# Assignment 1

*David Fleischer – 260396047*

*Last Update: 20 January, 2018*

## DON'T FORGET TO UNCOMMENT Q4

## Question 1

From our definitions of $\tilde{X}$ and $\tilde{Y}$

$$\begin{aligned}
\tilde{X} &= X_{-1} - \mathbf{1}_n \bar{x}^T \\
\tilde{Y} &= Y - \mathbf{1}_n^T \bar{Y},
\end{aligned}$$

we find

$$\begin{aligned}
\hat{\beta}_{-1} &= \underset{\beta \in \mathbb{R}^{p-1}}{\arg\min} \ \|\tilde{Y} - \tilde{X}\beta\|_2^2 \\
&= \underset{\beta \in \mathbb{R}^{p-1}}{\arg\min} \ \|Y - \mathbf{1}_n \bar{Y} - \left(X_{-1} - \mathbf{1}_n \bar{x}^T\right)\beta_{-1}\|_2^2 \\
&= \underset{\beta \in \mathbb{R}^{p-1}}{\arg\min} \ \|Y - X_{-1}\beta_{-1} - \mathbf{1}_n \left(\bar{Y} - \bar{x}^T \beta_{-1}\right)\|_2^2 \\
&= \underset{\beta \in \mathbb{R}^{p-1}}{\arg\min} \ \|Y - X_{-1}\beta_{-1} - \mathbf{1}_n \beta_1\|_2^2 \quad \text{(by definition of } \beta_1 \text{ above)} \\
&= \underset{\beta \in \mathbb{R}^{p-1}}{\arg\min} \ \|Y - [\mathbf{1}_n, \, X_{-1}] \, [\beta_1, \, \beta_{-1}]\|_2^2 \\
&\equiv \underset{\beta \in \mathbb{R}^{p-1}}{\arg\min} \ \|Y - X\beta\|_2^2.
\end{aligned}$$

Therefore, if $\hat{\beta} = \left(\hat{\beta}_1, \, \hat{\beta}_{-1}^T\right)^T \in \mathbb{R}^p$ and

$$\hat{\beta}_1 = \bar{Y} - \bar{x}^T \hat{\beta}_{-1},$$

then $\hat{\beta}$ also solves the uncentered problem

$$\hat{\beta} \equiv \left(\hat{\beta}_1, \, \hat{\beta}_{-1}^T\right)^T = \underset{\beta \in \mathbb{R}^{p}}{\arg\min} \ \|Y - X\beta\|_2^2,$$

as desired.

## Question 2

### (a)

Define our objective function $f : \mathbb{R}^p \to \mathbb{R}$ by

$$
\begin{aligned}
f(\beta) &= \|\tilde{Y} - \tilde{X}\beta\|_2^2 + \lambda\|\beta\|_2^2 \\
&= \left(\tilde{Y} - \tilde{X}\beta\right)^T \left(\tilde{Y} - \tilde{X}\beta\right)^T + \lambda\beta^T\beta \\
&= \tilde{Y}^T\tilde{Y} - \tilde{Y}^T\tilde{X}\beta - \beta^T\tilde{X}^T\tilde{Y} + \beta^T\tilde{X}^T\tilde{X}\beta + \lambda\beta^T\beta \\
&= \tilde{Y}^T\tilde{Y} - 2\beta^T\tilde{X}^T\tilde{Y} + \beta^T\tilde{X}^T\tilde{X}\beta + \lambda\beta^T\beta.
\end{aligned}
$$

Therefore, by taking the gradient we find

$$\nabla f(\beta) = -2\tilde{X}^T\tilde{Y} + 2\tilde{X}^T\tilde{X}\beta + 2\lambda\beta,$$

as desired.

## (b)

The Hessian $\nabla^2 f(\beta)$ is given by

$$\nabla^2 f(\beta) = 2\tilde{X}^T\tilde{X} + 2\lambda\mathbb{I}_{p-1},$$

where $\mathbb{I}_{p-1}$ is the $(p-1)\times(p-1)$ identity matrix. Note that $2\tilde{X}^T\tilde{X} \in \mathbb{S}_+^{p-1}$ (positive semi-definite) and, for $\lambda > 0$, we have $2\lambda\mathbb{I}_{p-1} \in \mathbb{S}_{++}^{p-1}$ (positive definite). Therefore, for all nonzero vectors $v \in \mathbb{R}^{p-1}$,

$$
\begin{aligned}
v^T\nabla^2 f(\beta)v &= v^T\left(2\tilde{X}^T\tilde{X} + 2\lambda\mathbb{I}_{p-1}\right)v \\
&= 2v^T\tilde{X}^T\tilde{X}v + 2\lambda v^T\mathbb{I}_{p-1}v \\
&= 2\left(\underbrace{\|\tilde{X}v\|_2^2}_{\geq 0} + \underbrace{\lambda\|v\|_2^2}_{>0 \text{ when } \lambda>0}\right) \\
&> 0.
\end{aligned}
$$

Hence,

$$\nabla^2 f(\beta) = 2\tilde{X}^T\tilde{X} + 2\lambda\mathbb{I}_{p-1} \in \mathbb{S}_{++}^{p-1},$$

and so $f$ must be strictly convex in $\beta$.

## (c)

Suppose a strictly convex function $f$ is globally minimized at distinct points $x$ and $y$. By strict convexity

$$\forall\, t \in (0,1) \quad f(tx + (1-t)y) < tf(x) + (1-t)f(y).$$

Since $f$ is minimized at both $x$ and $y$ we have $f(x) = f(y)$, so

$$f(tx + (1-t)y) < tf(x) + (1-t)f(x) = f(x).$$

However, this implies that the point $z = tx + (1-t)y$ yields a value of $f$ even *smaller* than at $x$, contradicting our assumption that $x$ is a global minimizer. Therefore, strict convexity implies that the global minimizer must be unique, and so for $\lambda > 0$, we are guaranteed that the above solution will be the unique solution to our penalized least squares problem.

