SDS 385 Ex 06:
The Proximal Gradient Method

October 8, 2016

Jennifer Starling

Proximal Operators

Part A

$\hat{x} = \underset{x}{\operatorname{argmin}} \left\{ \tilde{l}(x; x_0) + \phi(x) \right\}$

Plug into the Moreau envelope:

$$E_\gamma(\tilde{l}(x; x_0)) = \underset{x}{\operatorname{argmin}} \left\{ l(x_0) + (z - x_0)^T \nabla l(x_0) + \tfrac{1}{2\gamma}(z - x_0)^T (z - x_0) + \phi(z) \right\} \leq \tilde{l}(x; x_0)$$

$$= \underset{x}{\operatorname{argmin}} \left\{ l(x_0) + (z - x_0)^T \nabla l(x_0) + \tfrac{1}{2\gamma}(z^T z - 2z^T x_0 + x_0^T x_0) + \phi(z) \right\}$$

Take derivative of the Moreau envelope and set equal to zero:

$$\nabla_z E_\gamma(\tilde{l}(x; x_0)) = \nabla l(x_0) + \tfrac{1}{2\gamma}(2z - 2x_0) + \phi(z) = 0$$

Solve for $\phi(z)$:

$$\phi(z) + \tfrac{z - x_0}{\gamma} = -\nabla l(x_0)$$

$$\phi(z) = x_0 - z - \gamma \nabla l(x_0)$$

To minimize, need $z = x_0 - \gamma \nabla l(x_0)$.

Therefore, the proximal operator solution has this form:

$\hat{x} = prox_\gamma(\phi(z))$, where $z = x_0 - \gamma \nabla l(x_0)$.

Part B

1) Consider the Gaussian sampling model $(y|x) \sim N(Ax, \Omega^{-1})$.

The multivariate loglikelihood has form:

$$logl(y) = \frac{1}{2}log(|\Sigma|) + \frac{1}{2}(y-\mu)^T\Sigma^{-1}(y-\mu) + \frac{k}{2}log(2\pi) \text{ for } N(\mu, \Sigma).$$

Plug into this form, and drop all terms which do not depend on x, to get:

$$l(x|y) = \frac{1}{2}(y-Ax)^T(y-Ax)$$

Then expand:

$$= \frac{1}{2}(y^T - x^TA^T)\Omega(y-Ax)$$
$$= \frac{1}{2}(y^T\Omega y - 2y^T\Omega Ax + x^TA^T\Omega Ax)$$
$$= \frac{x^TA^T\Omega Ax}{2} - y^T\Omega Ax + \frac{y^T\Omega y}{2}$$

Let:

- $P = A^T\Omega A$
- $q = -(y^T\Omega Ax)^T = -A^T\Omega y$
- $r = \frac{y^T\Omega y}{2}$

Then can write $l(x|y) = \frac{1}{2}x^TPx + q^Tx + r$ ■

2) The proximal operator is derived as follows.

$$E_{\phi=1/\gamma}[l(x)] = \underset{z}{\operatorname{argmin}} \left\{ l(z) + \frac{1}{1\phi}||z-x||_2^2 \right\} \leq l(x)$$
$$= \underset{z}{\operatorname{argmin}} \left\{ z^TPz + q^Tz + r + \frac{1}{1\phi}(z-x)^T(z-x) \right\} \leq l(x)$$
$$= \underset{z}{\operatorname{argmin}} \left\{ z^TPz + q^Tz + r + \frac{1}{1\phi}(z^Tz - z^Tx + x^Tx) \right\} \leq l(x)$$

Then take derivative of the Moreau operator:

$$\nabla_x E_{\phi=1/\gamma}[l(x)] = \frac{1}{2}(2Pz) + q + \frac{1}{2\phi}(2x - 2z) = 0$$

$$Pz + q + \gamma(z - x) = 0, \text{ since } \phi = 1/\gamma$$

$$z(P + \gamma I) = \gamma x - q$$

$$z = (P - \gamma I)^{-1}(\gamma x - q)$$

Therefore, the proximal operator is $prox_\phi = (P - \gamma I)^{-1}(\gamma x - q)$ ■

Part C

Let $\phi(x) = \tau||x||_1$. Express the proximal operator in terms of the soft-thresholding function from Exercise 5, $S_\lambda(y) = sign(y)(|y| - \lambda)_+$

