ N(0, Syy)
                  Y = matrix(rnorm(N[n],
                    sd = sqrt(syy)), ncol = 1)

                  # set true alpha
                  alpha = matrix(rnorm(P[p]),
                    nrow = 1) * matrix(rbinom(P[p],
                    1, Bin[b]), nrow = 1)
                  alpha[1, 1] = rnorm(1)

                  # delta has off-diagonal entries equal to
                  # 0.9
                  delta = matrix(NA, nrow = P[p],
                    ncol = P[p])
                  for (j in 1:P[p]) {
                    for (k in 1:P[p]) {
                      delta[j, k] = 0.9 *
                        (j != k) + 1 * (j ==
                        k)
                    }
                  }

                  # X ~ N(mu, delta)
                  X = Y %*% alpha + matrix(rnorm(N[n] *
                    P[p]), ncol = P[p]) %*%
                    t(chol(delta))

                  # Based on the previous values we can
                  # solve for beta and Sxx
                  beta = qr.solve(delta,
                    t(alpha))/as.numeric(1/syy +
                    alpha %*% (qr.solve(delta,
                      t(alpha))))
                  Sxx = delta + t(alpha) %*%
                    alpha * syy
```

```
                    # run CV to find optimal betas
                    cv = CV(X, Y, lam = lam,
                      K = 3, quiet = T)
                    beta_hat = cv$beta_hat

                    # fill in metrics for each estimators
                    for (i in 1:4) {
                      # MSE and ME criteria
                      sim[r, i, n, p, 1, s,
                        b] = MSE(beta_hat[,
                        i, drop = F], X, Y)
                      sim[r, i, n, p, 2, s,
                        b] = ME(beta_hat[,
                        i, drop = F], beta,
                        Sxx)

                      # boundary of lambda?
                      if (min(lam) %in% cv$best.lam[i]) {
                        cat("Oops! Lamda on boundary. \n")
                        sim[r, i, n, p, 3,
                          s, b] = 1
                      }
                    }

                    sim[r, 5, n, p, 1, s, b] = MSE(betaMP(X,
                      Y), X, Y)
                    sim[r, 5, n, p, 2, s, b] = ME(betaMP(X,
                      Y), beta, Sxx)

                    if (min(lam) %in% cv$best.lam[5]) {
                      cat("Oops! Lamda on boundary. \n")
                      sim[r, 5, n, p, 3, s,
                        b] = 1
                    }

                    cat("Finished rep", r,
                      "bin", Bin[b], "sigma",
                      Syy[s], "P", P[p], "N",
                      N[n], "\n")
                  }
                }
              }
            }
          }
```

```r
# designate simulation data as table
data = sim %>% as.data.frame.table(responseName = "Error")
```