# SDS 385 Homework 2

Qiaohui Lin & Yuguang Yue

$21^{\text{st}}$ Oct, 2018

## 1 Question 1: Subgradient in case of Quadratic Form

### 1.1 Part 1 condition of convergence

$f(x) = \frac{1}{2}x^T Ax + b^T x$ is differentiable, subgradient is gradient.

$$\begin{aligned}
x_{k+1} &= x_k - \alpha\nabla f(x_k) \\
x_{k+1} - x^* &= x_k - x^* - \alpha(\nabla f(x_k) - \nabla f(x^*)) \\
&= x_k - x^* - \alpha(Ax - Ax^*) \\
||x_{k+1} - x^*|| &\leq ||I - \alpha A|| * ||x_k - x^*|| \\
&\leq max(|1 - \alpha\mu|, |1 - \alpha Ł|) * ||x_k - x^*|| \\
&\text{where } \mu, L \text{ are the smallest and largest eigenvalues of A}
\end{aligned}$$

It is converging as long as $max(|1 - \alpha\mu|, |1 - \alpha Ł|) < 1$, i.e.,$0 < \alpha < \frac{2}{L}$,$L$ is the largest eigenvalue of A.

### 1.2 Part 2 stepsize at fastest convergence

Fastest convergence is achieved by minimizing $max(|1 - \alpha\mu|, |1 - \alpha Ł|)$, at $\alpha = \frac{2}{L+\mu}$, with $||x_{k+1} - x^*|| \leq \frac{L-\mu}{L+\mu}||x_t - x^*||$

## 2 Question 2

### 2.1 Part 1 example of subgradient not a descent method

Based on the definition of subgradient, $f(y) \geq f(x) + g^T(y-x)$. $x_{k+1} = x_k - \alpha g_k$, if $-g_k$ is not a descent direction, i.e., $f'(x, -g^k) = \lim_{h->0} \frac{f(x-hg)-f(x)}{h} \geq 0$, then we always have $f(x_{k+1}) > f(x_k)$. Thus, it suffices to find a $g$ that is not a descent direction.
A simple example is $f(x) = |x|$,pick $g = -0.5$ at 0,

$$\begin{aligned} f'(x, -g) &= \lim_{h \to 0} \frac{f(x-hg) - f(x)}{h} \\ &= \frac{0.5h}{h} = 0.5 > 0 \end{aligned}$$

$g = -0.5$ is not a descent direction for $f(x) = |x|$
An example that is commonly used in multivariate case is $f(x_1, x_2) = |x_1| + |x_2|$. $g = (1, 2)$ is a subgradient at $(1, 0)$, i.e. $g \in \partial f(1, 0)$ but $-g$ is not a descent direction at (1,0).

$$\begin{aligned} x - \alpha * g = (1,0) - \alpha * (1,2) = (1-\alpha, -2\alpha) \\ f(x - \alpha * g) = f((1-\alpha, -2\alpha)) = (1 - \alpha + 4\alpha = 1 + 3\alpha > 1 = f(x) \end{aligned}$$

Thus,$-g^k$ is not a descent direction, subgradient is not a descent method at this point.

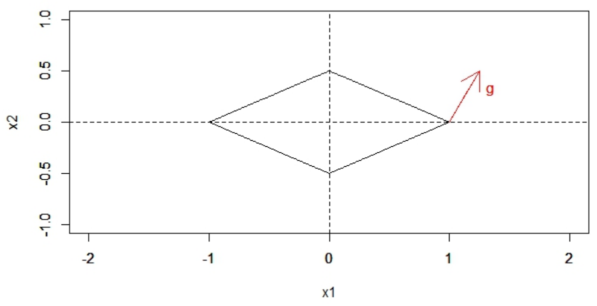

**Figure 1:** An example of g is not descent direction in a two variable case

### 2.2 Part 2 Prove convergence if $\alpha < \frac{2(f(x_t)-f(x^*))}{||g||^2}$

$$\begin{aligned}||x_{t+1}-x^*||^2 &= ||x_t - \alpha * g - x^*||^2 \\ &\leq ||x_t - x^*||^2 - 2 * ||x_t - x^*|| * \alpha * g + \alpha^2 ||g^2|| \\ &\text{based on definition, } f(x_t) \geq f(x^*) + g * (x_t - x^*) \\ &\leq ||x_t - x^*||^2 - 2 * \alpha(f(x_t) - f(x^*)) + \alpha^2 ||g^2|| \\ &\leq ||x_t - x^*||^2 - \alpha[2(f(x_t) - f(x^*)) - \alpha ||g^2||]\end{aligned}$$

We have $||x_{t+1} - x^*||^2 \leq ||x_t - x^*||^2$ if $2 * (f(x_t) - f(x^*)) - \alpha ||g^2|| < 0$, i.e., $\alpha < \frac{2(f(x_t)-f(x^*))}{||g||^2}$

# 3 Question 3

## 3.1 Part 1: subgradient with fixed and decreasing stepsize

**[[code in the appendix]]**
Goal: Minimize loss function $\min ||y - X\beta||^2 + \lambda||\beta||_1 = g(\beta) + h(\beta)$
Simulation: We generate a $10^5 * 500$ dataset of $X$ and, $500*1$ sparse $\beta_{true}$ with random 13 1s (with index 11,22,108,166,175,208,254,355,385,406,412,457,472) and 10 $-1$s (with index 61,170,187,225,284,291,308,332,338,466) and $y = X\beta_{true}$.

