MATH5472 Assignment 2

CHAN SHUN HIN (20100545)

Mar 19, 2020

Question 1.

(a) Let $k = \min\{n, p\}$. Note that β minimizes $\|X\beta - y\|^2$ iff $X\beta$ is the orthogonal projection of y onto the column space of X as a subspace U of $\mathbb{R}^m$, since $\|X\beta\|^2 + \|y - X\beta\|^2 = \|y\|^2$ so that there is a unique $X\beta$ that minimize the distance from y.

We first introduce the singular value decomposition of $X = VDU^T$, where $D = \text{diag}(\lambda_1, \ldots, \lambda_k, 0, \ldots, 0)$. Define $D^+ = \text{diag}(\lambda_1^{-1}, \ldots, \lambda_k^{-1}, 0, \ldots, 0)$, the *pseudo-inverse* of X is defined as

$$A^+ = UD^+V^T.$$

We claim that the solution of the problem $\hat{\beta} = \arg\min_\beta \|y - X\beta\|^2$ is given by

$$\beta^+ = X^+y = UD^+V^Ty.$$

Proof. Assume D to be a rectangular diagonal matrix. Consider a simple case of diagonal D. Then, $\beta = D^+y$ minimizes $\|D\beta - y\|^2$.

For general X, we write

$$\|X\beta - y\| = \|VDU^T\beta - y\| = \|DU^T\beta - V^Ty\|$$

since V is an isometry. Since U is surjective, $\|X\beta - y\|$ is minimized iff $\|D\gamma - V^Ty\|$ is minimized, where $\gamma = U^T\beta$. The solution is

$$\hat{\gamma} = D^+V^Ty$$

from the first case of our proof.

Then, the solution of the problem $\hat{\beta} = \arg\min_\beta \|y - X\beta\|^2$ is given by

$$\hat{\beta} = UD^+V^Ty = X^+y.$$

Furthermore, the Moore-Penrose inverse is given by

$$X^+ = \begin{cases} (X^TX)^{-1}X^T & \text{when } X \text{ has linearly independent columns (i.e., } n \geq p) \\ X^T(XX^T)^{-1} & \text{when } X \text{ has linearly independent rows (i.e., } n < p) \end{cases}$$

which fulfill Moore–Penrose conditions. Since the pseudo inverse is unique, then the solution of the minimization problem is given by $\hat{\beta} = X^+y$. □

Note that the prediction of y is given by $\hat{y} = X\hat{\beta}$. The degrees of freedom is given by

$$\hat{\text{df}} = \text{trace}(XX^+) = \begin{cases} \text{trace}(X(X^TX)^{-1}X^T) & \text{if } n \geq p \\ \text{trace}(XX^T(XX^T)^{-1}) & \text{if } n < p \end{cases} = \min\{n, p\}.$$

(b) A sample of size $n = 150$ was simulated from the following setting:

$$\begin{aligned} \vec{y} &= X_{150\times 50}\vec{\beta}_{50\times 1} + \vec{\varepsilon}, \\ [X]_{ij} &\overset{iid}{\sim} N(0, 0.2^2), \\ \beta_i &\overset{iid}{\sim} N(0, 0.2^2), \\ \varepsilon_i &\overset{iid}{\sim} N(0, 1). \end{aligned}$$

for i=1,2,...,150; j=1,2,...,50. Only ε's are considered as random when fitting least squares regression model. n is always fixed at 150, while we try to change the dimension of y_i's. 50% of data is assigned as training data and the remaining 50% is for testing.

For each procedure below, we do replication of R=200 times. For each dimension $p = 1, 2, \ldots, 300$, we generate the design matrix whose elements are from $[X_{150\times p}]_{ij} \overset{iid}{\sim} N(0, 0.2^2)$ (since we keep changing the dimension, we cannot fix a design matrix), then do the prediction of y's and we call them $\hat{y}$'s. Define $\theta^{(r)}$ as the value in the r-th replication for any paramete θ. For both training and testing data, the **average bias** is calculated as

$$\overline{Bias} = \frac{1}{R}\frac{1}{0.5n}\sum_{i=1}^{0.5n}\sum_{r=1}^{R}\left(\hat{y}_i^{(r)} - y_i\right)^2,$$

