Problem 1 Proximal operators

Let $f(x)$ be a convex function. The Moreau envelope $E_\gamma f(x)$ and proximal operator $prox_\gamma f(x)$ for parameter $\gamma > 0$ are defined as

$$E_\gamma f(x) = \min_z \{ f(z) + \frac{1}{2\gamma} \|z - x\|_2^2 \} \leq f(x)$$

$$\underset{\gamma}{prox}\, f(x) = \arg\min_z \{ f(z) + \frac{1}{2\gamma} \|z - x\|_2^2 \}$$

Intuitively, the Moreau envelope is a regularized version of *f*. It approximates f from below, and has the same set of minimizing values as *f*. The proximal mapping returns the value that solves the little minimization problem defined by the Moreau envelope. The objective in this little minimization problem balances two goals: minimizing *f*, and staying near *x*. The proximal operator says where the minimum occurs, while the Moreau envelope says what the value of the minimum is.

The following Figure 1 shows a simple one-dimensional example of a Moreau envelope and a proximal operator for the absolute-value function.

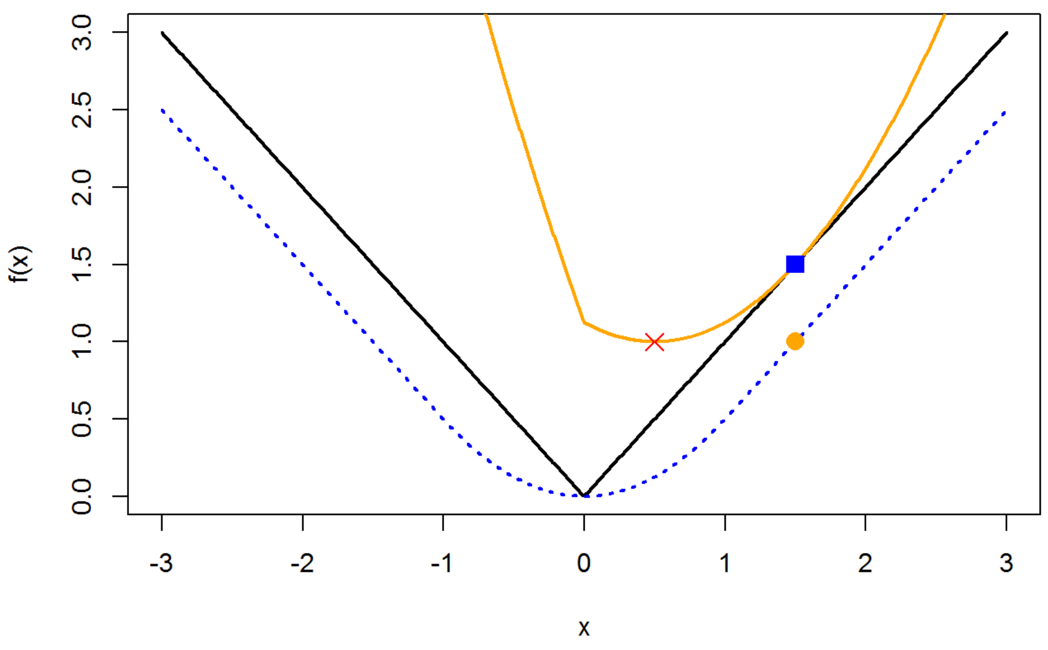


Figure 1: simple one-dimensional example of a Moreau envelope of $f(x) = |x|$

The solid black line shows the function $f(x) = |x|$, and the dotted blue line shows the corresponding Moreau envelope $E_\gamma f(x)$ with parameter $\gamma = 1$,

$$E_\gamma f(x) = Huber(x, c=1) = \begin{cases} 0.5x^2, & |x| \le 1 \\ |x| - 0.5, & otherwise \end{cases}$$

The orange line shows the function $|x| + 0.5(x - x_0)^2$ for *x₀* = 1.5, whose minimum (shown as the red cross) defines the Moreau envelope and proximal operator. This point has horizontal coordinate $prox_\gamma f(x^*) = 0.5$ and vertical coordinate $E_\gamma f(x) = 1$, and is closer than *x₀* to the overall minimum at *x* = 0. The orange circle shows the point $(x_0, E_\gamma f(x_0))$; the vertical coordinate of the orange point is precisely the vertical coordinate of the red point, emphasizing the point-wise construction of the Moreau envelope in terms of a simple optimization problem.

(A) The proximal operator gives a nice interpretation of classical gradient descent. Consider the local linear approximation of *f (x)* about a point *x₀*:

$$f(x) \approx \hat{f}(x; x_0) = f(x_0) + (x - x_0)^T \nabla f(x_0)$$

Derive the proximal operator (with parameter *γ*) of the linear approximation $\hat{f}(x; x_0)$, and show that this proximal operator is identical to a gradient-descent step for *f (x)* of size *γ*, starting from the point *x₀*.

Proof:

$$\underset{\gamma}{prox}\, \hat{f}(x; x_0) = \arg\min_z \{ \hat{f}(z; x_0) + \frac{1}{2\gamma} \|z - x\|_2^2 \}$$

$$= \arg\min_z \{ f(x_0) + (z - x_0)^T \nabla f(x_0) + \frac{1}{2\gamma}(z - x)^T (z - x) \}$$

$$= \arg\min_z \{ f(x_0) + z^T \nabla f(x_0) - x_0^T \nabla f(x_0) + \frac{1}{2\gamma}(z^T z - x^T z - z^T x + x^T x) \}$$

$$\Rightarrow \frac{d}{dz}[z^T \nabla f(x_0) + \frac{1}{2\gamma}(z^T z - x^T z - z^T x)] \overset{set}{=} 0 \ \Rightarrow \nabla f(x_0) + \frac{1}{\gamma}(\hat{z} - x) = 0 \ \Rightarrow \hat{z} = x - \gamma \nabla f(x_0)$$

For $x = x_0$, the proximal operator is $\underset{\gamma}{prox}\, \hat{f}(x; x_0) = x_0 - \gamma \nabla f(x_0)$.

That is, the gradient step minimizes a local linear approximation of the function, subject to a quadratic regularizer that keeps the next iterate close to *x₀*.