## (d)

To write our function computing the ridge coefficients we first set $\nabla f(\beta) = 0$

$$\hat{\beta}_{-1}^{(\lambda)} = \left(\tilde{X}^T\tilde{X} + \lambda\mathbb{I}_{p-1}\right)^{-1}\tilde{X}^T\tilde{Y}.$$

For the purpose of computational efficiency we make use of the singular value decomposition of $\tilde{X}$

$$\tilde{X} = UDV^T,$$

for $U \in \mathbb{R}^{n\times n}$ and $V \in \mathbb{R}^{(p-1)\times(p-1)}$ both orthogonal matrices, $U^TU = \mathbb{I}_n$, $V^TV = \mathbb{I}_{p-1}$, and $D \in \mathbb{R}^{n\times(p-1)}$ a diagonal matrix with entries $\{d_j\}_{j=1}^{\min(n,\,p-1)}$ along the main diagonal and zero elsewhere. Hence,

$$\begin{aligned}
\hat{\beta}_{-1}^{(\lambda)} &= \left(\tilde{X}^T\tilde{X} + \lambda\mathbb{I}_{p-1}\right)^{-1}\tilde{X}^T\tilde{Y} \\
&= \left(\left(UDV^T\right)^T UDV^T + \lambda VV^T\right)^{-1}\left(UDV^T\right)^T\tilde{Y} \\
&= \left(VD^TU^TUDV^T + \lambda VV^T\right)^{-1}VD^TU^T\tilde{Y} \\
&= \left(V\left(D^TD + \lambda\mathbb{I}_{p-1}\right)V^T\right)^{-1}VD^TU^T\tilde{Y} \\
&= V\left(D^TD + \lambda\mathbb{I}_{p-1}\right)^{-1}V^TVD^TU^T\tilde{Y} \\
&= V\left(D^TD + \lambda\mathbb{I}_{p-1}\right)^{-1}D^TU^T\tilde{Y}.
\end{aligned}$$

Note that $D^TD + \lambda\mathbb{I}_{p-1}$ is a diagonal $(p-1)\times(p-1)$ matrix with entries $d_j^2 + \lambda$, $j = 1, ..., p-1$, and so the inverse $\left(D^TD + \lambda\mathbb{I}_{p-1}\right)^{-1}$ must also be diagonal with entries $\left(d_j^2 + \lambda\right)^{-1}$, $j = 1, ..., p-1$. We exploit this to avoid performing a matrix inversion in our function. For brevity, let

$$D^* = \left(D^TD + \lambda I_{p-1}\right)^{-1}D^T,$$

so that

$$\hat{\beta}^{(\lambda)} = VD^*U^T\tilde{Y}.$$

We present a function written in `R` performing such calculations below.

```
ridge_coef <- function(X, y, lam) {
  Xm1 <- X[,-1] # remove leading column of 1's marking the intercept

  ytilde <- y - mean(y) # center response
  xbar <- colMeans(Xm1) # find predictor means
  Xtilde <- sweep(Xm1, 2, xbar) # center each predictor according to its mean

  # compute the SVD on the centered design matrix
  Xtilde_svd <- svd(Xtilde)
```

```
  U <- Xtilde_svd$u
  d <- Xtilde_svd$d
  V <- Xtilde_svd$v

  # compute the inverse (D^T D + lambda I_{p-1})^{-1} D^T
  Dstar <- diag(d/(d^2 + lam))

  # compute ridge coefficients
  b <- V %*% (Dstar %*% crossprod(U, ytilde)) # slopes
  b1 <- mean(y) - crossprod(xbar, b) # intercept
  list(b1 = b1, b = b)
}
```

Note the choice to use `V %*% (Dstar %*% crossprod(U, ytilde))` to compute the matrix product $VD^*U^T\tilde{Y}$ as opposed to (the perhaps more intuitive) `V %*% Dstar %*% t(U) %*% ytilde`. Such a choice is empirically justified in an appendix.

## (e)

We first take the expectation of $\hat{\beta}_{-1}^{(\lambda)}$

$$\begin{aligned}\mathbb{E}\left[\hat{\beta}_{-1}^{(\lambda)}\right] &= \mathbb{E}\left[\left(\tilde{X}^T\tilde{X}+\lambda\mathbb{I}_{p-1}\right)^{-1}\tilde{X}^T\tilde{Y}\right]\\ &= \left(\tilde{X}^T\tilde{X}+\lambda\mathbb{I}_{p-1}\right)^{-1}\tilde{X}^T\mathbb{E}\left[\tilde{Y}\right]\\ &= \left(\tilde{X}^T\tilde{X}+\lambda\mathbb{I}_{p-1}\right)^{-1}\tilde{X}^T\tilde{X}\beta_{-1}\end{aligned}$$

If $p >> n$ then using the SVD on $\tilde{X}$ may yield some speed improvements, that is, with $\tilde{X} = UDV^T$ as above, we find

$$\begin{aligned}\mathbb{E}\left[\hat{\beta}_{-1}^{(\lambda)}\right] &= \left(\tilde{X}^T\tilde{X}+\lambda\mathbb{I}_{p-1}\right)^{-1}\tilde{X}^T\tilde{X}\beta_{-1}\\ &= V\left(D^TD+\lambda\mathbb{I}_{p-1}\right)^{-1}D^TDV^T\beta_{-1}\\ &= VD^*V^T\beta_{-1}\end{aligned}$$

where $D^*$ is a diagonal $\min(n, p-1)\times\min(n, p-1)$ matrix with diagonal entries $\left\{\frac{d_j^2}{d_j^2+\lambda}\right\}_{j=1}^{\min(n,p-1)}$ and zero elsewhere. [1]

We next compute the variance of our centered ridge estimates

$$\begin{aligned}\mathrm{Var}\left(\hat{\beta}_{-1}^{(\lambda)}\right) &= \mathrm{Var}\left(\left(\tilde{X}^T\tilde{X}+\lambda\mathbb{I}_{p-1}\right)^{-1}\tilde{X}^T\tilde{Y}\right)\\ &= \left(\tilde{X}^T\tilde{X}+\lambda\mathbb{I}_{p-1}\right)^{-1}\tilde{X}^T\mathrm{Var}\left(\tilde{Y}\right)\left(\left(\tilde{X}^T\tilde{X}+\lambda\mathbb{I}_{p-1}\right)^{-1}\tilde{X}^T\right)^T\\ &= \left(\tilde{X}^T\tilde{X}+\lambda\mathbb{I}_{p-1}\right)^{-1}\tilde{X}^T\mathrm{Var}\left(\tilde{Y}\right)\tilde{X}\left(\tilde{X}^T\tilde{X}+\lambda\mathbb{I}_{p-1}\right)^{-1}\\ &= \sigma_*^2\left(\tilde{X}^T\tilde{X}+\lambda\mathbb{I}_{p-1}\right)^{-1}\tilde{X}^T\tilde{X}\left(\tilde{X}^T\tilde{X}+\lambda\mathbb{I}_{p-1}\right)^{-1}\end{aligned}$$

[1] Benchmarks are provided in an appendix for the cases of large $n$, large $p$, and $n \approx p$.

as desired. We once again may be interested in applying the SVD on $\tilde{X}$ as we had done before. Such a decomposition gives us a more concise solution

$$\text{Var}\left(\hat{\beta}_{-1}^{(\lambda)}\right) = VD^{**}V^T$$

where $D^{**}$ is a diagonal $\min(n, p-1) \times \min(n, p-1)$ matrix with diagonal entries $\left\{\frac{d_j^2}{\left(d_j^2+\lambda\right)^2}\right\}_{j=1}^{\min(n,p-1)}$ and zero elsewhere.