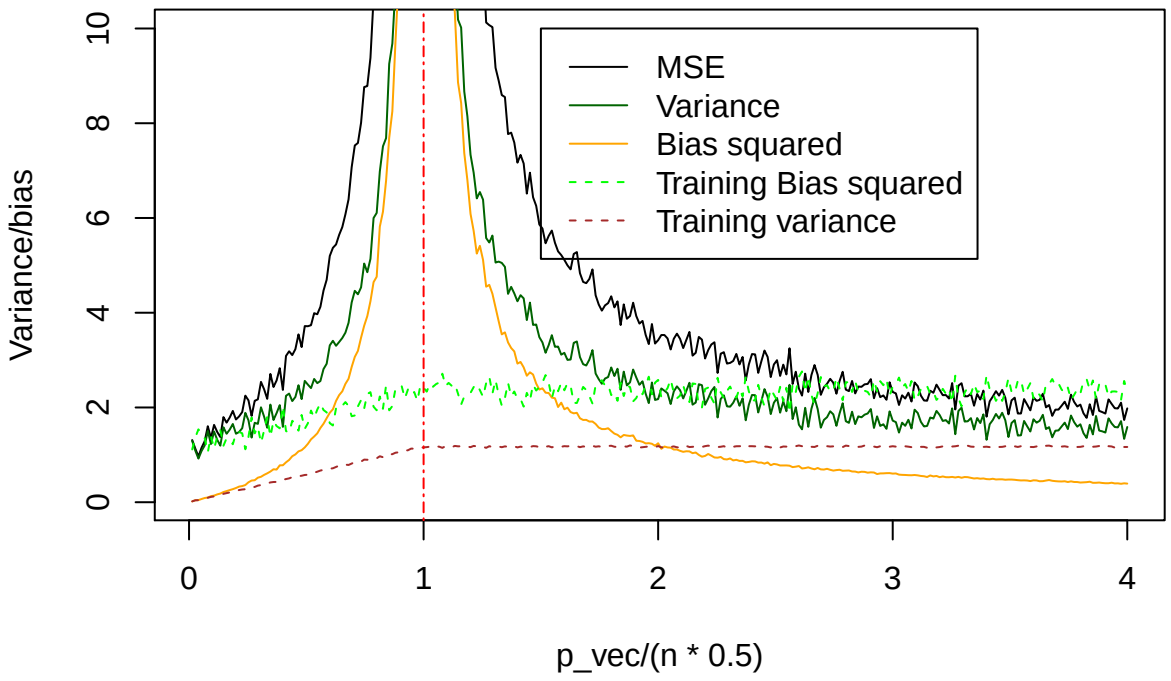
and the **average variance** is calculated as

$$\overline{Var} = \frac{1}{R}\frac{1}{0.5n}\sum_{i=1}^{0.5n}\sum_{r=1}^{R}\left(\hat{y}_i^{(r)} - \frac{1}{R}\sum_{r'=1}^{R}(\hat{y}_i^{(r')})\right)^2.$$

The code and the plot are shown below. The y-axis is the MSE, average variance, average bias for both training and testing data, and the x-axis is $\gamma = \frac{p}{0.5n}$. We can observe double-descent pattern with interpolation threshold at $\gamma = 1$.

```
# n <- 150
# p_vec <- seq(1,300,1)

#true model
# p <- 50
# betas <- array(rnorm(p,0,0.2),dim=c(p,1))
# X <- array(rnorm(n*max(p_vec),0,0.2),dim=c(n,max(p_vec)))
# ep <- array(rnorm(n,0,1),dim=c(n,1))
#
# Y <- X[,1:p] %*% betas + ep
# nrep <- 200
# bias <- array(dim=max(p_vec))
# var <- array(dim=max(p_vec))
#
# train.bias <- array(dim=max(p_vec))
# train.var <- array(dim=max(p_vec))
# for(p in p_vec){ # p<-2
#   pred <- array(dim=c(n*0.5,nrep))
#   train.pred <- array(dim=c(n*0.5,nrep))
#   for(r in 1:nrep){ #r<-1
#     gamma <- p/(0.5*n)
#     x <- array(rnorm(n*p,0,0.2),dim=c(n,p))
#     n.train <- sample(1:n,0.5*n)
#     n.test <- setdiff(1:n,n.train)
#     x.train <- x[n.train,]
```

```r
#     x.test <- x[n.test,]
#     y.train <- Y[n.train]
#     y.test <- Y[n.test]
#     if(gamma<1){
#       beta.est <- solve(t(x.train)%*%x.train)%*%t(x.train)%*%y.train
#       pred[,r] <- x.test %*% beta.est
#       train.pred[,r] <- x.train %*% beta.est
#     }else{
#       beta.est <- t(x.train)%*% solve(x.train%*%t(x.train))%*%y.train
#       pred[,r] <- x.test %*% beta.est
#       train.pred[,r] <- x.train %*% beta.est
#     }
#   }
#   bias[p] <- 1/nrep*1/(n*0.5)*sum((pred-array(rep(y.test,n*0.5),dim=c(n*0.5,nrep)))^2)
#   var[p] <-sum((pred-array(rep(rowMeans(pred),n*0.5),dim=c(n*0.5,nrep)))^2)/(n*0.5)/nrep
#
#   train.bias[p] <- 1/nrep*1/(n*0.5)*sum((train.pred-array(rep(y.train,n*0.5)
#,dim=c(n*0.5,nrep)))^2)
#   train.var[p] <-sum((train.pred-array(rep(rowMeans(train.pred),n*0.5)
#,dim=c(n*0.5,nrep)))^2)/(n*0.5)/nrep
# }
n <-150
ylim_vec <- c(min(bias[p_vec],var[p_vec]),10)
plot(p_vec/(n*0.5),bias[p_vec]+var[p_vec],type="l",ylim=ylim_vec,ylab="Variance/bias")
lines(p_vec/(n*0.5),bias[p_vec],lty=1,col="darkgreen")
lines(p_vec/(n*0.5),var[p_vec],lty=1,col="orange")
lines(p_vec/(n*0.5),train.bias[p_vec],lty=2,col="green")
lines(p_vec/(n*0.5),train.var[p_vec],lty=2,col="brown")
legend(1.5,10,legend=c("MSE","Variance","Bias squared",
"Training Bias squared","Training variance"),
       col=c("black","darkgreen","orange","green","brown"),lty=c(1,1,1,2,2))
abline(v=1,lty=6,col="red")
```



We can observe that, when γ is close to 1, both MSE and bias increase. When γ is away from 1, they both decrease. The Variance is higher than the bias when near $\gamma = 1$, while the variance is lower than the bias when γ is away from 1. This shows the double descent phenomenon of bias-variance trade-off.