(B) Many intermediate steps in statistical optimization problems can be written very compactly in terms of proximal operators of log likelihoods or penalty functions. For example, consider a negative log likelihood of the form

$$l(x) = \frac{1}{2} x^T P x - q^T x + r$$

Show that the proximal operator with the parameter $1/\gamma$ of *l(x)* take the form

$$\underset{1/\gamma}{prox}\, l(x) = (P + \gamma I)^{-1}(\gamma x + q)$$

assuming the relevant inverse exists.

$$\underset{1/\gamma}{prox}\, l(x) = \arg\min_z \{l(z) + \frac{\gamma}{2} \| z - x \|_2^2\}$$
$$= \arg\min_z \{\frac{1}{2} z^T P z - q^T z + r + \frac{\gamma}{2} \| z - x \|_2^2\}$$
$$\frac{d}{dz}[\frac{1}{2} z^T P z - q^T z + r + \frac{\gamma}{2} \| z - x \|_2^2] = Pz - q + \gamma(z - x) \overset{set}{=} 0$$
$$\Rightarrow \underset{1/\gamma}{prox}\, l(x) = (P + \gamma I)^{-1}(q + \gamma x)$$

Next show that if we have a Gaussian sampling mode of the form $(y \mid x) \sim N(Ax, \Omega^{-1})$, then our negative log likelihood can be written in the form given above. Here *A* is like our feature matrix, *Ω*$^{-1}$ is a covariance matrix, and *x* is like a regression vector. Specify what *P*, *q*, and *r* are for this negative log likelihood.

Proof:

The multivariate Gaussian distribution is

$$p(x \mid \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} \mid \Sigma \mid^{1/2}} \exp\{-\frac{1}{2}(x - \mu)^T \Sigma^{-1}(x - \mu)\}$$

Then the likelihood function is

$$L(y \mid x) = (2\pi)^{-n/2} \mid \Omega \mid^{1/2} \exp\{-\frac{1}{2}(y - Ax)^T \Omega (y - Ax)\}$$

The negative log likelihood function is

$$l(y \mid x) = \frac{n}{2}\log(2\pi) - \frac{1}{2}\log(\mid \Omega \mid) + \frac{1}{2}(y - Ax)^T \Omega (y - Ax)$$
$$= \frac{n}{2}\log(2\pi) - \frac{1}{2}\log(\mid \Omega \mid) + \frac{1}{2}(y^T \Omega y - 2x^T A^T \Omega y + x^T A^T \Omega A x)$$
$$\Rightarrow l(y \mid x) = \frac{1}{2} y^T \Omega y - x^T A^T \Omega y + \frac{1}{2}[x^T A^T \Omega A x - \log(\mid \Omega \mid) + n \log(2\pi)]$$

Therefore,

$$P = \Omega$$
$$q = (x^T A^T \Omega)^{-1} = \Omega A x$$
$$r = \frac{1}{2}[x^T A^T \Omega A x - \log(\mid \Omega \mid) + n \log(2\pi)]$$

(C) Let $\phi(x) = \tau \| x \|_1$. Express the proximal operator of this function in terms of the soft-thresholding function.

$$\underset{\gamma}{prox}\, \phi(x) = \arg\min_z \{\phi(z) + \frac{1}{2\gamma} \| z - x \|_2^2\}$$
$$= \arg\min_z \{\tau \| z \|_1 + \frac{1}{2\gamma} \| z - x \|_2^2\}$$
$$= \arg\min_z \{\gamma\tau \| z \|_1 + \frac{1}{2} \| z - x \|_2^2\}$$

$$= \arg\min_z\{\gamma\tau\sum_i |z_i| + \frac{1}{2}\sum_i (z_i - x_i)^2\}$$

$$= \arg\min_z\{\gamma\tau |z_i| + \frac{1}{2}(z_i - x_i)^2\}$$

Now find the element-wise solution to find each z_i. In the previous set of exercise, we have shown that this is equal to soft thresholding function

$$S_{\gamma\tau}(x_i) = sign(x_i)(|z_i| - \gamma\tau)_+$$

Problem 2 The proximal gradient method

Suppose that we have some objective function that can be expressed as $f(x) = l(x) + \phi(x)$, where *l(x)* is differentiable but is $\phi(x)$ not. The proximal gradient method is designed for precisely this situation. Recall from above the idea of forming a local linear approximation to a function at some point x_0 and then adding a quadratic regularizer. This gave us an interpretation of gradient descent evaluating the proximal operator of our locally linear approximation.

Here, we'll apply this idea to the first term in our objective *l(x)*. Define

$$l(x) = \tilde{l}(x; x_0) = l(x_0) + (x - x_0)^T \nabla l(x_0) + \frac{1}{2\gamma} \| x - x_0 \|_2^2$$

as our linear approximation to *l(x)*, plus the quadratic regularizer. Now we add in the $\phi(x)$ term to get the approximation for our original objective:

$$f(x) \approx \tilde{f}(x; x_0) = \tilde{l}(x; x_0) + \phi(x) \qquad (1)$$

(A) Consider the surrogate optimization problem in which we minimize the approximation $\tilde{f}(x; x_0)$ in the above equation, in lieu of our original objective *f (x)*.

$$\hat{x} = \arg\min_x\{\tilde{l}(x; x_0) + \phi(x)\}$$

Now show the solution of this problem.

$$\hat{x} = \arg\min_x\{\tilde{l}(x; x_0) + \phi(x)\}$$

$$= \arg\min_x\{l(x_0) + (x - x_0)^T \nabla l(x_0) + \frac{1}{2\gamma} \| x - x_0 \|_2^2 + \phi(x)\}$$

$$= \arg\min_x\{x^T \nabla l(x_0) + \frac{1}{2\gamma} \| x - x_0 \|_2^2 + \phi(x)\}$$

$$= \arg\min_x\{\frac{1}{2\gamma}[(x - x_0)^T (x - x_0) + 2\gamma x^T \nabla l(x_0)] + \phi(x)\}$$

$$= \arg\min_x\{\frac{1}{2\gamma}[(x - x_0)^T (x - x_0) + 2(x - x_0)^T \gamma\nabla l(x_0) + (\gamma\nabla l(x_0))^T \gamma\nabla l(x_0)$$

$$- 2x_0^T \gamma\nabla l(x_0) - (\gamma\nabla l(x_0))^T \gamma\nabla l(x_0)] + \phi(x)\}$$

$$= \arg\min_x \{ \frac{1}{2\gamma}[(x - x_0 + \gamma\nabla l(x_0))^T (x - x_0 + \gamma\nabla l(x_0))] + \phi(x)\}$$

$$= \arg\min_x \{\phi(x) + \frac{1}{2\gamma} \| x - (x_0 - \gamma\nabla l(x_0)) \|_2^2\}$$

$$\hat{x} = \underset{\gamma}{prox}\,\phi(x_0 - \gamma\nabla l(x_0)) = \underset{\gamma}{prox}\,\phi(u), \qquad where \quad u = x_0 - \gamma\nabla l(x_0) \tag{2}$$

This is just the proximal operator of the non-smooth part of the objective $\phi(x)$, evaluated at an intermediate gradient-descent step for the smooth part *l (x)*.