We now wish to perform a simulation study to estimate our theoretical values $\mathbb{E}\left[\hat{\beta}_{-1}^{(\lambda)}\right]$ and $\text{Var}\left(\hat{\beta}_{-1}^{(\lambda)}\right)$. For readability we first define functions computing the theoretical mean and variance according to our above expressions.

```
ridge_coef_params <- function(X, lam, beta, sigma) {
  n <- nrow(X); p <- ncol(X)
  betam1 <- beta[-1] # remove intercept term
  Xm1 <- X[,-1] # remove leading column of 1's in our design matrix

  xbar <- colMeans(Xm1) # find prector means
  Xtilde <- sweep(Xm1, 2, xbar) # center each predictor according to its mean

  if (n >= p) {
    I <- diag(p - 1)
    inv <- solve(crossprod(Xtilde) + lam * I)

    b <- solve(crossprod(Xtilde) + lam * I) %*% (crossprod(Xtilde) %*% betam1)
    vcv <- sigma^2 * inv %*% crossprod(Xtilde) %*% inv
    list(b = b, vcv = vcv)

  } else {
    # compute SVD on the centered design matrix
    Xtilde_svd <- svd(Xtilde)
    d <- Xtilde_svd$d
    V <- Xtilde_svd$v

    Dstar <- diag(d^2/(d^2 + lam))
    Dstar2 <- diag(d^2/(d^2 + lam)^2)

    b <- V %*% (Dstar %*% crossprod(V, betam1))
    vcv <- V %*% tcrossprod(Dstar2, V)
    list(b = b, vcv = vcv)
  }
}
```

We may now perform our simulation.

```
set.seed(124)

# set parameters
nsims <- 1e3
n <- 25
p <- 7
lam <- 4
beta_star <- 1:p
```

```r
sigma_star <- 1

# generate fixed design matrix
X <- cbind(1, matrix(rnorm(n * (p - 1)), nrow = n))

# compute theoretical mean and variance
par_true <- ridge_coef_params(X, lam, beta_star, sigma_star)
b_true <- as.vector(par_true$b)
vcv_true <- par_true$vcv

# simulate ridge coefficients nsims times
# outputs a matrix with rows corresponding to coefficients
# and columns correspond to simulation number
b_hat <- replicate(nsims, {
  y <- X %*% beta_star + rnorm(n, 0, sigma_star)
  as.vector(ridge_coef(X, y, lam)$b)
})

# estimate variance of b1, ..., b_p estimates
vcv_hat <- var(t(b_hat))

# print estimated fused ridge coefficients vs. expected values
b <- rbind(rowMeans(b_hat), b_true)
rownames(b) <- c("b_hat", "b_true")
round(b, 4)
```

```
##          [,1]   [,2]   [,3]   [,4]   [,5]   [,6]
## b_hat  0.7861 1.6595 3.2916 3.8786 4.2007 6.3650
## b_true 0.7797 1.6636 3.2936 3.8779 4.2025 6.3689
```

```r
# print absolute error between estimated and true fused ridge variances
round(abs(vcv_true - vcv_hat), 4)
```

```
##        [,1]   [,2]   [,3]   [,4]   [,5]   [,6]
## [1,] 0.0010 0.0008 0.0013 0.0012 0.0008 0.0009
## [2,] 0.0008 0.0008 0.0009 0.0017 0.0011 0.0003
## [3,] 0.0013 0.0009 0.0012 0.0006 0.0015 0.0015
## [4,] 0.0012 0.0017 0.0006 0.0014 0.0005 0.0001
## [5,] 0.0008 0.0011 0.0015 0.0005 0.0007 0.0012
## [6,] 0.0009 0.0003 0.0015 0.0001 0.0012 0.0013
```

We see that the empirical sample estimates are very close to their theoretical values, as expected.

# Question 3

Prior to writing our cross-validation function we create some helper functions for the sake of readability

```r
ridge_fit <- function(X, y, lam) {
  # fully fit a ridge regression model given predictors, response, and penalty

  b <- unlist(ridge_coef(X, y, lam)) # extract coefficient estimates
  yhat <- X %*% b # fit a response estimate given fitted coefficients
  res <- sum((y - yhat)^2) # find prediction error
```

```
  list(X = X, y = y, lam = lam, coef = b, fit = yhat, res = res)
}
ridge_cv_lam <- function(X, y, lam, K) {
  # Helper function for ridge_cv()
  # perform K-fold cross-validation on the ridge regression
  # estimation problem over a single tuning parameter lam
  n <- nrow(X)

  if (K > n) {
    stop(paste0("K > ", n, "."))
  } else if (K < 2) {
    stop("K < 2.")
  }

  # groups to cross-validate over
  folds <- cut(1:nrow(X), breaks = K, labels = F)
  # get indices of training subset
  train_idxs <- lapply(1:K, function(i) !(folds %in% i))

  cv_err <- sapply(train_idxs, function(tis) {
    # train our model
    train_fit <- ridge_fit(X[tis,], y[tis], lam)

    # find observations needed for testing fits
    test_idx <- !((1:n) %in% tis)

    # extract fitted coefficients
    b <- train_fit$coef
    # fit data
    yhat <- X[test_idx,] %*% b
    # compute test error
    sum((y[test_idx] - yhat)^2)
  })
  # weighted average (according to group size, some groups may have
  # +/- 1 member depending on whether sizes divided unevenly) of
  # cross validation error for a fixed lambda
  sum((cv_err * table(folds)))/n
}
```

Then, our cross-validation function is as follows:

```
ridge_cv <- function(X, y, lam.vec, K) {
  # perform K-fold cross-validation on the ridge regression
  # estimation problem over tuning parameters given in lam.vec
  n <- nrow(X); p <- ncol(X); L <- length(lam.vec)

  cv.error <- sapply(1:L, function(i) ridge_cv_lam(X, y, lam.vec[i], K))

  # extract best tuning parameter and corresponding coefficient estimates
  best.lam <- lam.vec[cv.error == min(cv.error)]
  best.fit <- ridge_fit(X, y, best.lam)
  b1 <- best.fit$coef[1]
  b <- best.fit$coef[-1]
```

```r
  list(b1 = b1, b = b, best.lam = best.lam, cv.error = cv.error)
}
```

# Question 4

For this problem we first set some global libraries/functions

```r
library(doParallel)

rmvn <- function(n, p, mu = 0, S = diag(p)) {
  # generates n (potentially correlated) p-dimensional normal deviates
  # given mean vector mu and variance-covariance matrix S
  # NOTE: S must be a positive-semidefinite matrix
  Z <- matrix(rnorm(n * p), nrow = n, ncol = p) # generate iid normal deviates
  C <- chol(S)
  mu + Z %*% C # compute our correlated deviates
}
loss1 <- function(beta, b) sum((b - beta)^2)
loss2 <- function(X, beta, b) sum((X %*% (beta - b))^2)
```

and global parameters which remain constant across (a)-(d)

```r
set.seed(124)

# global parameters
nsims <- 50
n <- 100
lams <- 10^seq(-8, 8, 0.5)
sigma_star <- sqrt(1/2)
```