1) First, the Moreau envelope is $E_\lambda(\phi(z)) = \underset{x}{\operatorname{argmin}} \left\{ \phi(z) + \frac{1}{2\gamma}||z-x||_2^2 \right\} \leq \phi(x)$

2) The Moreau envelope for a single element:

$\tau|z| + \frac{1}{2\gamma}(z-x)^2$

<u>Case 1, if $z > 0$:</u>

$$\frac{\delta}{\delta z}\left(\tau|z| + \frac{1}{2\gamma}(z-x)^2\right) = \tau + \frac{1}{\gamma}(z-x) \to z = (\frac{x}{\gamma} - \tau)\gamma \to z = (x - \tau\gamma)$$

<u>Case 2, if $z < 0$:</u>

$$\frac{\delta}{\delta z}\left(\tau|z| + \frac{1}{2\gamma}(z-x)^2\right) = -\tau + \frac{1}{\gamma}(z-x) \to z = (\frac{x}{\gamma} + \tau)\gamma \to z = (x + \tau\gamma)$$

<u>Case 3, if $z = 0$:</u>

The Moreau reduces to $\frac{1}{2\gamma}(-x)^2$, and the derivative with respect to z is zero.

These cases can be summarized by the soft thresholding function:

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

Therefore, the proximal operator $prox_{\tau\gamma}(x) = S_{\tau\gamma}(x) = sign(x)(|x| - \tau\gamma)_+$. ■

The Proximal Gradient Method

Part A

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

1) Plug the approximation $\hat{f}(x; x_0)$ into the Moreau envelope:

$E_\gamma(\hat{f}(x; x_0)) = \underset{x}{\text{argmin}} \left\{ \hat{f}(x; x_0) + \frac{1}{2\gamma} ||x - x_0||_2^2 \right\} \leq f(x)$

$= \underset{x}{\text{argmin}} \left\{ f(x_0) + (x - x_0)^T \nabla f(x_0) + \frac{1}{2\gamma} (x - x_0)^T (x - x_0) \right\} \leq f(x)$

$= \underset{x}{\text{argmin}} \left\{ f(x_0) + (x - x_0)^T \nabla f(x_0) + \frac{1}{2\gamma} (x^T x - 2x^T x_0 + x_0^T x_0) \right\} \leq f(x)$

2) Take gradient of the Moreau operator, set equal to zero, and solve for the x which minimizes the derivative.

$\nabla_x (E_\gamma(\hat{f}(x; x_0))) = \nabla f(x_0) + \frac{1}{2\gamma}(2x - 2x_0) = 0 \rightarrow$

$\nabla f(x_0) + \frac{1}{\gamma}(x - x_0) = 0 \rightarrow$

$x = x_0 - \gamma \nabla f(x_0)$

The proximal operator is the x which minimizes the Moreau envelope, therefore:

$prox_\gamma = x_0 - \gamma \nabla f(x_0)$

This has the form of a gradient descent update with step size γ. ■

Part B

Pseudo-code for the proximal gradient algorithm is as follows.

<u>Initial Steps:</u>

- Set initial beta value (such as a vector of zeros).
- Set initial objective function value, using f(X,y,lambda,beta0).
- Set up matrix to hold gradient values at each iteration.

Begin iterations at i=2.

<u>Gradient Step:</u>

- Update $gradient^{(i-1)}$ using gradient for $f(x)$ part of objective (see below).

<u>Proximal Step:</u>

- Calculate intermediate vector $z = beta^{(i-1)} - \gamma * gradient^{(i-1)}$
- Update betas using $beta^{(i)} = prox(z, gamma, tau = lambda)$

<u>Convergence Housekeeping Step:</u>

- Upadte objective function using $beta^{(i)}$.
- Check for convergence using abs change in objective function.

A few notes for implementing the proximal gradient algorithm for LASSO.