(B) The proximal gradient method is an iterative algorithm in which we repeatedly form the approximation in Equation (1) (the Moreau envelope objective function) about the current point, and minimize this surrogate function using Equation (2). Written concisely,

$$x^{(t+1)} = \underset{\gamma^{(t)}}{prox}\,\phi(u^{(t)}), \qquad u^{(t)} = x^{(t)} - \gamma^{(t)}\nabla l(x^{(t)})$$

Now consider the lasso regression problem:

$$\hat{\beta} = \arg\min_\beta \{\| y - X\beta \|_2^2 + \lambda \| \beta \|_1\}$$

$$l(\beta) = \frac{1}{2n} \| y - X\beta \|_2^2 = \frac{1}{2n} (y - X\beta)^T (y - X\beta)$$

Define $\nabla l(\beta) = \frac{1}{n}(X^T X\beta - X^T y) = -\frac{1}{n} X^T (y - X\beta)$

$$\phi(\beta) = \lambda \| \beta \|_1$$

Using the results on proximal operators derived already, we have

$$\begin{aligned} u^{(t)} &= \beta^{(t)} - \gamma^{(t)}\nabla_\beta \frac{1}{2n} \| y - X\beta^{(t)} \|_2^2 \\ &= \beta^{(t)} - \gamma^{(t)}\nabla_\beta \frac{1}{2n}[(y - X\beta^{(t)})^T (y - X\beta^{(t)})] \\ &= \beta^{(t)} + \gamma^{(t)} X^T (y - X\beta^{(t)}) / n \end{aligned}$$

$$\Rightarrow \beta_j^{(t+1)} = \underset{\gamma^{(t)}}{prox}\,\phi(u^{(t)}) = \underset{\gamma^{(t)}}{prox}\,\lambda \| u^{(t)} \|_1 = sign(u_j^{(t)})(| u_j^{(t)} | - \lambda\gamma^{(t)})_+$$

We will not apply the L1 penalty to the intercept and have the following pseudo code of proximal gradient algorithm:

```
i)   Procedure $proxGD(X, Y, \lambda, \gamma = 0.01, \beta_0)$
ii)      $for\ t = 1 : it$ do
iii)         $gradient \leftarrow gradient(X, Y, \beta^{(t)})$
iv)          $u \leftarrow \beta^{(t)} - \gamma \cdot gradient$
v)           $\beta^{(t+1)} \leftarrow prox.l1(u, \gamma \cdot \lambda)$        (soft thresholding function – proximal operator)
```

The primary computational cost the proximal gradient algorithm is computing the two matrix products of $\nabla l(\beta)$. Finding the proximal operator is trivial, simply involving the sign and max commands within R. Implementation in R gives very similar results to the outputs of the glmnet package, as shown in Figure 2.

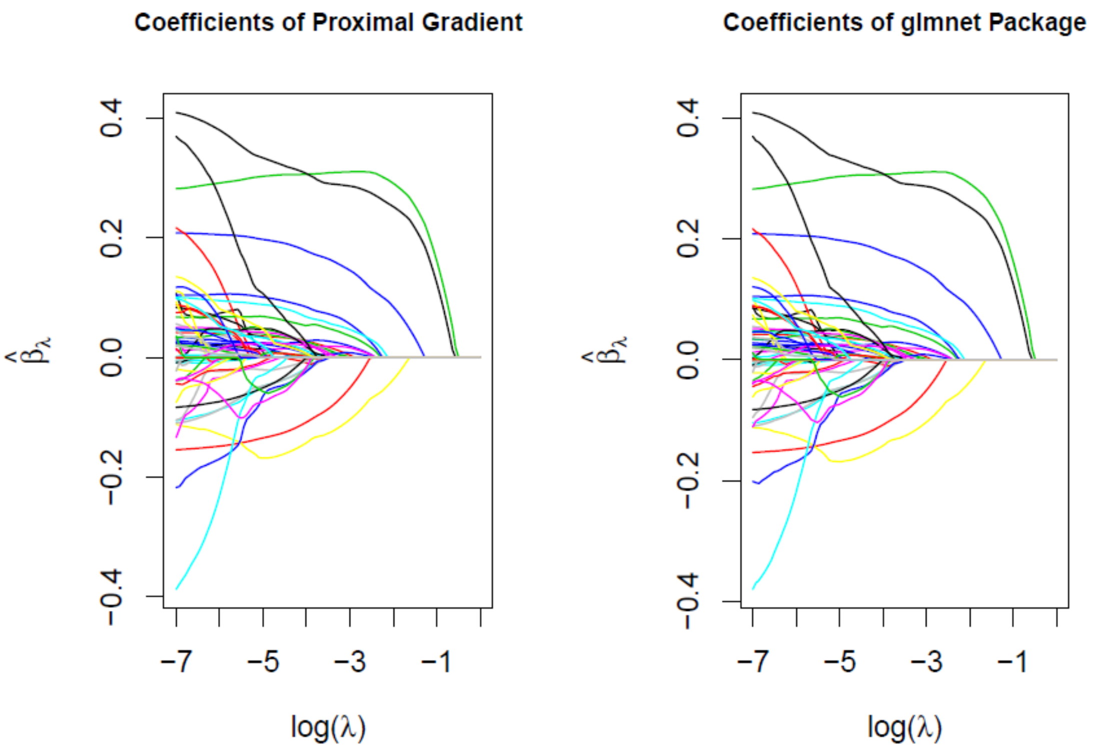


Figure 2: Comparison of outputs from proximal gradient method (left) and glmnet (right)

(C) A cool variation on proximal gradient is called the accelerated proximal gradient algorithm. The following scheme (due to Nesterov, who's super famous in this area) involves a simple extrapolation step based on the previous iteration:

$$x^{(t+1)} = \underset{\gamma^{(t)}}{prox}\,\phi(u^{(t)}), \qquad u^{(t)} = z^{(t)} - \gamma^{(t)}\nabla l(z^{(t)})$$

$$s^{(t+1)} = \frac{1+(1+4[s^{(t)}]^2)^{1/2}}{2}$$

$$z^{(t+1)} = x^{(t+1)} + \frac{s^{(t)}-1}{s^{(t+1)}}(x^{(t+1)} - x^{(t)})$$

The first step involves the proximal operator of the penalty function, evaluated not at the previous iterate $x^{(t)}$, but at an extrapolated version of $x^{(t)}$, based on the magnitude of the previous step's update. In this sense, large steps give "momentum" to the next step, where the amount of momentum is modulated by the scalar $s^{(t)}$ terms. Implement this acceleration scheme in accelerated proximal gradient code and compare its convergence speed to that of the unaccelerated version.