## (a)

```r
# set parameters
p <- 50
theta <- 0.5

# generate data
beta_star <- rnorm(p, 0, sigma_star)
SIGMA <- outer(1:(p - 1), 1:(p - 1), FUN = function(a, b) theta^abs(a - b))
X <- cbind(1, rmvn(n, p - 1, 0, SIGMA))

# simulation
pt <- proc.time()
registerDoParallel(cores = 4)

sim <- foreach(1:nsims, .combine = cbind) %dopar% {
  y <- X %*% beta_star + rnorm(n, 0, sigma_star)

  ols_fit <- ridge_fit(X, y, 0)
  k5_fit <- ridge_cv(X, y, lam.vec = lams, K = 5)
  k10_fit <- ridge_cv(X, y, lam.vec = lams, K = 10)
```

```r
  kn_fit <- ridge_cv(X, y, lam.vec = lams, K = n)

  coef_list <- list(OLS = ols_fit$coef,
                    k5  = c(k5_fit$b1, k5_fit$b),
                    k10 = c(k10_fit$b1, k10_fit$b),
                    kn  = c(kn_fit$b1, kn_fit$b))
  l1 <- sapply(coef_list, function(b) loss1(beta_star, b))
  l2 <- sapply(coef_list, function(b) loss2(X, beta_star, b))
  list(l1, l2)
}

sim_loss <- lapply(1:nrow(sim),
                   function(i) sapply(sim[i,], function(s) s))
names(sim_loss) <- c("Loss 1", "Loss 2")

sim_means <- t(sapply(sim_loss, function(s) rowMeans(s)))
sim_se <- t(sapply(sim_loss,
                 function(s) apply(s, 1, function(x) sd(x)/sqrt(length(x)))))

proc.time() - pt
```

```
##    user  system elapsed
##  11.299   0.183  12.661
```

```r
# report results
round(sim_means, 4)
```

```
##             OLS      k5     k10      kn
## Loss 1   0.5198  0.5241  0.5241  0.5184
## Loss 2 23.7197 23.5876 23.5876 23.6716
```

```r
round(sim_se, 4)
```

```
##           OLS     k5    k10     kn
## Loss 1 0.2113 0.1891 0.1891 0.2095
## Loss 2 3.1801 2.5653 2.5653 3.1254
```

## (b)

```r
# set parameters
p <- 50
theta <- 0.9

# generate data
beta_star <- rnorm(p, 0, sigma_star)
SIGMA <- outer(1:(p - 1), 1:(p - 1), FUN = function(a, b) theta^abs(a - b))
X <- cbind(1, rmvn(n, p - 1, 0, SIGMA))

# simulation
pt <- proc.time()
registerDoParallel(cores = 4)

sim <- foreach(1:nsims, .combine = cbind) %dopar% {
  y <- X %*% beta_star + rnorm(n, 0, sigma_star)
```

```r
  ols_fit <- ridge_fit(X, y, 0)
  k5_fit <- ridge_cv(X, y, lam.vec = lams, K = 5)
  k10_fit <- ridge_cv(X, y, lam.vec = lams, K = 10)
  kn_fit <- ridge_cv(X, y, lam.vec = lams, K = n)

  coef_list <- list(OLS = ols_fit$coef,
                    k5  = c(k5_fit$b1, k5_fit$b),
                    k10 = c(k10_fit$b1, k10_fit$b),
                    kn  = c(kn_fit$b1, kn_fit$b))
  l1 <- sapply(coef_list, function(b) loss1(beta_star, b))
  l2 <- sapply(coef_list, function(b) loss2(X, beta_star, b))
  list(l1, l2)
}

sim_loss <- lapply(1:nrow(sim),
                   function(i) sapply(sim[i,], function(s) s))
names(sim_loss) <- c("Loss 1", "Loss 2")

sim_means <- t(sapply(sim_loss, function(s) rowMeans(s)))
sim_se <- t(sapply(sim_loss,
                 function(s) apply(s, 1, function(x) sd(x)/sqrt(length(x)))))

proc.time() - pt
```

```
##    user  system elapsed
##  22.331   0.330  11.751
```

```r
# report results
round(sim_means, 4)
```

```
##            OLS      k5     k10      kn
## Loss 1  4.2982  3.4987  3.8023  4.1696
## Loss 2 25.3519 22.1985 23.1262 24.9134
```

```r
round(sim_se, 4)
```

```
##           OLS     k5    k10     kn
## Loss 1 0.0417 0.2872 0.0164 0.0665
## Loss 2 1.4346 1.6328 2.5605 1.5040
```

## (c)

```r
# set parameters
p <- 200
theta <- 0.5

# generate data
beta_star <- rnorm(p, 0, sigma_star)
SIGMA <- outer(1:(p - 1), 1:(p - 1), FUN = function(a, b) theta^abs(a - b))
X <- cbind(1, rmvn(n, p - 1, 0, SIGMA))

# simulation
pt <- proc.time()
registerDoParallel(cores = 4)
```

```r
sim <- foreach(1:nsims, .combine = cbind) %dopar% {
  y <- X %*% beta_star + rnorm(n, 0, sigma_star)

  ols_fit <- ridge_fit(X, y, 0)
  k5_fit <- ridge_cv(X, y, lam.vec = lams, K = 5)
  k10_fit <- ridge_cv(X, y, lam.vec = lams, K = 10)
  kn_fit <- ridge_cv(X, y, lam.vec = lams, K = n)

  coef_list <- list(OLS = ols_fit$coef,
                    k5  = c(k5_fit$b1, k5_fit$b),
                    k10 = c(k10_fit$b1, k10_fit$b),
                    kn  = c(kn_fit$b1, kn_fit$b))
  l1 <- sapply(coef_list, function(b) loss1(beta_star, b))
  l2 <- sapply(coef_list, function(b) loss2(X, beta_star, b))
  list(l1, l2)
}

sim_loss <- lapply(1:nrow(sim),
                   function(i) sapply(sim[i,], function(s) s))
names(sim_loss) <- c("Loss 1", "Loss 2")

sim_means <- t(sapply(sim_loss, function(s) rowMeans(s)))
sim_se <- t(sapply(sim_loss,
                 function(s) apply(s, 1, function(x) sd(x)/sqrt(length(x)))))

proc.time() - pt
```

```
##    user  system elapsed
##  66.517   1.624  59.927
```

```r
# report results
round(sim_means, 4)
```

```
##              OLS      k5     k10      kn
## Loss 1 47.5307 47.3791 47.3794 47.3804
## Loss 2 57.7869 57.7869 57.7628 57.7194
```

```r
round(sim_se, 4)
```

```
##            OLS     k5    k10     kn
## Loss 1 0.2194 0.2719 0.2716 0.2714
## Loss 2 6.9140 6.9140 6.9382 6.9950
```