(c) Rearrange the gradient descent equation:

$$\beta^{(t)} = \left(I - \varepsilon\frac{X^TX}{n}\right)\beta^{(t-1)} + \varepsilon\frac{X^Ty}{n} := A\beta^{(t-1)} + c.$$

Inductively,

$$\begin{aligned}
A\beta^{(t)} &= A\beta^{(t-1)} + c \\
&= A^2\beta^{(t-2)} + (I + A)c \\
&= \dots \\
&= A^t\beta^{(0)} + (I + A + A^2 + \dots + A^{t-1})c \\
&= A^t\beta^{(0)} + (I - A)^{-1}(1 + A^t)c && \text{if } X^TX \text{ is invertible.} \\
&= A^t\beta^{(0)} + \frac{n}{\varepsilon}(X^TX)^{-1}(1 + A^t)\frac{\varepsilon}{n}X^Ty. \\
&= A^t\beta^{(0)} + (X^TX)^{-1}X^Ty + (X^TX)^{-1}A^tX^Ty.
\end{aligned}$$

We claim that $\lim_{t\to+\infty} A^t = 0$.

Proof. We diagonalize $X^TX = PDP^T$, where $D = \text{diag}\lambda_1, \dots, \lambda_p$, where λ_i is the i-th largest eigenvalue of X^TX and P is the corresponding orthogonal matrix of eigenvectors. Then, we rewrite A as

$$\begin{aligned}
A &= I - \frac{\varepsilon}{n}X^TX \\
&= PP^T - \frac{\varepsilon}{n}PDP^T \\
&= P(I - \frac{\varepsilon}{n}D)P^T \\
&= P\text{diag}\left(1 - \frac{1}{n}\varepsilon\lambda_i\right)_{i=1,2,\dots,p} P^T
\end{aligned}$$

It is given that $\varepsilon \le 1/\lambda_{max}(X^TX)$ so that $\varepsilon\lambda_i \le \frac{\lambda_i}{\lambda_{max}} < 1$ since the matrix has no linear dependent columns so that $\lambda_i < \lambda_{max} \equiv \lambda_1$ for all $i \ge 2$.

Then,

$$\begin{aligned}
\lim_{t\to+\infty} A^t &= \lim_{t\to+\infty} PD^tP^T \\
&= \lim_{t\to+\infty} P\text{diag}\left(1 - \frac{1}{n}\varepsilon\lambda_i\right)^t_{i=1,2,\dots,p} P^T && \text{since } 1 - \frac{1}{n}\varepsilon\lambda_i < 1, \\
&= \mathbf{0}.
\end{aligned}$$

Hence, we have

$$\lim_{t\to+\infty} \beta^{(t)} = \mathbf{0} + (X^TX)^{-1}X^Ty + \mathbf{0} = (X^TX)^{-1}X^Ty,$$

if X^TX is invertible. i.e., the gradient descent converge to the optimal solution given in (a) only if $n \ge p$.

(c) Rearrange the terms of the differential equation:

$$\beta'(t) + \frac{X^TX}{n}\beta(t) = \frac{X^Ty}{n}$$

Let $A := \frac{X^T X}{n}$ and $B := \frac{X^T y}{n}$. Left multiplying the integrating factor $e^{At} := \sum_{k=0}^{\infty} \frac{(At)^k}{k!}$:

$$\begin{aligned}
\beta'(t) + A\beta(t) &= B \\
e^{At}\beta'(t) + e^{At}A\beta(t) &= e^{At}B \\
\frac{d}{dt}\left(e^{At}\beta(t)\right) &= e^{At}B \\
e^{AT}\beta(T) - \beta(0) &= \int_0^T e^{At}B dt \\
&= A^{-1}e^{At}B\Big|_0^T \\
&= A^{-1}e^{AT}B - A^{-1}B \\
\implies \beta(T) &= e^{-AT}(\beta(0) + A^{-1}e^{AT}B - A^{-1}B) \\
&= e^{-AT}\beta(0) + e^{-AT}A^{-1}e^{AT}B - e^{-AT}A^{-1}B \\
&= e^{-AT}\beta(0) + e^{-AT}e^{AT}A^{-1}B - e^{-AT}A^{-1}B \quad \text{(by (1))} \\
&= e^{-AT}\beta(0) + (I - e^{-AT})A^{-1}B \\
&= e^{-AT}\beta(0) + (I - e^{-AT})(X^T X)^{-1}X^T y.
\end{aligned}$$

(1): $A^{-1}e^{AT} = e^{AT}A^{-1}$ since

$$A^{-1}e^{AT} = A^{-1}\sum_{k=0}^{\infty}\left(\frac{1}{k!}A^{k+1}A^{-1}\right) = \left(A^{-1}\sum_{k=0}^{\infty}\frac{1}{k!}A^{k+1}\right)A^{-1} = \sum_{k=0}^{\infty}\frac{1}{k!}A^k A^{-1} = e^{AT}A^{-1}.$$

□

(e) One may consider $\beta(0)$ is a prior knowledge and T is the tuning parameter of the shrinkage. When $T = 0$, we put all weight to the prior knowledge; when $T \to \infty$, we ignore the prior knowledge.