The pseudo code of this accelerated proximal gradient algorithm is

```
i)   Procedure $proxACCE(X, Y, \lambda, \gamma = 0.01, \beta_0)$
ii)  $for\ t = 1 : iter$ do
iii)     $gradient \leftarrow gradient(X, Y, z^{(t)})$
iv)      $s^{(t+1)} \leftarrow \frac{1+\sqrt{1+4(s^{(t)})^2}}{2}$
v)       $u \leftarrow z^{(t)} - \gamma \cdot gradient$
vi)      $\beta^{(t+1)} \leftarrow prox.l1(u, \gamma \cdot \lambda)$     (soft thresholding function – proximal operator)
vii)     $z^{(t+1)} \leftarrow \beta^{(t+1)} + \frac{s^{(t)}-1}{s^{(t+1)}}(\beta^{(t+1)} - \beta^{(t)})$
```

The performance of the two algorithms is illustrated in Figure 3. We used, as a convergence criterion, the relative change of the objective function less than 10^{-10}. The accelerated version converges (in 652 iterations) faster than the normal one (in 4807 iterations) with both $\lambda = 0.01$ and $\gamma = 0.01$. Comparing the final given values of the coefficients rounded two decimal, they are exactly the same. Figure 4 compares all coefficients for the two methods and clearly shows that the coefficients are in fairly close agreement when computed under the two algorithms.

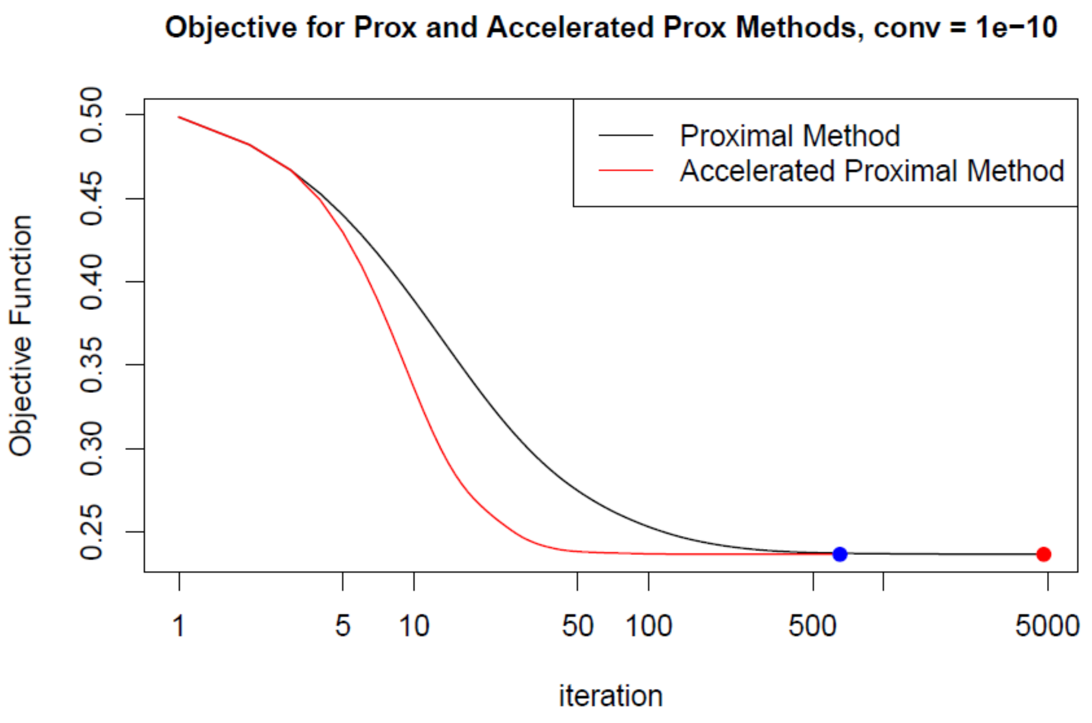


Figure 3: Comparison of convergence for unaccelerated vs. accelerated proximal gradient method