- LASSO: $\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \left\{ ||y - X\beta||_2^2 + \lambda ||\beta||_1 \right\} = \underset{\beta}{\operatorname{argmin}} \{f + g\}$
- Objective function is rewritten as: $(y - X\beta)^T(y - X\beta) + \lambda \sum |\beta|$
- Gradient of the non-penalty portion of the objective function is $\nabla_\beta(f) = \nabla_\beta \left((y - XB)^T(y - XB) \right) = -2X^Ty + 2X^TX\beta$

Part B - Implementation in R

Implementation of the accelerated proximal gradient descent converged more quickly than the regular proximal gradient descent. See figure below for the objective functions.

```
> print(output$iter)
[1] 7687
> print(outputAccel$iter)
[1] 1853
> print(output$converged)
[1] 1
> print(outputAccel$converged)
[1] 1
```

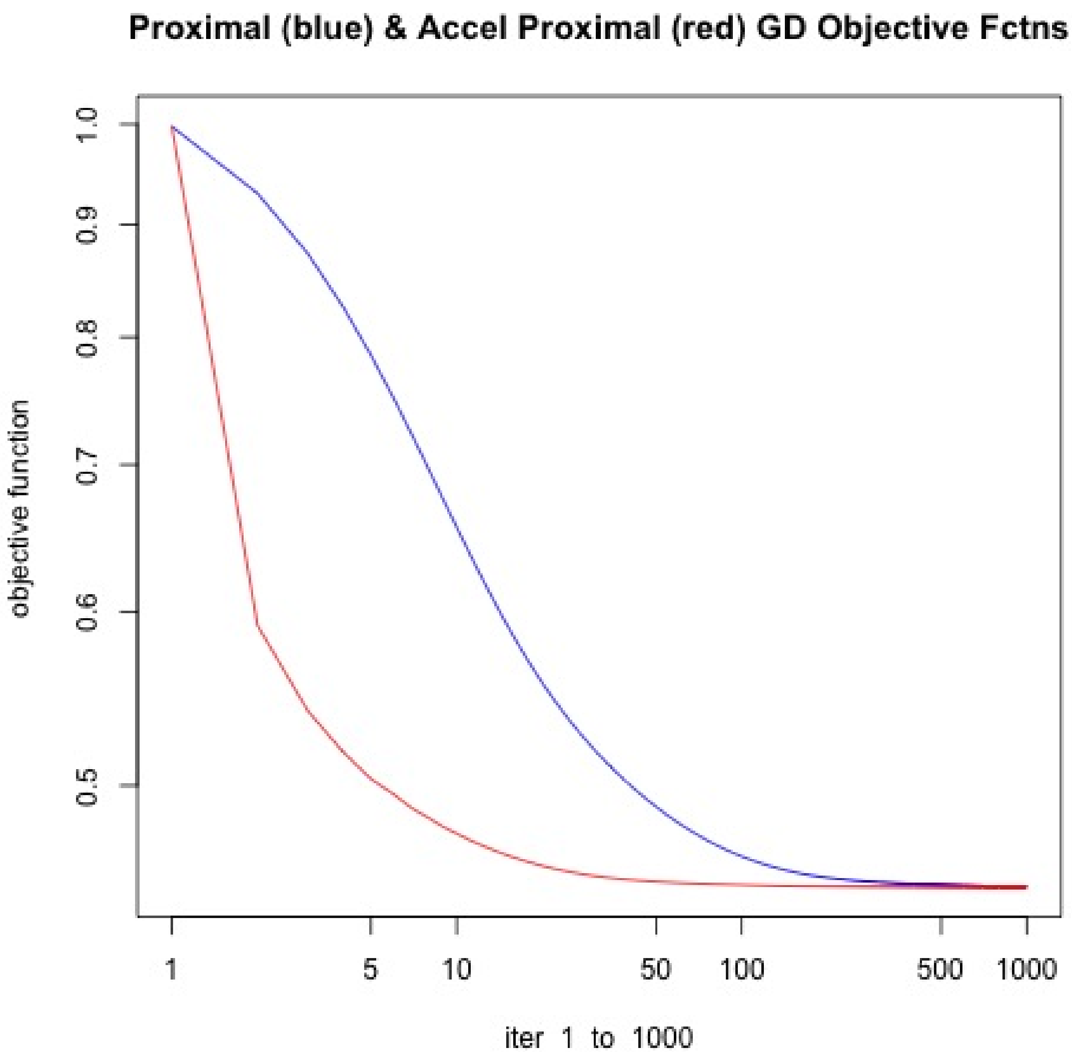

Figure 1: Prox & Accel Prox Objectives

For Accelerated Proximal, it did seem to descend consistently, but this was a surprise to me. In reading, I saw documentation of 'Nesterov Ripples', which noted that Accelerated Proximal Gradient Descent is not truly a descent algorithm.