### (d)

```r
# set parameters
p <- 200
theta <- 0.9

# generate data
beta_star <- rnorm(p, 0, sigma_star)
SIGMA <- outer(1:(p - 1), 1:(p - 1), FUN = function(a, b) theta^abs(a - b))
X <- cbind(1, rmvn(n, p - 1, 0, SIGMA))
```

```r
# simulation
pt <- proc.time()
registerDoParallel(cores = 4)

sim <- foreach(1:nsims, .combine = cbind) %dopar% {
  y <- X %*% beta_star + rnorm(n, 0, sigma_star)

  ols_fit <- ridge_fit(X, y, 0)
  k5_fit <- ridge_cv(X, y, lam.vec = lams, K = 5)
  k10_fit <- ridge_cv(X, y, lam.vec = lams, K = 10)
  kn_fit <- ridge_cv(X, y, lam.vec = lams, K = n)

  coef_list <- list(OLS = ols_fit$coef,
                    k5  = c(k5_fit$b1, k5_fit$b),
                    k10 = c(k10_fit$b1, k10_fit$b),
                    kn  = c(kn_fit$b1, kn_fit$b))
  l1 <- sapply(coef_list, function(b) loss1(beta_star, b))
  l2 <- sapply(coef_list, function(b) loss2(X, beta_star, b))
  list(l1, l2)
}

sim_loss <- lapply(1:nrow(sim),
                   function(i) sapply(sim[i,], function(s) s))
names(sim_loss) <- c("Loss 1", "Loss 2")

sim_means <- t(sapply(sim_loss, function(s) rowMeans(s)))
sim_se <- t(sapply(sim_loss,
                 function(s) apply(s, 1, function(x) sd(x)/sqrt(length(x)))))
proc.time() - pt
```

```
##    user  system elapsed
## 114.124   3.091  63.202
```

```r
# report results
round(sim_means, 4)
```

```
##            OLS      k5     k10      kn
## Loss 1 46.8483 46.6573 46.5641 46.8013
## Loss 2 49.8882 48.5122 47.9379 49.4292
```

```r
round(sim_se, 4)
```

```
##           OLS     k5    k10     kn
## Loss 1 0.6355 0.5398 0.4467 0.3958
## Loss 2 2.5764 3.2858 2.7114 2.3688
```

# Question 5

## (a)

Taking the gradient of our objective function $g$ with respect to coefficient vector $\beta$ yields

$$\begin{aligned}
\nabla_{\beta} g(\beta, \sigma^2) &= \nabla_{\beta} \left( \frac{n}{2} \left( \log \sigma^2 \right) + \frac{1}{2\sigma^2} \|\tilde{Y} - \tilde{X}\beta\|_2^2 + \frac{\lambda}{2} \|\beta\|_2^2 \right) \\
&= \frac{1}{\sigma^2} \left( -\tilde{X}^T \tilde{Y} + \tilde{X}^T \tilde{X} \beta \right) + \lambda \beta,
\end{aligned}$$

while the gradient of $g$ with respect to $\sigma^2$ is given by

$$\begin{aligned}
\nabla_{\sigma^2} g(\beta, \sigma^2) &= \nabla_{\beta} \left( \frac{n}{2} \left( \log \sigma^2 \right) + \frac{1}{2\sigma^2} \|\tilde{Y} - \tilde{X}\beta\|_2^2 + \frac{\lambda}{2} \|\beta\|_2^2 \right) \\
&= \frac{n}{2\sigma^2} - \frac{1}{2\sigma^4} \|\tilde{Y} - \tilde{X}\beta\|_2^2.
\end{aligned}$$

as desired.

## (b)

## (c)

Let $\bar{\beta}$ be a solution to our maximum likelihood ridge estimation problem such that, for $\lambda > 0$, we have

$$\tilde{Y} - \tilde{X}\bar{\beta} = 0.$$

Since $\bar{\beta}$ is a solution it must satisfy our first order condition

$$\nabla_{\beta} g(\beta, \sigma^2) = \frac{1}{\sigma^2} \left( -\tilde{X}^T \tilde{Y} + \tilde{X}^T \tilde{X} \beta \right) + \lambda \beta = 0 \iff \frac{1}{\sigma^2} \left( \tilde{X}^T \left( -\tilde{Y} + \tilde{X}\beta \right) \right) + \lambda \beta = 0.$$

Thus, for such a solution $\bar{\beta}$ and $\lambda > 0$,

$$\begin{aligned}
0 &= \frac{1}{\sigma^2} \left( \tilde{X}^T \left( -\tilde{Y} + \tilde{X}\bar{\beta} \right) \right) + \lambda \bar{\beta} \\
&= \frac{1}{\sigma^2} \left( \tilde{X}^T \left( -\tilde{Y} + \tilde{Y} \right) \right) + \lambda \bar{\beta} \\
&= \lambda \bar{\beta} \\
\iff \bar{\beta} &= 0.
\end{aligned}$$

Similarly, using our second first order condition $\nabla_{\sigma^2} g(\beta, \sigma^2) = 0$, at $\beta = \bar{\beta}$,

$$\begin{aligned}
\nabla_{\sigma^2} g(\beta, \sigma^2) &= \frac{n}{2\sigma^2} - \frac{1}{2\sigma^4} \|\tilde{Y} - \tilde{X}\beta\|_2^2 \\
&= \frac{n}{2\sigma^2} - \frac{1}{2\sigma^4} \|\tilde{Y} - \tilde{X}\hat{\beta}\|_2^2 \\
&= \frac{n}{2\sigma^2} - \frac{1}{2\sigma^4} \|\tilde{Y} - \tilde{Y}\|_2^2 \\
&= \frac{n}{2\sigma^2} = 0
\end{aligned}$$

This conditions implies that either $n = 0$ or $\sigma^2 \to \infty$. Thus, no such global minimizer could exist.

### (d)

Solving our first order conditions

$$\begin{aligned}\frac{1}{\sigma^2}\left(\tilde{X}^T\left(-\tilde{Y}+\tilde{X}\bar{\beta}\right)\right)+\lambda\bar{\beta}=0\\ \frac{n}{2\sigma^2}-\frac{1}{\sigma^4}\|\tilde{Y}-\tilde{X}\beta\|_2^2=0,\end{aligned}$$

we find the maximum likelihood estimate $\hat{\beta}^{(\lambda,\,ML)}$ to be

$$\hat{\beta}^{(\lambda,\,ML)}=\left(\tilde{X}^T\tilde{X}+\sigma^2\lambda\mathbb{I}_{p-1}\right)^{-1}\tilde{X}^T\tilde{Y}.$$

and the maximum likelihood estimate $\hat{\sigma}^{2\,(\lambda,\,ML)}$ to be

$$\hat{\sigma}^{2\,(\lambda,\,ML)}=\frac{2}{n}\|\tilde{Y}-\tilde{X}\hat{\beta}^{(\lambda,\,ML)}\|_2^2$$

To compute such estimates we may use the following algorithm: Consider some fixed data set $\mathcal{D}=\{X,Y\}$ and a fixed tuning parameter $\lambda$.

(1) Center the data: Center each predictor by its mean $X\mapsto\tilde{X}$, center the response vector by its mean $Y\mapsto\tilde{Y}$.

(2) Have some initial proposal for the estimate $\hat{\sigma}_0^{2\,(\lambda,\,ML)}\in\mathbb{R}^+$.