The similarity between ridge regression and gradient flow is that they are both shrinkage estimations of the regression coefficient. The difference is that the ridge regression shrinks the parameters to 0 while the gradient flow can shrinks the parameters to any pre-specified vector $\beta(0)$.

The shrinkage term is $e^{-T\frac{X^T X}{n}}$, which is the matrix exponential of the sample covariance matrix. The gradient flow considers the correlation between beta's when doing shrinkage.

A simulation study was conducted, with a sample of size 900 simulated from

$$\begin{aligned}
\vec{y} &= X_{900\times 5}\vec{\beta}_{5\times 1} + \vec{\varepsilon}, \\
[X]_{ij} &\overset{iid}{\sim} N(0,1), \\
\beta_i &\overset{iid}{\sim} N(0,1), \\
\varepsilon_i &\overset{iid}{\sim} N(0,5^2).
\end{aligned}$$

The errors are intentionally set to have a high variance for modelling noisy data.

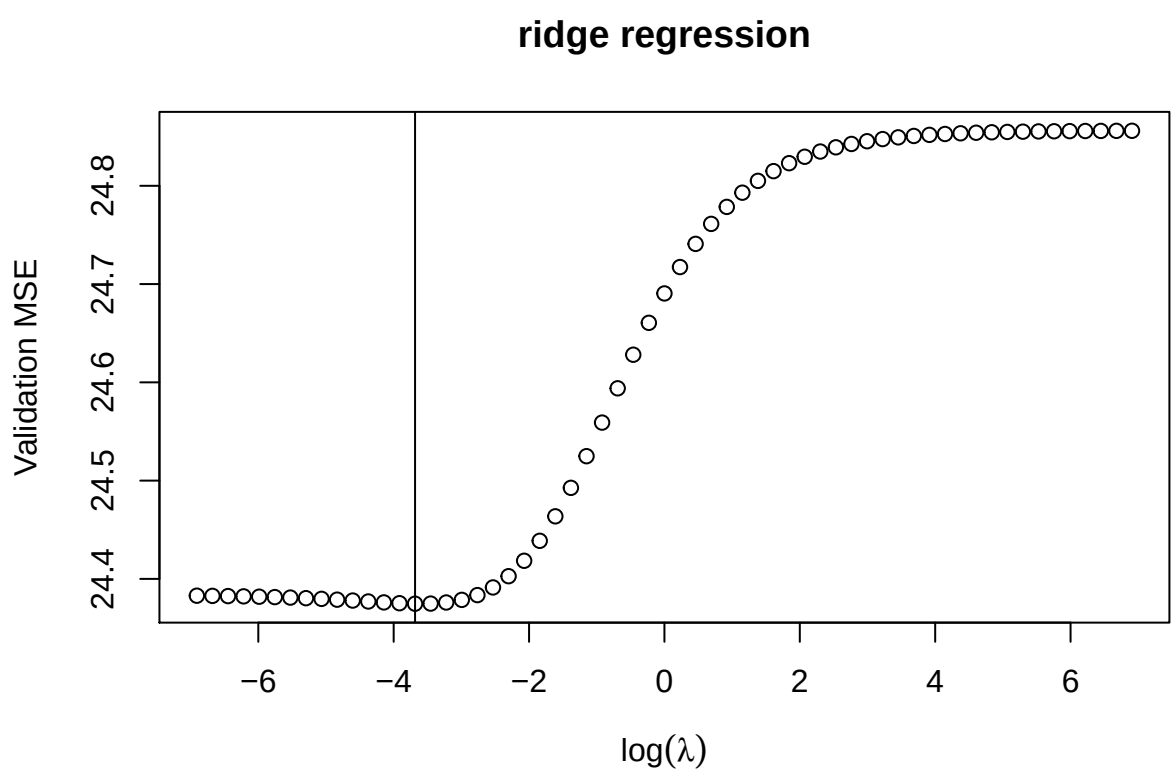
20 replications were done. In each replication, the dataset is divided into training subset, validation set and testing set. The tuning parameters are found using 10-fold cross validation with MSE measure. From the plots, we can see there is a value that minimize the MSE for ridge regression and gradient flow.

An interesting point is that the MSEs of gradient flow is slightly smaller than that of ridge regression on average.