Here is a comparison of the beta coefficient estimates I obtained with both methods, versus those calculated by glmnet. The coefficients look to be of the same magnitude and direction in both cases.

```
64 x 3 sparse Matrix of class "dgCMatrix"
                    s0      proximal  accel.prox
age       0.0084994686  0.0171642800  0.01572673
sex      -0.1274718132 -0.1394743658 -0.13943695
bmi       0.3053578438  0.3051274718  0.30015079
map       0.1928804642  0.2009670773  0.20139367
tc       -0.0387865853 -0.0377155655 -0.06178009
ldl       .            -0.0376434751  .
hdl      -0.1635236545 -0.1426736417 -0.15577212
tch       .             0.0537160979  0.01436791
ltg       0.3230766076  0.3198859003  0.34286594
glu       0.0337240035  0.0373169619  0.03762443
age.2     0.0351045119  0.0427545290  0.04263404
bmi.2     0.0238787313  0.0231121365  0.02483871
map.2     .            -0.0015880181 -0.00208119
tc.2      .             .             .
ldl.2     .             .             .
hdl.2     .             .             .
tch.2     .             0.0182890006  0.03565516
ltg.2     .             .             0.01278624
glu.2     0.0636440798  0.0735089090  0.06831205
age.sex   0.0961729059  0.1030491523  0.10416798
age.bmi   .             .             .
age.map   0.0145699540  0.0118708666  0.01113105
age.tc    .             .             .
age.ldl  -0.0389574124 -0.0517479493 -0.05308852
age.hdl   0.0258647293  0.0390369802  0.03935157
age.tch   .             0.0005535911  0.00218560
age.ltg   0.0407916229  0.0539588720  0.05090272
age.glu   0.0184222925  0.0237785514  0.02458299
sex.bmi   0.0270461479  0.0366747723  0.03564408
sex.map   0.0387600107  0.0438562290  0.04318104
sex.tc    .             .             .
sex.ldl  -0.0181954985 -0.0202717755 -0.01920683
sex.hdl   0.0429324533  0.0509889240  0.04841902
sex.tch   .            -0.0103067180 -0.01352070
sex.ltg   .             .             .
sex.glu   .             0.0095186855  0.01010271
bmi.map   0.0837040826  0.0921837729  0.09113240
bmi.tc   -0.0002777938 -0.0127770877 -0.01684757
bmi.ldl   .             .             .
bmi.hdl   .             .             .
bmi.tch   .             .             .
```

```
bmi.ltg  .             .             .
bmi.glu  .             0.0015817727  0.00314029
map.tc   0.0083855329  0.0139689484  0.02330329
map.ldl  .             0.0088425035  .
map.hdl  0.0233788843  0.0269141017  0.02262387
map.tch  .             .             .
map.ltg  .             .             .
map.glu -0.0362793960 -0.0593616103 -0.05895615
tc.ldl   .             0.0097864282  0.01510727
tc.hdl   0.0072487602  0.0046025918  .
tc.tch  -0.0477242186 -0.0772068542 -0.09664302
tc.ltg  -0.0098056312 -0.0369761727 -0.05132028
tc.glu   .             .             .
ldl.hdl  .             .             .
ldl.tch  .             .             .
ldl.ltg  0.0742412609  0.1157947622  0.13540195
ldl.glu  0.0067025050  0.0041732415  .
hdl.tch -0.0522309970 -0.0435042241 -0.04250106
hdl.ltg  .             .             .
hdl.glu  .             0.0235383572  0.03276363
tch.ltg -0.0607125224 -0.0786600477 -0.08710951
tch.glu  0.0214432611  0.0436959702  0.05905093
ltg.glu  .             0.0023713767  0.00480166
```