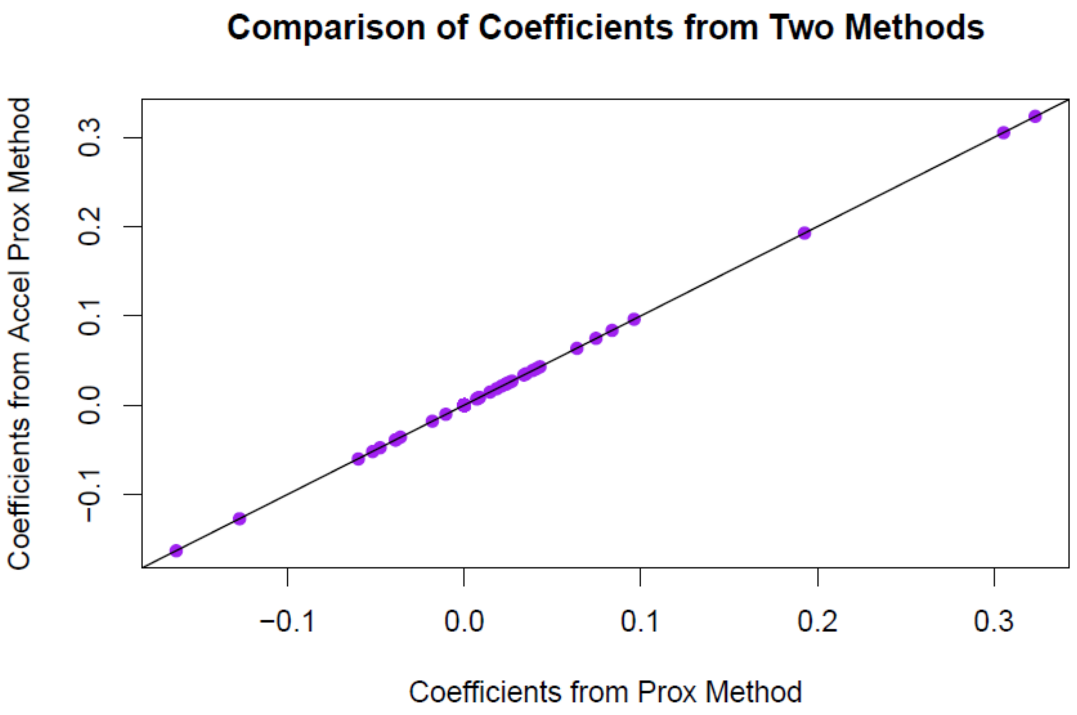


Figure 4: Comparison of coefficients for unaccelerated vs. accelerated proximal gradient method

exercise06.R

Shuchen

Sat Oct 21 19:04:54 2017

```r
# Big Data Exercise06 The Proximal Gradient Method
# Yanxin Li
# Oct 20 2017

rm(list=ls())
setwd("D:/2017 UT Austin/Statistical Models for Big Data/R")
```

```r
# Negative log likelihood function
nll <- function(X, Y, beta) {
  A <- Y - X %*% beta
  loglike <- (0.5/nrow(X)) * crossprod(A)
  return(loglike)
}
```

```r
# Gradient of negative log likelihood function
gradient <- function(X, Y, beta) {
  A <- Y - X %*% beta
  grad <- -(1/nrow(X)) * crossprod(X, A)
  return(grad)
}
```

```r
# Proximal operator
prox.l1 <- function(u, lambda) {
  uhat <- abs(u) - lambda
  ubind <- cbind(rep(0, length(u)), uhat)
  prox <- sign(u) * apply(ubind, 1, max) # 1: row, 2: column
  return(prox)
}
```

```r
# Proximal gradient descent
proxGD <- function(X,Y,lambda=0.01,gamma=0.01,beta0=NA,iter=50000,conv=1e-10){
  # Set beta0 equal to a series of zeros
  if (is.na(beta0)) { beta0 <- rep(0, ncol(X)) }

  # Initialize coefficients matrix
  beta <- matrix(rep(NA, ncol(X)*(iter+1)), nrow=ncol(X))
  beta[, 1] <- beta0

  # Initialize objective funtion
  obj <- rep(NA, iter+1)
  obj[1] <- nll(X, Y, beta0) + lambda * sum(abs(beta0))

  # Initialize convergence message in case convergence not reached
  message <- "Convergence not reached..."

  for (t in 1:iter) {
    # Update u, beta and obj
    u <- beta[,t] - gamma * gradient(X, Y, beta[,t])
    beta[,t+1] <- prox.l1(u, gamma * lambda)
    obj[t+1] <- nll(X, Y, beta[,t+1]) + lambda * sum(abs(beta[,t+1]))

    # Check convergence
    delta <- abs(obj[t+1]-obj[t]) /(abs(obj[t])+conv)
    if (delta < conv) {
      # Remove excess betas and obj
      beta <- beta[, -((t+2):ncol(beta))]
      obj <- obj[-((t+2):length(obj))]

      # Update convergence message
      message <- sprintf("Convergence reached after %i iterations", (t+1))
      break
    }
  }

  result <- list("beta.hat"=beta[,ncol(beta)],"beta"=beta,"objective"=obj,"conv"=message)
  return(result)
}
```

```
# Accelerated proximal gradient method
proxACCE <- function(X,Y,lambda=0.01,gamma=0.01,beta0=NA,iter=50000,conv=1e-10){
  # Set beta0 equal to a series of zeros
  if (is.na(beta0)) { beta0 <- rep(0, ncol(X)) }

  # Create s vector
  s <- rep(NA, iter+1)
  s[1] <- 1

  for (j in 2:length(s)) {
    s[j] <- (1+sqrt(1+4*s[j-1]^2))/2
  }

  # Initialize z matrix
  z <- matrix(0, nrow=ncol(X), ncol=iter+1)

  # Initialize coefficients matrix
  beta <- matrix(rep(NA, ncol(X)*(iter+1)), nrow=ncol(X))
  beta[, 1] <- beta0

  # Initialize objective funtion
  obj <- rep(NA, iter+1)
  obj[1] <- nll(X, Y, beta0) + lambda * sum(abs(beta0))

  # Initialize convergence message in case convergence not reached
  message <- "Convergence not reached..."

  for (t in 1:iter) {
    # Update u, beta, z and obj
    u <- z[,t] - gamma * gradient(X, Y, z[,t])
    beta[,t+1] <- prox.l1(u, gamma * lambda)
    z[,t+1] <- beta[,t+1] + (s[t] - 1)/s[t+1] * (beta[,t+1] - beta[,t])
    obj[t+1] <- nll(X, Y, beta[,t+1]) + lambda * sum(abs(beta[,t+1]))
    # Check convergence
    delta <- abs(obj[t+1]-obj[t]) /(abs(obj[t])+conv)
    if (delta < conv) {
      # Remove excess betas and nll
      beta <- beta[, -((t+2):ncol(beta))]
      obj <- obj[-((t+2):length(obj))]

      # Update convergence message
      message <- sprintf("Convergence reached after %i iterations", (t+1))
      break
    }
  }

  result <- list("beta.hat"=beta[,ncol(beta)],"beta"=beta,"objective"=obj,"conv"=message)
  return(result)
}
```

```
library(MASS)
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 3.4.2
```

```
## Loading required package: Matrix
```

```
## Loading required package: foreach
```