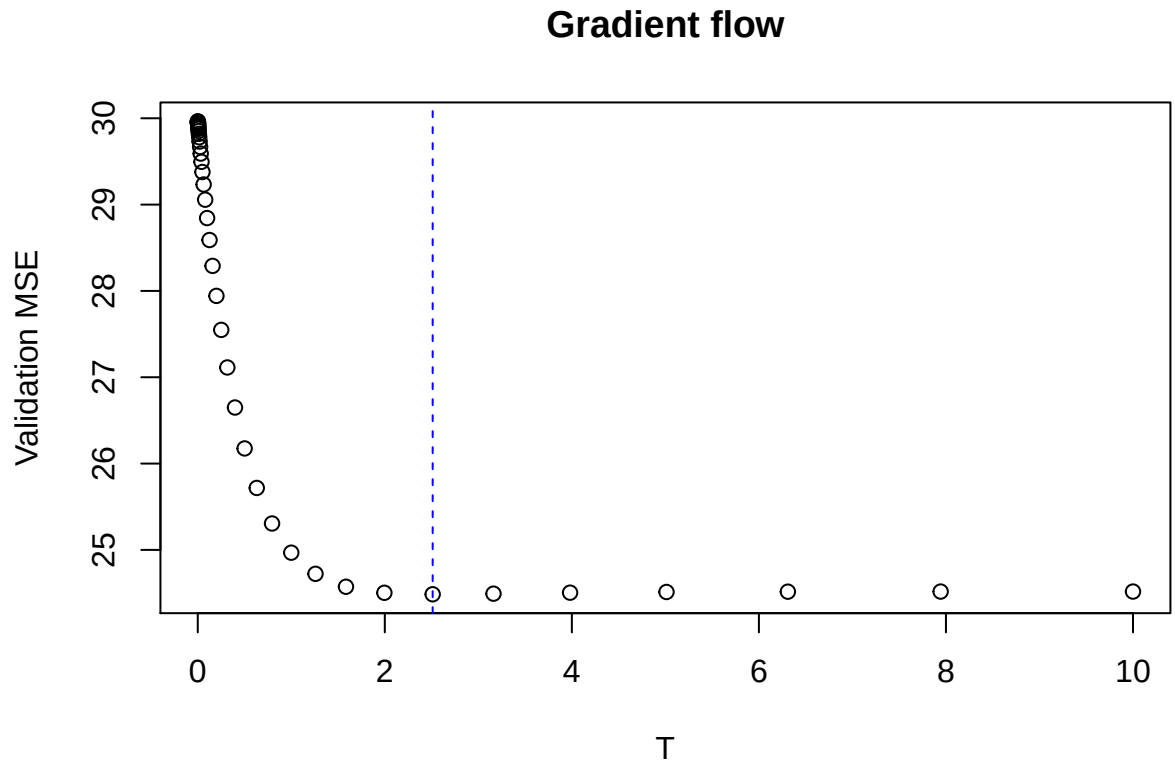
```
# # install.packages("tidyverse")
# # install.packages("broom")
# # install.packages("glmnet")
# library(tidyverse)
# library(broom)
# library(glmnet)
#
# set.seed(1)
# n <- 900
# p <- 5
#
#
# X <- array(rnorm(n*p,0,1),dim=c(n,p))
# #X.prime <- X[,1:p.prime]
#
# betas <- array(c(rnorm(p-1,0,1),0.01),dim=c(p,1))
# epsilon <- array(rnorm(n,0,5),dim=c(n,1))
#
# y <- X%*%betas+epsilon
#
# train.prop <- 0.5
# test.prop <- 1-train.prop
#
# nrep <- 20
# ncv <- 10
#
# lambdas <- 10^seq(-3,3, by = .1)
# t_vec <-10^seq(-3,1, by = .1)
# # par(mfrow=c(1,1))
# # model1 <- cv.glmnet(X[train.vec,],y[train.vec],alpha=0,lambda=lambdas,type.measure = "mse")
# # plot(model1)
# #
# # model1$lambda.min
# # model1$lambda.1se
#
# ridge.mse <- array(dim=c(nrep,2))
# t.mse <- array(dim=c(nrep,2))
#
# for(r in 1:nrep){
#  #Ridge
#  train.vec <- sample(1:n,0.5*n)
#  test.vec <- setdiff(1:n,train.vec)
#  cv.vec <- split(sample(train.vec,length(train.vec)),1:ncv)
#  cv.error.ridge <- array(dim=c(length(lambdas),ncv))
#  i <- 1
#  #CV
#  for(lamb in lambdas){ #lamb <- lambdas[1]
#    for(icv in 1:ncv){ #icv<-1
#      validation.vec <- cv.vec[[icv]]
#      train.subset.vec <- setdiff(train.vec,validation.vec)
#      n.train.sub <- length(train.subset.vec)
#      X.train.sub <- X[train.subset.vec,]
#      y.train.sub <- y[train.subset.vec,]
```

```
#      beta.ridge <- solve(t(X.train.sub)%*%X.train.sub+lamb*
#n.train.sub)%*%t(X.train.sub)%*%y.train.sub
#
#      X.valid <- X[validation.vec,]
#      y.valid <- y[validation.vec,]
#
#      y.pred.cv <- X.valid %*% beta.ridge
#      cv.error.ridge[i,icv] <- mean((y.pred.cv-y.valid)^2)
#    }
#    i <- i+1
#  }
#  #find best lambda
#  lambda.min <- lambdas[which.min(rowMeans(cv.error.ridge))]
#  # lambda.min
#  # plot(log(lambdas),rowMeans(cv.error))
#  # abline(v=log(lambda.min))
#
#  X.train <- X[train.vec,]
#  y.train <- y[train.vec]
#  X.test <- X[test.vec,]
#  y.test <- y[test.vec]
#  n.train <- nrow(X.train)
#  beta.train <- solve(t(X.train.sub)%*%X.train.sub+lambda.min*
#n.train)%*%t(X.train.sub)%*%y.train.sub
#  y.pred <- X.test %*% beta.train
#  ridge.mse[r,1] <- mean((y.pred-y.test)^2)
#  ridge.mse[r,2] <- lambda.min
#  #Gradient ------------------------------------------------------------------
#  #train.vec <- sample(1:n,0.5*n)
#  #test.vec <- setdiff(1:n,train.vec)
#  cv.vec <- split(sample(train.vec,length(train.vec)),1:ncv)
#  cv.error.grad <- array(dim=c(length(t_vec),ncv))
#  i <- 1
#  #CV
#  for(t in t_vec){ #lamb <- lambdas[1] #t<-0.1
#    for(icv in 1:ncv){ #icv<-1
#      validation.vec <- cv.vec[[icv]]
#      train.subset.vec <- setdiff(train.vec,validation.vec)
#      n.train.sub <- length(train.subset.vec)
#      X.train.sub <- X[train.subset.vec,]
#      y.train.sub <- y[train.subset.vec,]
#      A <- t(X.train.sub)%*%X.train.sub/n.train.sub
#      beta.grad <- (diag(p)-expm(-A*t))%*%solve(t(X.train.sub)%*%
#                              X.train.sub)%*%t(X.train.sub)%*%y.train.sub
#
#      X.valid <- X[validation.vec,]
#      y.valid <- y[validation.vec,]
#
#      y.pred.cv <- X.valid %*% beta.grad
#      cv.error.grad[i,icv] <- mean((y.pred.cv-y.valid)^2)
#    }
#    i <- i+1
#  }
```