Appendix: R Code

```
### SDS 385 — Exercises 06 — Proximal Gradient Descent for LASSO.

#Jennifer Starling
#7 October 2016

rm(list=ls())   #Clean workspace.

library(glmnet)
library(Matrix)

#Read in Diabetes.csv data.
X <- read.csv(file='/Users/jennstarling/UTAustin/2016_Fall_SDS 385_Big_Data/
    Exercise 05 R Code/DiabetesX.csv',header=T)
y <- read.csv(file='/Users/jennstarling/UTAustin/2016_Fall_SDS 385_Big_Data/
    Exercise 05 R Code/DiabetesY.csv',header=F)

#Scale X and y.
X = scale(X)
y = scale(y)

#————————————————————————————————————————————————
#LASSO objective function:
#Inputs:
#   X = X matrix (scaled)
#   y = response data (scaled)
#   lambda = a chosen lambda value
#   beta = a vector of beta coefficients.
#Output:
#   Value of the LASSO objective function at specified inputs.
fx <- function(X,y,lambda,beta){
    f = (1/nrow(X)) * (t(y - X %*% beta) %*% (y - X %*% beta))
    g = lambda * sum(abs(beta))
    obj = (f+g)
    return(as.numeric(obj))
}

#Test:
fx(X,y,lam,beta_glmnet)

#————————————————————————————————————————————————
#Proximal L1 Operator function: (soft thresholding operator)
#Inputs:
#   x = vector of values.
#   lambda = the scaling factor of the l1 norm.
#   t = the step size.

#Output:
#   Value of the soft—thresholding proximal operator.

prox_l1 <- function(x,gamma,tau=1) {

    thresh <- gamma*tau
    prox = rep(0,length(x))

    idx.1 = which(x < -thresh)
    idx.2 = which(x > thresh)
```

```
    idx.3 = which(abs(x) <= thresh)

    if (length(idx.1) > 0) prox[idx.1] = x[idx.1] + thresh
    if (length(idx.2) > 0) prox[idx.2] = x[idx.2] - thresh
    if (length(idx.3) > 0) prox[idx.3] = 0

    return(prox)
}

#-----------------------------------------------
#Gradient for differentiable (non-penalty) part of LASSO objective:
gradient <- function(X,y,beta){
    grad = (2/nrow(X)) * (t(X) %*% X %*% beta - t(X) %*% y )
    return(grad)
}

#-----------------------------------------------
#Proximal Gradient Descent for L1 Norm Function:
#Inputs:
#   X = design matrix
#   y = response vector
#   gamma = step size
#   maxiter = maximum iterations
#   tol = tolerance for convergence
#   lambda = l1 norm penalty constant.
#Output:
#   List including estimated beta values and objective function.

proxGD <- function(X,Y,gamma=.01,maxiter=50,tol=1E-10,lambda=.1){

    i=0                 #Initialize iterator.
    converged <- 0      #Indicator for whether convergence met.

    #1. Initialize matrix to hold beta vector for each iteration.
    betas <- matrix(0,nrow=maxiter,ncol=ncol(X))
    betas[1,] <- rep(0,ncol(X)) #Initialize beta vector to 0 to start.

    #2. Initialize values for objective function.
    obj <- rep(0,maxiter)   #Initialize vector to hold loglikelihood fctn.
    obj[1] <- fx(X,y,lambda,betas[1,])

    #3. Initialize matrix to hold gradients for each iteration.
    grad <- matrix(0,nrow=maxiter,ncol=ncol(X))

    for (i in 2:maxiter){
        #STEP 1: Gradient Step.

        #Calc gradient.
        #grad[i-1,] = (2/nrow(X)) * (t(X) %*% X %*% betas[i-1,]  - t(X) %*% y )
        grad[i-1,] = gradient(X,y,betas[i-1,])

        #Determine intermediate point.
        z = betas[i-1,] - gamma*grad[i-1,]

        #STEP 2: Proximal step.
        betas[i,] = prox_l1(z,gamma,tau=lambda)

        #Update objective function.
        obj[i] = fx(X,y,lambda=lambda,beta=betas[i,])
```

```
        #Check if convergence met: If yes, exit loop.
        if (abs(obj[i]-obj[i-1])/abs(obj[i-1]+1E-3) < tol ){
            converged=1;
            break;
        }
    } #end for loop

    return(list(obj=obj, betas=betas, beta_hat=betas[i,], converged=converged,
        iter=i))
} #end function

#------------------------------------------------------------