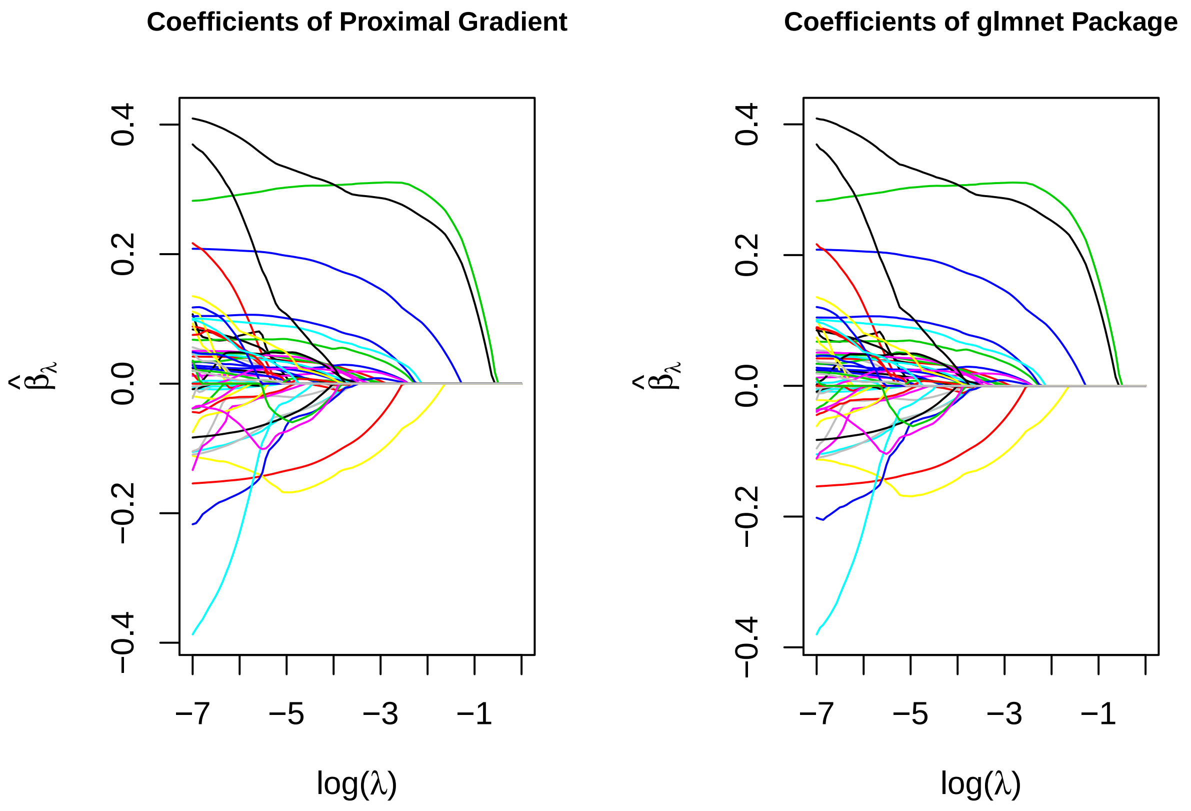
```
## Warning: package 'foreach' was built under R version 3.4.2
```

```
## Loaded glmnet 2.0-13
```

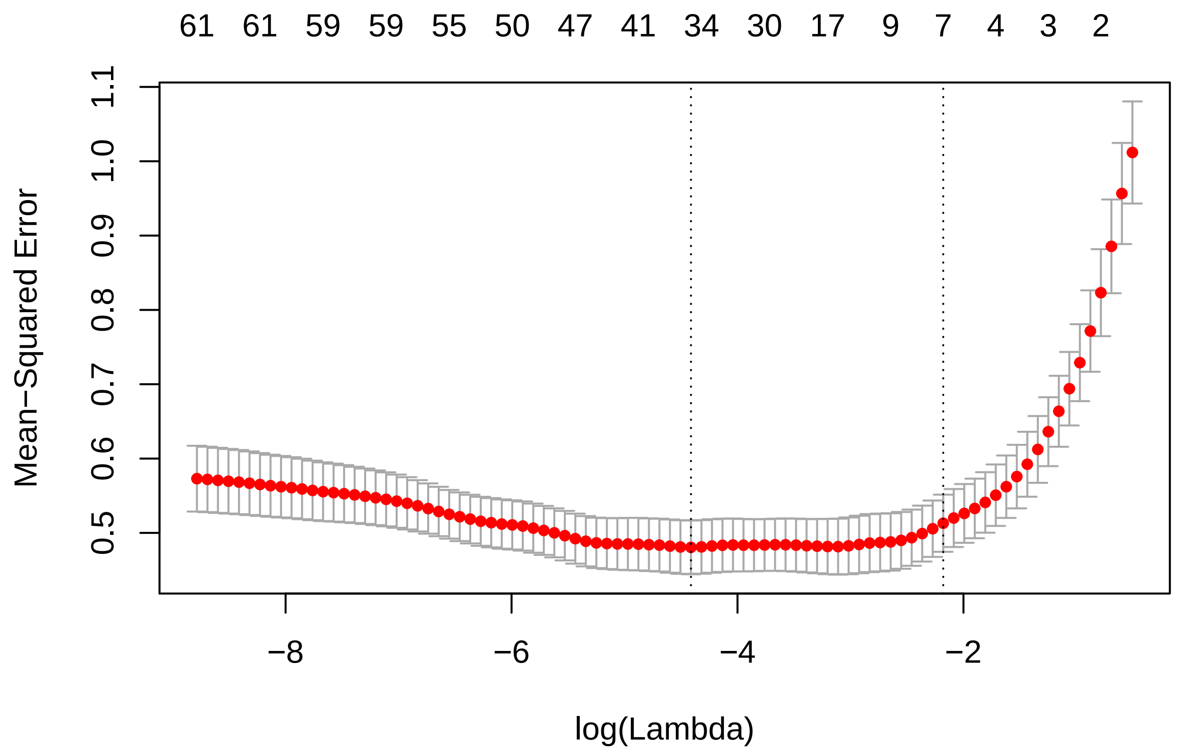
```
library(ggplot2)

# Read in data
X <- as.matrix(read.csv("diabetesX.csv", header = TRUE))
Y <- as.numeric(unlist(read.csv("diabetesY.csv", header = FALSE)))

# Scale data
X <- scale(X)
Y <- scale(Y)

# Setup the range of lambda
lambda <- exp(seq(0,-7,length.out=100)) # [0.00091, 1]
```

```
# Compare proximal gradient algorithm with glmnet
# Calculate coefficients using proximal gradient algorithm
betamat <- matrix(rep(NA, length(lambda)*ncol(X)), ncol=length(lambda))
for (i in 1:length(lambda)) {
  mylasso <- proxGD(X, Y, lambda=lambda[i])
  betamat[, i]  <- mylasso$beta.hat
}

# Fit Lasso model across a range of lambda values
myLasso <- glmnet(X,Y,alpha=1,lambda=lambda)

# Plot all beta's vs. lambda both from accelerated proximal gradient method and glmnet
par(mar=c(4,5,4,2))
par(mfrow=c(1,2))
```

```
plot(0,0,type='n', ylim=c(min(betamat), max(betamat)), cex.main=0.8,
     ylab=expression(hat(beta)[lambda]), xlim=log(range(lambda)),
     xlab=expression(paste('log(',lambda,')')), main='Coefficients of Proximal Gradient')

for (i in 1:nrow(betamat)) { lines(log(lambda), betamat[i, ], col=i) }

# plot(myLasso) # R default, the L1 norm is the regularization term for Lasso
plot(0,0,type='n', ylim=range(myLasso$beta), cex.main=0.8,
     ylab=expression(hat(beta)[lambda]), xlim=log(range(myLasso$lambda)),
     xlab=expression(paste('log(',lambda,')')), main='Coefficients of glmnet Package')

for(i in 1:nrow(myLasso$beta)){ lines(log(lambda),myLasso$beta[i,],col=i) }
```



```
# 10-fold Cross-validation to choose the best tuning parameter lambda for glmnet
set.seed(1)
ratio <- 0.5 # Split data
index <- sample(1:nrow(X), floor(ratio*nrow(X))) # Make equal partition

# 10-fold cross-validation
set.seed(1)
cv.out <- cv.glmnet(X[index, ],Y[index],alpha =1)
par(mfrow=c(1,1))
plot(cv.out)
```



```r
bestlam <- cv.out$lambda.min
# The value of lambda that results in the smallest CV error is 0.01212987
print(bestlam)
```

