The proximal gradient method

(B)

LASSO regression problem is to solve $\hat{\beta} = argmin_{\beta}\{\|y - X\beta\|_2^2 + \lambda|\beta|_1\}$

Define $l(\beta) = \|y - X\beta\|_2^2 = (y - X\beta)^T(y - X\beta) = y^Ty - 2y^TX\beta + \beta^TX^TX\beta$

Then $\nabla l(\beta) = -2X^Ty + 2X^TX\beta = -2X^T(y - X\beta)$

Therefore, $\hat{\beta} = prox_{\gamma}\lambda|u|_1$, where $u = \beta - \gamma\nabla l(\beta) = \beta + 2\gamma X^T(y - X\beta)$.

Hence, $\hat{\beta} = sgn(u)(|u| - \gamma\lambda)_+$

Pseudo Code:

Step 1. Initiate β^0;

Step 2. For $t = 0, 1, 2, ...,$ calculate $u^t = \beta^t - \gamma^t\nabla l(\beta^t) = \beta + 2\gamma X^T(y - X\beta)$;

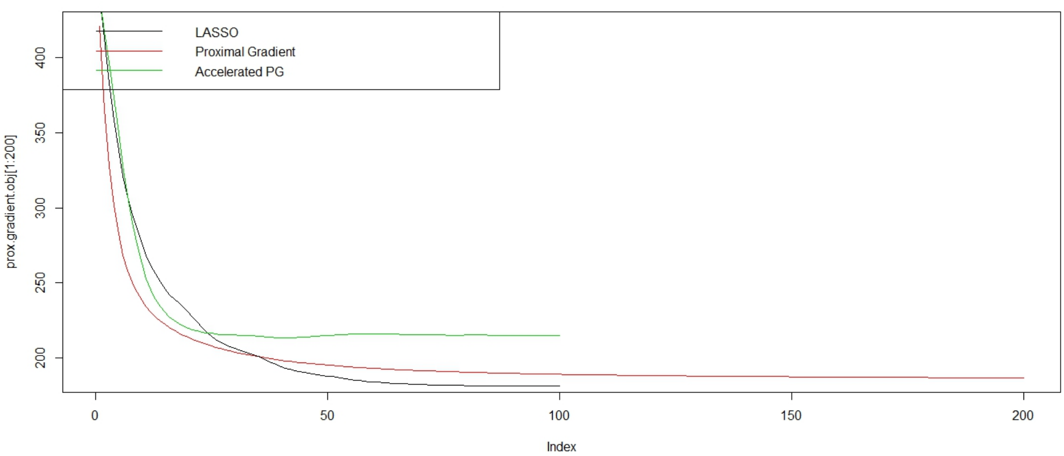
Calculate $\beta^{t+1} = prox_{\gamma}\phi(u^t) = sgn(u^t)(|u^t| - \gamma\lambda)_+$;

Step 3. Repeat Step 2 until $\hat{\beta}$ converges.

The primary computational cost is to calculate X^TX. There are also some minor calculations, such as the multiplication between vectors and matrix, vectors and vectors, and numbers.

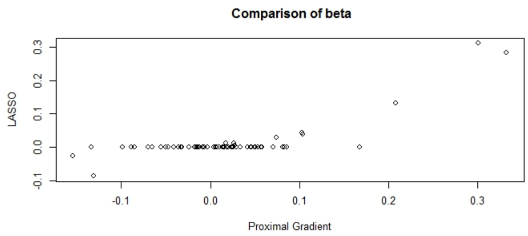
Comparison:

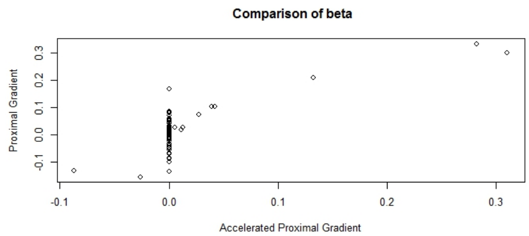
Following is a comparison of converging speed of 3 methods. Accelerated Proximal Gradient Method is the fastest, Lasso glmnet is the 2nd fast one, and Proximal Gradient Method is the slowest one among the three.



Following is a comparison of beta of 3 methods. The final result of Accelerated Proximal

Gradient Method is very similar to LASSO glmnet. However, Proximal Gradient Method converges significantly slower than the other 2 methods, and also has a lower sparsity.