#Accelerated Proximal Gradient Descent for L1 Norm Function:
#(Nesterov)
#Inputs:
#   X = design matrix
#   y = response vector
#   gamma = step size
#   maxiter = maximum iterations
#   tol = tolerance for convergence
#   lambda = l1 norm penalty constant.
#Output:
#   List including estimated beta values and objective function.

accelProxGD <- function(X,Y,gamma=.01,maxiter=50,tol=1E-10,lambda=.1){

    i=0                     #Initialize iterator.
    converged <- 0          #Indicator for whether convergence met.

    #1. Initialize matrix to hold beta vector for each iteration.
    betas <- matrix(0,nrow=maxiter,ncol=ncol(X))
    betas[1,] <- rep(0,ncol(X)) #Initialize beta vector to 0 to start.

    #2. Initialize values for objective function.
    obj <- rep(0,maxiter)    #Initialize vector to hold loglikelihood fctn.
    obj[1] <- fx(X,y,lambda,betas[1,])

    #3. Initialize matrix to hold gradients for each iteration.
    grad <- matrix(0,nrow=maxiter,ncol=ncol(X))
    grad[1,] =  gradient(X,y,betas[1,])

    #4. Initialize vectors to hold Nesterov update values.
    z = matrix(0,nrow=maxiter,ncol=ncol(X))
    s = rep(0,maxiter)

    #Set up first z value.  (Used a regular gradient calculation for beta0.)
    #z[1,] = betas[1,] - gamma * grad[1,]

    #Set up scalar s terms.  Ok before main loop, as do not depend on other terms'
         updates.
    for (j in 2:maxiter){
        s[j] = (1 + sqrt(1 + 4*(s[j-1])^2)) / 2
    }

    #Loop through iterations until converged or maxiter met.
    for (i in 2:maxiter){

        #STEP 1: Gradient Step.
```

```
        #Calc gradient.
        grad[i-1,] =  gradient(X,y,z[i-1,])

        #Update intermediate u term.
        u = z[i-1,] - gamma * grad[i-1,]

        #STEP 2: Proximal step; update betas.
        betas[i,] = prox_l1(u,gamma,tau=lambda)

        #STEP 3: Nesterov step; update Nesterov momentum z.
        z[i,] = betas[i-1,] + ((s[i-1]-1)/s[i]) * (betas[i-1,] - betas[i,])

        #Update objective function.
        obj[i] = fx(X,y,lambda=lambda,beta=betas[i,])

        #Check if convergence met: If yes, exit loop.
        #if (abs(obj[i]-obj[i-1])/abs(obj[i-1]+1E-10) < tol ){
        #   converged=1;
        #   break;
        #}
    } #end for loop

    return(list(obj=obj, betas=betas, beta_hat=betas[i,], converged=converged,
       iter=i,s=s))
} #end function

#------------------------------------------------------------

#Run proximal gradient descent & accelerated proximal gradient descent.
lam=.01
output <- proxGD(X,y,gamma=.01,maxiter=1000,tol=1E-10,lambda=lam)
outputAccel <- accelProxGD(X,y,gamma=.01,maxiter=1000,tol=1E-10,lambda=lam)

#Iterations to convergence:
print(output$iter)
print(outputAccel$iter)
print(output$converged)
print(outputAccel$converged)

#Compare results to glmnet:
myLasso <- glmnet(X,y,family='gaussian',lambda=lam) #fit glmnet model.
beta_glmnet <- myLasso$beta                          #Save glmnet betas.
cbind(glmnet=beta_glmnet,
    proximal=output$beta_hat,
    accel.prox=round(outputAccel$beta_hat,8))   #output comparison


#Plot objective function.
plot(1:output$iter,output$obj[1:output$iter],type='l',log='xy',col='blue',xlab=
   paste('iter ',1,' to ',output$iter),
    ylab='objective function')
lines(1:outputAccel$iter,outputAccel$obj[1:outputAccel$iter],type='l',col='red',
   xlab=paste('iter ',1,' to ',outputAccel$iter),
    ylab='objective function')

#Plot convergence of betas.
idx = which(output$beta_hat>0)
```

```
#idxPlot = sample(idx,9,replace=F)
for (j in idx){
    plot(1:length(output$betas[,j]),output$betas[,j],xlab='iter',ylab=paste('
        betahat',j),type='l',col='blue')
    abline(h=beta_glmnet[j],col='red')
}



#------------------------------------------------
#Run accelerated proximal gradient descent.
```