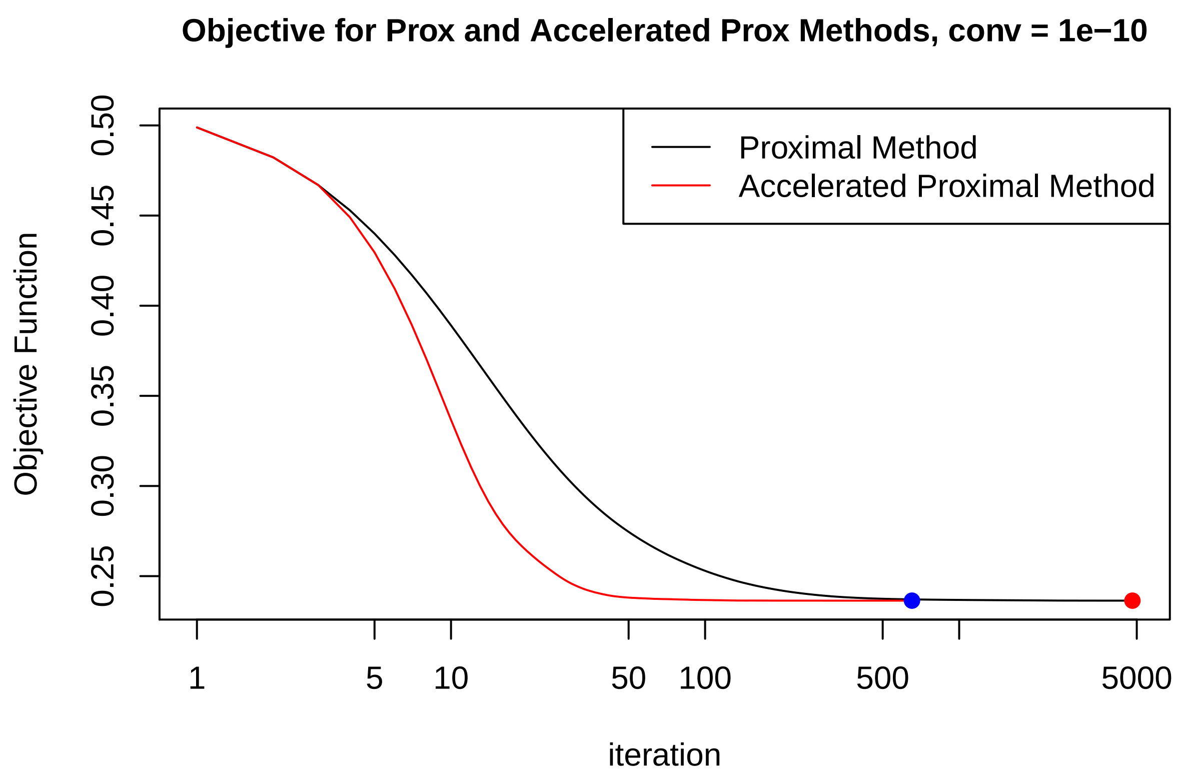
```
## [1] 0.01212987
```

```r
# Compare proximal gradient algorithm with accelerated proximal gradient algorithm
lassoGD <- proxGD(X, Y, lambda=0.01, gamma=0.01, conv=1e-10)
lassoACCE <- proxACCE(X, Y, lambda=0.01, gamma=0.01, conv=1e-10)

# Number of iterations when convergence met
n1 <- length(lassoGD$objective)
n2 <- length(lassoACCE$objective)
print(n1); print(n2)
```