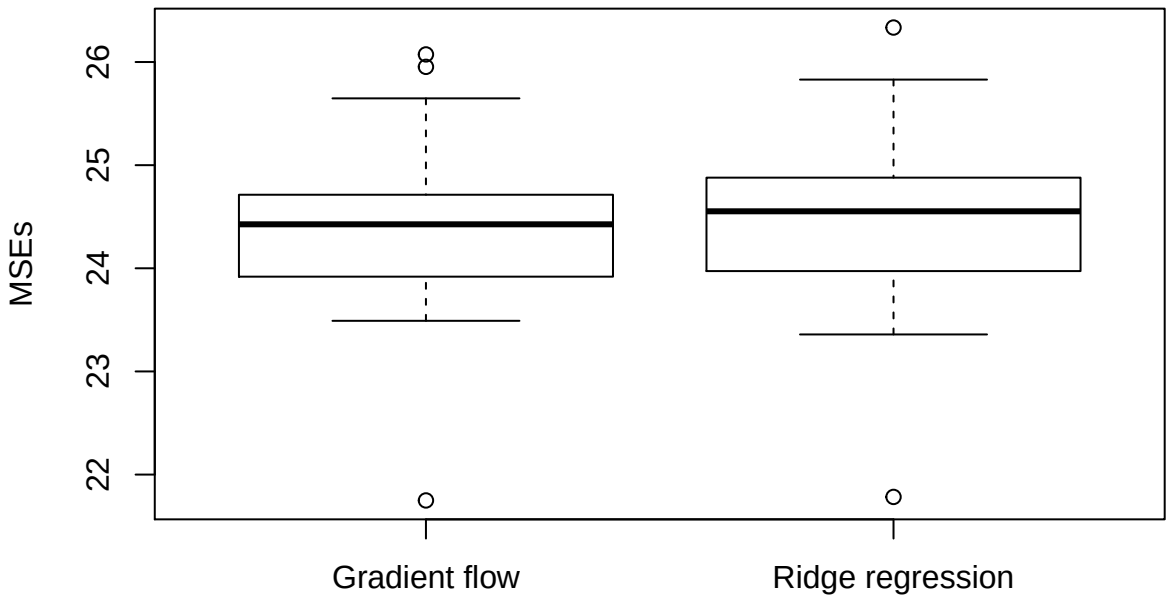
```
#
#  #Find best t
#  X.train <- X[train.vec,]
#  y.train <- y[train.vec]
#  X.test <- X[test.vec,]
#  y.test <- y[test.vec]
#  n.train <- nrow(X.train)
#
#  A <- t(X.train)%*%X.train/n.train
#  beta.train <- (diag(p)-expm(-A*t))%*%solve(t(X.train)%*%
#                                              X.train)%*%t(X.train)%*%y.train
#
#  y.pred <- X.test %*% beta.train
#  t.mse[r,1] <- mean((y.pred-y.test)^2)
#  t.mse[r,2] <- t.min
# }

# lambda.min
plot(log(lambdas),rowMeans(cv.error.ridge),main=
"ridge regression",xlab=bquote(log(lambda)),ylab="Validation MSE")
abline(v=log(lambda.min))
```




```
t.min  <- t_vec[which.min(rowMeans(cv.error.grad))]
plot(t_vec,rowMeans(cv.error.grad),main="Gradient flow",
     xlab=bquote(T),ylab="Validation MSE")
abline(v=t.min,lty=2,col="blue")
```




```
boxplot((t.mse[,1]),(ridge.mse[,1]),names=c("Gradient flow","Ridge regression"),ylab="MSEs")
```




```
print(paste("The average MSE of gradient flow is ",mean(t.mse[,1]),sep=""))
```

```
## [1] "The average MSE of gradient flow is 24.3963938089297"
```

```
print(paste("The average MSE of Ridge regression is ",mean(ridge.mse[,1]),sep=""))
```

```
## [1] "The average MSE of Ridge regression is 24.4728499693433"
```

Question 2.