(3) Compute an initial proposal for $\hat{\beta}_0^{(\lambda,\,ML)}$ based on $\hat{\sigma}_0^{2\,(\lambda,\,ML)}$.

(4) Update our variance estimate $\hat{\sigma}_i^{2\,(\lambda,\,ML)}$ using the previous estimate of $\hat{\beta}_{i-1}^{(\lambda,\,ML)}$.

(5) Update our coefficient estimate $\hat{\beta}^{(\lambda,\,ML)}$ using the new estimate of $\hat{\sigma}_i^{2\,(\lambda,\,ML)}$.

(6) Repeat steps (5)-(6) until some convergence criteria is met, say $\|\hat{\sigma}_i^{2\,(\lambda,\,ML)}-\hat{\sigma}_{i-1}^{2\,(\lambda,\,ML)}\|$ is small.

### (e)

```r
ridge_coef_mle <- function(X, y, lam, tol = 1e-5) {
  Xm1 <- X[,-1] # remove leading column of 1's marking the intercept

  ytilde <- y - mean(y) # center response
  xbar <- colMeans(Xm1) # find predictor means
  Xtilde <- sweep(Xm1, 2, xbar) # center each predictor according to its mean

  # compute the SVD on the centered design matrix
  Xtilde_svd <- svd(Xtilde)
  U <- Xtilde_svd$u
  d <- Xtilde_svd$d
  V <- Xtilde_svd$v

  # compute the inverse (D^T D + sigma^2 * lambda I_{p-1})^{-1} D^T
  #Dstar <- diag(d/(d^2 + lam))

  # # compute ridge coefficients
```

```
  # b <- V %*% (Dstar %*% crossprod(U, ytilde)) # slopes
  # b1 <- mean(y) - crossprod(xbar, b) # intercept
  # list(b1 = b1, b = b)
}
```

### (f)

## Question 6

### (a)

Consider our objective function

$$f(\beta) = \frac{1}{2}\|\tilde{Y} - \tilde{X}\beta\|_2^2 + \frac{\lambda_1}{2}\|\beta\|_2^2 + \frac{\lambda_2}{2}\sum_{j=2}^{p}(\beta_j - \beta_{j-1})^2$$

To show convexity we wish to show $\nabla^2 f(\beta) \in \mathbb{S}_+^{p-1}$. However, it's not immediately obvious how to take such a gradient with our fused sum terms $(b_j - \beta_{j-1})^2$. One way to get around this is to define vector $B \in \mathbb{R}^{p-1}$ given by

$$B = \begin{bmatrix} \beta_2 - \beta_1 \\ \vdots \\ \beta_p - \beta_{p-1} \end{bmatrix}$$

Then

$$\sum_{j=2}^{p}(\beta_j - \beta_{j-1})^2 = B^T B$$

In order to achieve our task of expressing the fused sum in terms of the vector $\beta$ we must next decompose $B$ into a product of $\beta$ and some matrix. To this end we define matrix $A \in \mathbb{R}^{(p-2)\times(p-1)}$ with entries -1 along the main diagonal and 1 along the upper diagonal, i.e.,

$$A = \begin{bmatrix} -1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & -1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -1 & 1 \end{bmatrix}$$

Then

$$\begin{aligned}\sum_{j=2}^{p}(\beta_j - \beta_{j-1})^2 &= B^T B \\ &= \beta^T A^T A \beta \\ &\equiv \|A\beta\|_2^2\end{aligned}$$

Therefore, our objective function can be expressed as

$$
\begin{aligned}
f(\beta) &= \frac{1}{2}\|\tilde{Y} - \tilde{X}\beta\|_2^2 + \frac{\lambda_1}{2}\|\beta\|_2^2 + \frac{\lambda_2}{2}\|A\beta\|_2^2 \\
&\equiv \frac{1}{2}\tilde{Y}^T\tilde{Y} - \beta^T\tilde{X}^T\tilde{Y} + \frac{1}{2}\beta^T\tilde{X}^T\tilde{X}\beta + \frac{\lambda_1}{2}\beta^T\beta + \frac{\lambda_2}{2}\beta^T A^T A\beta
\end{aligned}
$$

Hence

$$
\nabla f(\beta) = -\tilde{X}^T\tilde{Y} + \tilde{X}^T\tilde{X}\beta + \lambda_1\beta + \lambda_2 A^T A\beta
$$

admitting the Hessian

$$
\nabla^2 f(\beta) = \tilde{X}^T\tilde{X} + \lambda_1\mathbb{I}_{p-1} + \lambda_2 A^T A
$$

Recalling that a matrix multiplied with its transpose must always be positive semi-definite, we find $\tilde{X}^T X$ and $A^T A$ must be positive semi-definite. Thus, since $\lambda_1 > 0$, we find that our sum $\tilde{X}^T\tilde{X} + \lambda_1\mathbb{I}_{p-1} + \lambda_2 A^T A = \nabla^2 f(\beta)$ is positive semi-definite, and so $f(\beta)$ must be strictly convex, as desired.

## (b)

We first solve for $\hat{\beta}_{-1}^{(\lambda_1,\,\lambda_2)}$ in (a) by setting $\nabla f(\beta) = 0$

$$
\begin{aligned}
0 &= -\tilde{X}^T\tilde{Y} + \tilde{X}^T\tilde{X}\beta + \lambda_1\beta + \lambda_2 A^T A\beta \\
\tilde{X}^T\tilde{Y} &= \left(\tilde{X}^T\tilde{X} + \lambda_1\mathbb{I}_{p-1} + \lambda_2 A^T A\right)\beta \\
\implies \hat{\beta}_{-1}^{(\lambda_1,\,\lambda_2)} &= M\tilde{X}^T\tilde{Y}
\end{aligned}
$$

where we have set $M = \left(\tilde{X}^T\tilde{X} + \lambda_1\mathbb{I}_{p-1} + \lambda_2 A^T A\right)^{-1}$ for brevity. Therefore

$$
\begin{aligned}
\mathbb{E}\left[\hat{\beta}_{-1}^{(\lambda_1,\,\lambda_2)}\right] &= \mathbb{E}\left[M\tilde{X}^T\tilde{Y}\right] \\
&= M\tilde{X}^T\mathbb{E}\left[\tilde{Y}\right] \\
&= M\tilde{X}^T\beta_{*,\,-1}
\end{aligned}
$$

and

$$
\begin{aligned}
\mathrm{Var}\left(\hat{\beta}_{-1}^{(\lambda_1,\,\lambda_2)}\right) &= \mathrm{Var}\left(M\tilde{X}^T Y\right) \\
&= M\tilde{X}^T\mathrm{Var}\left(\tilde{Y}\right)\tilde{X}M^T \\
&= \sigma_*^2 M\tilde{X}^T\tilde{X}M^T
\end{aligned}
$$