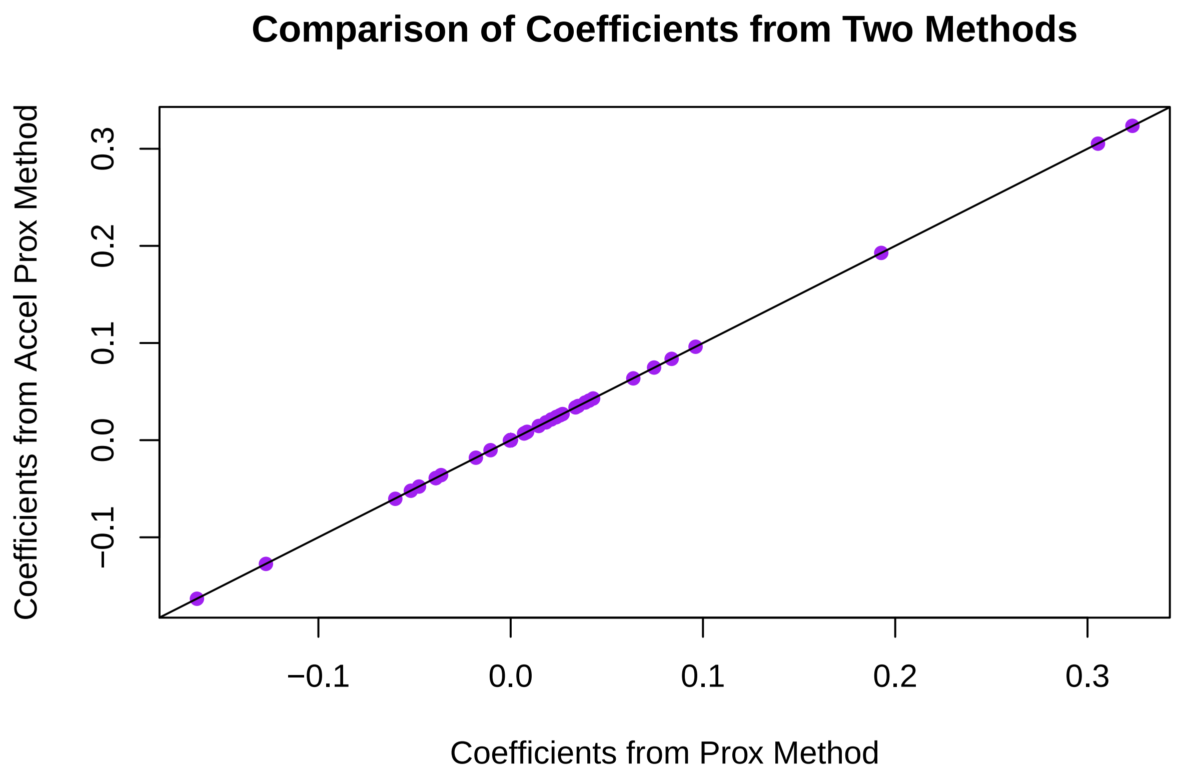
```
## [1] 4807
```

```
## [1] 652
```

```r
# Plot objective values and compare number of iterations used when convergence met
par(mfrow=c(1,1))
plot(lassoGD$objective, type="l", col="black", log="x",
     xlab="iteration", ylab="Objective Function", cex.main=1,
     main=sprintf("Objective for Prox and Accelerated Prox Methods, conv = 1e%i", -10))
lines(lassoACCE$objective, col="red")
points(n1, lassoGD$objective[n1], pch=19, col="red")
points(n2, lassoACCE$objective[n2], pch=19, col="blue")
legend("topright", legend=c("Proximal Method", "Accelerated Proximal Method"),
       col=c("black", "red"), lty=c(1,1), cex=1)
```



```
# Compare beta.hat's of the two algorithms
plot(lassoGD$beta.hat, lassoACCE$beta.hat,
     main="Comparison of Coefficients from Two Methods",
     xlab="Coefficients from Prox Method",
     ylab="Coefficients from Accel Prox Method",pch=16,col="purple")
abline(0, 1)
```



```
# Eyeball beta.hat
cbind(lassoGD$beta.hat,lassoACCE$beta.hat)
```

```
##               [,1]          [,2]
##  [1,]  0.0085564429  0.0085915940
##  [2,] -0.1272850118 -0.1273894244
##  [3,]  0.3054399755  0.3052213663
##  [4,]  0.1927202076  0.1927523430
##  [5,] -0.0389473856 -0.0390691752
##  [6,]  0.0000000000  0.0000000000
##  [7,] -0.1631759396 -0.1632479365
##  [8,]  0.0000000000  0.0000000000
##  [9,]  0.3233600948  0.3235603113
## [10,]  0.0337271163  0.0337561048
## [11,]  0.0350896498  0.0351730733
## [12,]  0.0238524080  0.0239467991
## [13,]  0.0000000000  0.0000000000
## [14,]  0.0000000000  0.0000000000
## [15,]  0.0000000000  0.0000000000
## [16,]  0.0000000000  0.0000000000
## [17,]  0.0000000000  0.0000000000
## [18,]  0.0000000000  0.0000000000
## [19,]  0.0637721035  0.0635910679
## [20,]  0.0961571643  0.0961659263
## [21,]  0.0000000000  0.0000000000
## [22,]  0.0145922703  0.0145892034
## [23,]  0.0000000000  0.0000000000
## [24,] -0.0389685266 -0.0390514872
## [25,]  0.0256917089  0.0256249736
## [26,]  0.0000000000  0.0000000000
## [27,]  0.0407828633  0.0407070250
## [28,]  0.0183655356  0.0183254028
## [29,]  0.0269937207  0.0268586053
## [30,]  0.0387531390  0.0387718137
## [31,]  0.0000000000  0.0000000000
## [32,] -0.0181018978 -0.0180600307
## [33,]  0.0428830143  0.0429078150
## [34,]  0.0000000000  0.0000000000
## [35,]  0.0000000000  0.0000000000
## [36,]  0.0000000000  0.0000000000
## [37,]  0.0837098733  0.0836933459
## [38,] -0.0004062054 -0.0003973438
## [39,]  0.0000000000  0.0000000000
## [40,]  0.0000000000  0.0000000000
## [41,]  0.0000000000  0.0000000000
## [42,]  0.0000000000  0.0000000000
## [43,]  0.0000000000  0.0000000000
## [44,]  0.0083483261  0.0082331604
## [45,]  0.0000000000  0.0000000000
## [46,]  0.0235917687  0.0235682975
## [47,]  0.0000000000  0.0000000000
## [48,]  0.0000000000  0.0000000000
## [49,] -0.0361962960 -0.0359988507
## [50,]  0.0000000000  0.0000000000
## [51,]  0.0074551531  0.0073961617
## [52,] -0.0476875712 -0.0477806271
## [53,] -0.0104831124 -0.0103190057
## [54,]  0.0000000000  0.0000000000
## [55,]  0.0000000000  0.0000000000
## [56,]  0.0000000000  0.0000000000
## [57,]  0.0745775553  0.0747505605
## [58,]  0.0069703990  0.0068581242
## [59,] -0.0519113331 -0.0520842898
## [60,]  0.0000000000  0.0000000000
## [61,]  0.0000000000  0.0000000000
## [62,] -0.0600073437 -0.0603700670
## [63,]  0.0211350485  0.0213435212
## [64,]  0.0000000000  0.0000000000
```

```r
# Number of nonzero coefficients by using proximal gradient method
sum(lassoGD$beta.hat != 0)
```

```
## [1] 34
```

```r
# Number of nonzero coefficients by using accelerated proximal gradient
sum(lassoACCE$beta.hat != 0)
```

```
## [1] 34
```