Code:

```
Diabetes = read.csv( "C:/Users/Yuxin/Dropbox/Courses/2016 Fall/Stat Model for Big Data/E

y=scale(Diabetes$Y)
N=length(y)
p=ncol(Diabetes)-1
x=scale(Diabetes[, -1])

objective = function (x, y, beta, lambda){
obj=c()
N.fit=ncol( beta )
for (i in 1:N.fit){
obj.t=sum((y-x %*% beta[,i] )^2) + lambda*sum(abs(beta[,i]))
#obj.t=norm(y-x %*% beta[,i],type="2")^2 + lambda*sum(abs(beta[,i]))
obj = c( obj, obj.t )
}
return(obj)
}
```

```
gradient=function( x, y, beta ){

gradient = -2 * t(x) %*% (y - x %*% beta)

return(gradient=gradient)
}

ut=function(beta, gamma, gradient){

u.t = beta - gamma * gradient

return(u.t)
}

prox=function(ut, gamma, lambda){

prox=sign(ut) * pmax( ( abs(ut) -  gamma* lambda ), 0)

return(prox)
}

prox.gradient=function( p, y, x, beta1, lambda, gamma, maxiter, tol ){

beta=beta1
u=rep(0,p)

t=1

obj = sum((y-x %*% beta1)^2) + lambda*sum(abs(beta1))

while (t < maxiter  ){
```

```
grad=gradient( x, y, beta[,t] )
u.t=ut(beta[,t], gamma , grad )
beta.t1=prox( u.t, gamma, lambda )

u=cbind(u,u.t)
beta=cbind(beta, beta.t1)

obj.t= sum((y-x %*% beta.t1)^2) + lambda*sum(abs(beta.t1))
obj = c( obj , obj.t )
# if (abs(obj0 - obj) < tol){
# break
# }
# obj0 = obj

t=t+1
}

return( list( beta=beta, u=u, t=t, obj=obj ) )
}



beta1=matrix(0.01, nrow=p, ncol=1)
lambda=0.01*p
gamma=0.02/N
maxiter=500
tol=1e-8

fit.prox.gradient = prox.gradient( y=y,x=x, beta1=beta1, lambda=lambda, gamma=gamma, max

#### Used Tracy Yang's code for APG ####

obj <- function(y,X,beta,lambda){
```

```
        A = y - X %*% beta
        l = (0.5 / nrow(X)) * crossprod(A)
        phi = lambda * sum(abs(beta))
        obj = l + phi
        return(as.numeric(obj))
}

gradl <- function (y, X, beta){
    grad = (1 / nrow(X)) * (crossprod(X) %*% beta - t(X) %*% y)
    return(grad)
}

prox <- function (x, gamma, lambda){
    r = sign(x) * pmax(rep(0, length(x)), abs(x) - gamma * lambda)
    return(r)
}


# Accerlerated proximal gradient algorithm

APG <- function(y,X,gamma , lambda , num.iteration , tol ){
    p = ncol(X)

    # initialize values
    beta_old = rep(0, p)
    z_old = rep(0,p)

    # initialize matrix to hold all betas
    beta.path = matrix(NA, nrow = num.iteration,ncol = p)
    beta.path[1,] = beta_old
    z.path = matrix(NA, nrow = num.iteration,ncol = p)
    z.path[1,] = z_old
```

```
  s = array(NA, dim = num.iteration)
  s[1] = 1

  # create empty vector to stor objective value
  objective = c()

  for (j in 2:num.iteration){
    s[j] = 0.5 * (1 + sqrt(1 + 4 * s[j - 1] ^2))
  }

  for (iter in 2: num.iteration){
      gradient <- gradl(y, X, z.path[iter-1, ])
      u <- z.path[iter-1, ] - gamma * gradient

      # Update beta
      beta.path[iter, ] <- prox(u, gamma, lambda)
      s[iter] <- (1 + sqrt(1 + 4 * s[iter-1]^2))/2
      z.path[iter, ] <- beta.path[iter, ] + ((s[iter-1] - 1)/s[iter]) * (beta.path[ite

      # Compute log-likelihood
      objective = c(objective, obj(y,X, beta.path[iter, ], lambda))

      #if (abs(obj(y,X,beta.path[iter, ], lambda) - obj(y,X,beta.path[iter-1, ],lambda
      #   break
      #}


  }
  return(list(beta = beta.path[iter,], objective = objective, iter = iter, betas = bet
}
```