as desired. We now perform our fused ridge simulation study to test the theoretical values with some empirical estimates. We first define our fused ridge coefficient estimation function (as well as functions permitting us to easily compute the theoretical means and variances of the fused ridge problem)

```r
fused_ridge_coef <- function(X, y, lam1, lam2) {
  n <- nrow(X); p <- ncol(X)
  Xm1 <- X[,-1] # remove leading column of 1's marking the intercept

  ytilde <- y - mean(y) # center response
  xbar <- colMeans(Xm1) # find predictor means
  Xtilde <- sweep(Xm1, 2, xbar) # center each predictor according to its mean

  I <- diag(p - 1)
  UD <- cbind(rep(0, p - 2), diag(p - 2)) # upper diagonal matrix
  J <- -1 * cbind(diag(p - 2), rep(0, p - 2)) # diag (p - 2)*(p - 1) matrix
  A <- J + UD

  M <- solve(crossprod(Xtilde) + lam1 * I + lam2 * crossprod(A))
  b <- M %*% crossprod(Xtilde, y)
  b0 <- mean(y) - crossprod(xbar, b)
  return(list(b0 = b0, b = b))
}
fused_ridge_coef_params <- function(X, lam1, lam2, beta, sigma) {
  # omits intercept term b0
  # returns theoretical means and variances for the fused ridge problem
  n <- nrow(X); p <- ncol(X)
  Xm1 <- X[,-1] # remove leading column of 1's marking the intercept
  betam1 <- beta[-1] # remove intercept term

  xbar <- colMeans(Xm1) # find predictor means
  Xtilde <- sweep(Xm1, 2, xbar) # center each predictor according to its mean

  I <- diag(p - 1)
  UD <- cbind(rep(0, p - 2), diag(p - 2)) # upper diagonal matrix
  J <- -1 * cbind(diag(p - 2), rep(0, p - 2)) # diag (p - 2)*(p - 1) matrix
  A <- J + UD

  M <- solve(crossprod(Xtilde) + lam1 * I + lam2 * crossprod(A))
  b <- M %*% crossprod(Xtilde, (Xtilde %*% betam1))

  vcv <- matrix(0, nrow = p - 1, ncol = p - 1)
  if (n > p) { # when n > p this matrix multiplication routine is quicker
    vcv <- sigma^2 * M %*% tcrossprod(crossprod(Xtilde), M)
  } else { # when p > n this matrix multiplication routine is quicker
   vcv <- sigma^2 * tcrossprod(M, Xtilde) %*% tcrossprod(Xtilde, M)
  }

  return (list(b = b, vcv = vcv))
}
```

We now simulate some data to test our estimates:

```r
set.seed(124)

# set parameters
nsims <- 1e4
n <- 1e2
p <- 5
```

```r
lam1 <- 1
lam2 <- 1
sigma_star <- 1
beta_star <- rnorm(p)

# generate (fixed) design matrix
X <- cbind(rep(1, n), matrix(rnorm(n * (p - 1)), nrow = n, ncol = p - 1))

# compute expected parameter values
par_true <- fused_ridge_coef_params(X, lam1, lam2, beta_star, sigma_star)
b_true <- as.vector(par_true$b)
vcv_true <- par_true$vcv

# simulate our fused ridge coefficients nsims times
# outputs a matrix with rows corresponding to coefficients
# and columns correspond to simulation number
pt <- proc.time()
b_hat <- replicate(nsims, {
  y <- X %*% beta_star + rnorm(n, 0, sigma_star) # generate response
  return (as.vector(fused_ridge_coef(X, y, lam1, lam2)$b))
})
proc.time() - pt
```

```
##    user  system elapsed
##   1.779   0.015   1.810
```

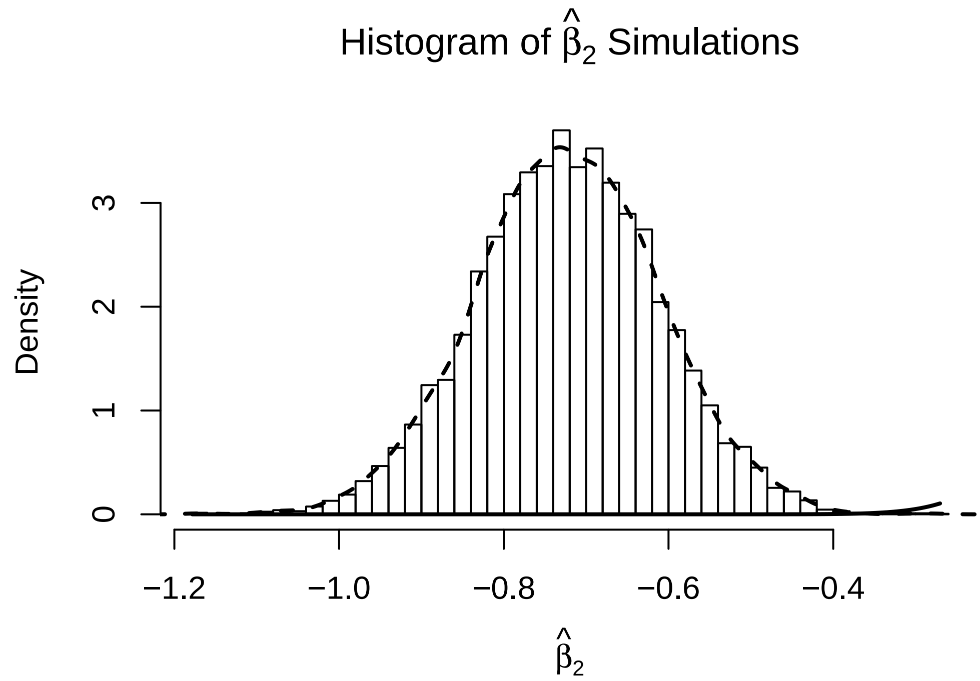
```r
# estimate variance of b2, ..., b_p estimates
vcv_hat <- var(t(b_hat))

# print estimated fused ridge coefficients vs. expected values
b <- rbind(rowMeans(b_hat), b_true)
rownames(b) <- c("b_hat", "b_true")
round(b, 4)
```

```
##          [,1]    [,2]   [,3]   [,4]
## b_hat  0.0316 -0.7226 0.2226 1.3899
## b_true 0.0313 -0.7240 0.2235 1.3920
```

```r
# print absolute error between estimated and true fused ridge variances
round(abs(vcv_true - vcv_hat), 4)
```

```
##       [,1]  [,2]  [,3]  [,4]
## [1,] 2e-04 1e-04 1e-04 1e-04
## [2,] 1e-04 1e-04 1e-04 2e-04
## [3,] 1e-04 1e-04 0e+00 1e-04
## [4,] 1e-04 2e-04 1e-04 3e-04
```

As a case study, we may look at the simulations of $\hat{\beta}_2^{(\lambda_1,\lambda_2)}$ and compare it with it's theoretical distribution. Note that the estimates $\hat{\beta}^{(\lambda_1,\lambda_2)} = M\tilde{X}^T\tilde{Y}$ are normally distributed because they are a linear combination of $\tilde{Y} \sim \mathcal{N}(\tilde{X}\beta, \sigma^2)$ (when our noise terms $\epsilon \sim \mathcal{N}(0, \sigma^2)$). We visualize the histogram of the $\hat{\beta}_2^{(\lambda_1,\lambda_2)}$ simulations with its empirical and theoretical densities overlaid (dashed, solid), along with its expected value (vertical line) below.