(a) Denote $\mu_i := \begin{bmatrix}\mu_{A,i}\\ \mu_{B,i}\end{bmatrix} \sim N_2(\mathbf{0}, \Sigma)$ and $z := \begin{bmatrix}z_{A,i}\\ z_{B,i}\end{bmatrix}$. Then, we know that

$$\mu_i | z_i \sim N((\Sigma^{-1}+1)^{-1} z_i, (\Sigma^{-1}+1)^{-1}).$$

Now, we need to estimate the posterior covariance matrix $\Sigma_p := (Sigma^{-1}+1)^{-1}$. Note that, since $z_i \sim N(0, I+\Sigma)$ so that

$$V = \sum_{i=1}^{n} z_i z_i^T \sim W_2(\Sigma + I, n),$$

where $W_p(\cdot)$ is a Wishart distribution. Thus,

$$V^{-1} \sim W_2^{-1}((\Sigma+I)^{-1}, n),$$

where $W_p^{-1}(\cdot)$ is an inverse Wishart distribution. We know that

$$E(V) = \frac{1}{n-p-1}(\Sigma+I)^{-1} = \frac{1}{n-3}(\Sigma+I)^{-1}.$$

Then we can derive an estimator for $A := (\Sigma+I)^{-1}$ as

$$\hat{A} = (n-3)V = (n-3)\left(\sum_{i=1}^{n} z_i^T z_i\right)^{-1}.$$

Rewrite $\Sigma_p = (I+\Sigma^{-1})^{-1} = (I+\Sigma-I)(\Sigma+1)^{-1} = I - A$, then the estimator of Σ_p is

$$\hat{\Sigma}_p = I - (n-3)\left(\sum_{i=1}^{n} z_i z_i^T\right)^{-1}.$$

Then, the Empirical Bayes estimate of the posterior mean is given by

$$\hat{E}(\mu_i | z_i) = \left[I - (n-3)\left(\sum_{i=1}^{n} z_i z_i^T\right)^{-1}\right] z_i.$$

(b) A sample of 250 was generated from the following model:

$$z_t \equiv \begin{bmatrix}z_{1,t}\\ z_{2,t}\end{bmatrix} = \begin{bmatrix}\mu_{1,t}\\ \mu_{2,t}\end{bmatrix} + \begin{bmatrix}\varepsilon_{1,t}\\ \varepsilon_{2,t}\end{bmatrix} \equiv \mu_t + \varepsilon_t,$$

where $\mu \sim N_2\left(\mathbf{0}, \begin{bmatrix}1 & -0.5\\ -0.5 & 1\end{bmatrix}\right)$. Consider the following two cases:

(1) $\varepsilon \sim N(0, I)$, and

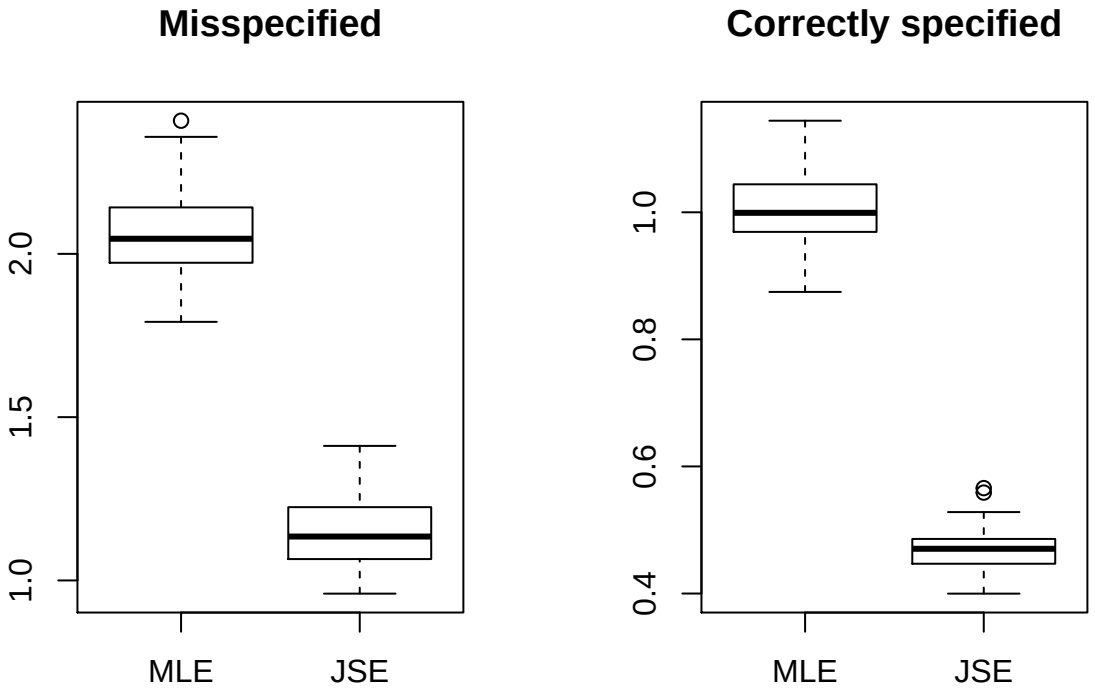
(2) $\varepsilon_t \sim N(0, \mathrm{diag}(\sigma_{1,t}^2, \sigma_{2,t}^2))$ follows an log-ARCH(1) model:

$$\log\begin{bmatrix}\sigma_{1,t}^2\\ \sigma_{2,t}^2\end{bmatrix} = 0.05 + \begin{bmatrix}0.75\\ 0.4\end{bmatrix} \log\begin{bmatrix}\varepsilon_{1,t-1}^2\\ \varepsilon_{2,t-1}^2\end{bmatrix}.$$

100 replications were done:

```
# n <- 250
# nrep <- 100
# library("mvtnorm")
# library("TSA")
#
# mse_mle <- array(dim=nrep)
# mse_JS <- array(dim=nrep)
#
# for(r in 1:nrep){
# mu <- rmvnorm(n,c(0,0),array(c(1,-0.5,-0.5,1),dim=c(2,2)))
# ep1_sigma <- exp(garch.sim(c(0.05,0.75),c(0),n=n))
# ep2_sigma <- exp(garch.sim(c(0.05,0.4),c(0),n=n))
# par(mfrow=c(2,1))
# plot(ep1_sigma,type="l",ylab=bquote(sigma[1~","~t]),xlab="Time")
# plot(ep2_sigma,type="l",ylab=bquote(sigma[2~","~t]),xlab="Time")
# ep1 <- rnorm(rep(0,n),sqrt(ep1_sigma))
# ep2 <- rnorm(rep(0,n),sqrt(ep2_sigma))
# z <- mu + array(c(ep1,ep2),dim=c(n,2))
#
# par(mfrow=c(1,2))
# plot(z[,1],type="l",ylab=bquote(z[1~","~t]),xlab="Time")
# plot(z[,2],type="l",ylab=bquote(z[2~","~t]),xlab="Time")
#
# #Task: Estimate all means
# #Traditional method
# mu_mle <- z
# mse_mle[r] <- mean((mu_mle-mu)^2)
#
# #JS
# z
# mu_JS <- t((diag(2)-(n-3)*solve(t(z)%*%z))%*%t(z))
# mse_JS[r] <- mean((mu_JS-mu)^2)
#
# }
#
#
#
#
#
# #Without misspecification
#
# mse_mle2 <- array(dim=nrep)
# mse_JS2 <- array(dim=nrep)
#
# for(r in 1:nrep){
#   mu <- rmvnorm(n,c(0,0),array(c(1,-0.5,-0.5,1),dim=c(2,2)))
#   par(mfrow=c(2,1))
#   plot(ep1_sigma,type="l",ylab=bquote(sigma[1~","~t]),xlab="Time")
#   plot(ep2_sigma,type="l",ylab=bquote(sigma[2~","~t]),xlab="Time")
#   ep1 <- rnorm(n,0,1)
#   ep2 <- rnorm(n,0,1)
#   z <- mu + array(c(ep1,ep2),dim=c(n,2))
#
```

```
#   par(mfrow=c(2,1))
#   plot(z[,1],type="l",ylab=bquote(z[1~","~t]),xlab="Time")
#   plot(z[,2],type="l",ylab=bquote(z[2~","~t]),xlab="Time")
#
#   #Task: Estimate all means
#   #Traditional method
#   mu_mle <- z
#   mse_mle2[r] <- mean((mu_mle-mu)^2)
#
#   #JS
#   z
#   mu_JS <- t((diag(2)-(n-3)*solve(t(z)%*%z))%*%t(z))
#   mse_JS2[r] <- mean((mu_JS-mu)^2)
#
# }

par(mfrow=c(1,2))
boxplot(mse_mle,mse_JS,names=c("MLE","JSE"),main="Misspecified")
boxplot(mse_mle2,mse_JS2,names=c("MLE","JSE"),main="Correctly specified")
```




```
effeiciency.misspec <- mean(mse_mle)/mean(mse_JS)
effeiciency.misspec2 <- mean(mse_mle2)/mean(mse_JS2)
effeiciency.misspec
```

```
## [1] 1.796269
```

```
effeiciency.misspec2
```

```
## [1] 2.134464
```

```
print(paste("The MSE ratio of MLE to JSE when the model is misspecified and the model is correct are res
```

```
## [1] "The MSE ratio of MLE to JSE when the model is misspecified and the model is correct are respecti
```

(c) We can observe that when the model is misspecified, the MSE for both MLE and JSE are both larger than that when the model is correct. In both cases, JSE performed much better than MLE from the boxplot. Comparing the MSE ratio of misspecified and correct model (1.8 vs 2.13), the ratio in the correct model is higher. Here is a summary:

- When the model is correctly specified, the JSE performs much better than MLE.
- When the model is incorrectly specified, the MSE will be larger on average. The JSE still performs better than MLE though the efficiency may be affected by the misclassification.