Histogram of $\hat{\beta}_2$ Simulations
Density
0
1
2
3
−1.2
−1.0
−0.8
−0.6
−0.4
$\hat{\beta}_2$

# Appendix

## Computing $\mathbb{E}\left[\hat{\beta}^{(\lambda)}\right]$

Consider the case of $n >> p$

```
library(microbenchmark)
set.seed(124)

#===== Large n case =====#
# parameters
n <- 1e2
p <- 1e1
lam <- 1

# generate data
beta <- rnorm(p)
X <- matrix(rnorm(n * p), nrow = n)
I <- diag(p)

# define functions
f1 <- function() solve(crossprod(X) + lam * I) %*% (crossprod(X) %*% beta)
f2 <- function() {
  X_svd <- svd(X)
  V <- X_svd$v
  d <- X_svd$d
  Dstar <- diag(d^2/(d^2 + lam))
  V %*% (Dstar %*% crossprod(V, beta))
}

# test speed
microbenchmark(f1(), f2(), times = 1e3, unit = "us")
```

```
## Unit: microseconds
##  expr     min       lq      mean   median       uq      max neval
##  f1()  40.444  44.0395  52.03565  49.1415  53.3540  383.244  1000
##  f2() 135.667 142.3900 171.28643 147.6030 164.7175 2083.201  1000
```

and the case for $p >> n$

```
#===== Large p case =====#
# parameters
n <- 1e1
p <- 1e2
lam <- 1

# generate data
beta <- rnorm(p)
X <- matrix(rnorm(n * p), nrow = n)
I <- diag(p)

# define functions
f1 <- function() solve(crossprod(X) + lam * I) %*% (crossprod(X) %*% beta)
f2 <- function() {
  X_svd <- svd(X)
```

```r
  V <- X_svd$v
  d <- X_svd$d
  Dstar <- diag(d^2/(d^2 + lam))
  V %*% (Dstar %*% crossprod(V, beta))
}

# test speed
microbenchmark(f1(), f2(), times = 1e3, unit = "us")
```

```
## Unit: microseconds
##  expr      min        lq      mean   median       uq       max neval
##  f1() 2507.265 2707.2475 3149.9153 2913.524 3246.726 41865.716  1000
##  f2()  144.897  166.7005  237.0413  198.463  246.933  2396.207  1000
```

and $n \approx p$

```r
#===== n ~ p case =====#
# parameters
n <- 1e2
p <- 1e2
lam <- 1

# generate data
beta <- rnorm(p)
X <- matrix(rnorm(n * p), nrow = n)
I <- diag(p)

# define functions
f1 <- function() solve(crossprod(X) + lam * I) %*% (crossprod(X) %*% beta)
f2 <- function() {
  X_svd <- svd(X)
  V <- X_svd$v
  d <- X_svd$d
  Dstar <- diag(d^2/(d^2 + lam))
  V %*% (Dstar %*% crossprod(V, beta))
}

# test speed
microbenchmark(f1(), f2(), times = 1e3, unit = "us")
```

```
## Unit: microseconds
##  expr      min      lq     mean   median       uq      max neval
##  f1() 3300.666 3480.52 3961.164 3688.882 4133.603 41792.41  1000
##  f2() 6324.855 6678.80 7330.393 7106.400 7689.252 45250.17  1000
```

## Matrix Multiplication Timing

Consider the following matrix multiplication benchmarks (for the cases of $n >> p$ and $p >> n$).

```r
set.seed(124)
#===== Large n case =====#

# set parameters
n <- 1e3
p <- 1e2
```

```r
lam <- 1

# generate data
X <- matrix(rnorm(n * p), nrow = n)
beta <- rnorm(p)
eps <- rnorm(n)
y <- X %*% beta + eps

ytilde <- y - mean(y)
xbar <- colMeans(X)
Xtilde <- sweep(X, 2, xbar)

# compute decomposition
Xtilde_svd <- svd(Xtilde)
U <- Xtilde_svd$u
d <- Xtilde_svd$d
V <- Xtilde_svd$v
Dstar <- diag(d/(d^2 + lam))

# define multiplication functions
f1 <- function() V %*% Dstar %*% t(U) %*% ytilde
f2 <- function() V %*% Dstar %*% (t(U) %*% ytilde)
f3 <- function() V %*% (Dstar %*% (t(U) %*% ytilde))
f4 <- function() V %*% (Dstar %*% crossprod(U, ytilde))
f5 <- function() V %*% crossprod(Dstar, crossprod(U, ytilde))

# test speed
microbenchmark(f1(), f2(), f3(), f4(), f5(), times = 100, unit = "us")
```

```
## Unit: microseconds
##   expr      min        lq       mean    median         uq       max neval
##   f1() 8902.576 9724.4920 10903.2958 10278.435 11331.7945 46141.138   100
##   f2() 1106.779 1258.9390  2081.1678  1446.653  2072.6245 42285.188   100
##   f3()  368.722  437.0350  1128.6115   522.717   695.6365 41738.803   100
##   f4()  129.717  139.9405   171.8732   154.455   180.8180   535.138   100
##   f5()  126.378  139.5715   169.8402   148.796   169.9505   649.357   100
```

```r
#===== Large p case =====#
set.seed(124)

# set parameters
n <- 1e2
p <- 1e3
lam <- 1

# generate data
X <- matrix(rnorm(n * p), nrow = n)
beta <- rnorm(p)
eps <- rnorm(n)
y <- X %*% beta + eps

# define multiplication functions
f1 <- function() V %*% Dstar %*% t(U) %*% ytilde
f2 <- function() V %*% Dstar %*% (t(U) %*% ytilde)
```

```r
f3 <- function() V %*% (Dstar %*% (t(U) %*% ytilde))
f4 <- function() V %*% (Dstar %*% crossprod(U, ytilde))
f5 <- function() V %*% crossprod(Dstar, crossprod(U, ytilde))

# test speed
microbenchmark(f1(), f2(), f3(), f4(), f5(), times = 100, unit = "us")
```

```
## Unit: microseconds
##  expr      min        lq       mean     median         uq       max neval
##  f1() 8696.477 9656.1220 11854.1073 10399.9315 11683.9650 74730.249   100
##  f2() 1103.554 1255.6490  1995.3685  1409.8800  1808.1070 37728.510   100
##  f3()  365.318  472.6250   858.6364   563.8255  1216.9440  2688.480   100
##  f4()  130.003  141.5435   182.3951   157.3140   178.4885  1094.263   100
##  f5()  126.270  131.6610   158.4705   147.7950   161.7560   492.192   100
```