The subgradient for the non-differentiable part $||\beta||_1 = \sum_{p=1}^{500} |\beta_p|$ is

$$subg_p = \partial h_p = \begin{cases} 1 & \beta_p > 0 \\ w \in (-1, 1) & \beta_p = 0 \\ -1 & \beta_p < 0 \end{cases}$$

Then the update is:

$$\beta^{t+1} = \beta^t - \alpha * (\nabla g(\beta^t) + \lambda * subg_p)$$

Pick $\alpha_{fixed} = 0.1$, $\alpha_{decreased} = 1/t$ at iteration t in subgradient, intiate $\beta$ from uniform(-1,1), set convergence criterion as $||\beta^{t+1} - \beta^t||_2^2 < 1e-6$. We first choose $\lambda$ using cross validation. Due to the restriction of running time, we pick the $\lambda$ grid of only values $(0.03, 0.05, 0.1, 0.25, 0.5)$. By the One-SE rule, we choose $\lambda = 0.03$. The plot of cross validation loss regard to lambda in our grid of $\lambda$ is:

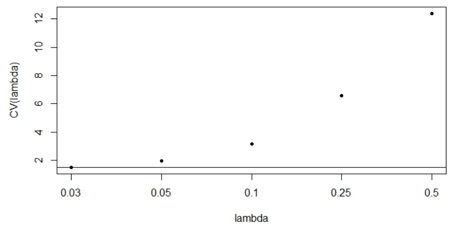

**Figure 2:** Cross-Validation to choose *lambda*, within our grid of possible $\lambda$s, we chose $\lambda = 0.03$

The fixed stepsize subgradient method takes 47 iterations and decreasing stepsize subgradient method only takes 30 iterations to converge.
The loss function with respect to number of iterations is plotted below:

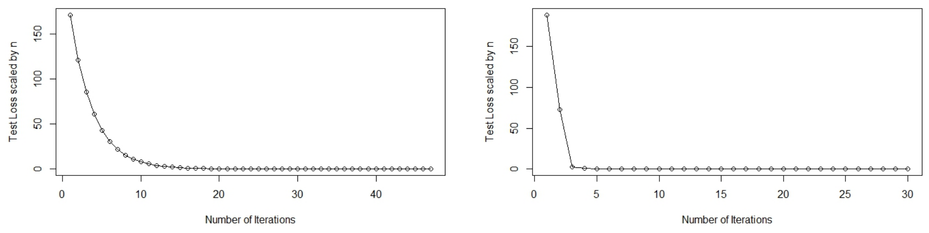

**Figure 3:** Left: loss function scaled by n for subgradient method of fixed stepsize $\alpha = 0.1$, Right: loss scaled by n for subgradient with decreasing stepsize $\alpha = 1/t$ at each iteration $t$.

Both methods successfully picked out the non-zero $\beta$s from 500 $\beta$s and the rest are 0. We only post one example of subgradient method of fixed alpha size. The rest is redundant.

**Table 1:** Non-zero$\beta$s and Estimates from Subgradients of Fixed Step-size Method

| Positive Index | Estimate | Negative Index | Estimate |
|---:|---:|---:|---:|
| 11 | 0.9998868 | 61 | -0.9998469 |
| 11 | 0.9999412 | 61 | -0.9996028 |
| 22 | 0.9996374 | 170 | -0.9998474 |
| 108 | 0.9998609 | 187 | -0.9996856 |
| 166 | 0.9993086 | 225 | -0.9996685 |
| 175 | 0.9992656 | 284 | -0.9997776 |
| 208 | 0.9996314 | 291 | -0.9995107 |
| 254 | 0.9995262 | 308 | -1.0000977 |
| 355 | 0.9993518 | 332 | -0.9992680 |
| 385 | 1.0000368 | 338 | -0.9997206 |
| 406 | 0.9996223 | 466 | -0.9999831 |
| 412 | 0.9994649 | 61 | -0.9996028 |
| 457 | 1.0001099 | 170 | -0.9998474 |
| 472 | 1.0000488 | 187 | -0.9996856 |

## 3.2 Part 2: proximal method

[[**code in the appendix**]]
With the same goal and simulation, we update $\beta$ by

$$\beta^{t+1} = prox_{\lambda\alpha}(\beta^t - \nabla g(\beta_t))$$

First calculate the gradient update of the differentiable part:

$$\beta^t - \nabla g(\beta_t) = \beta^t + \alpha X^T(y - X\beta^t)$$

Then do the proximation:

$$prox_{\lambda\alpha}(\beta^t - \nabla g(\beta_t)) = argmin_z \frac{1}{2\alpha}||\beta - z||^2 + \lambda||z||_1$$

, which results in the Soft Threshold Operator for $p = 1, 2, , 500$:

$$z_p^t = S_{\lambda\alpha}(\beta_p^t) = \begin{cases} \beta_p^t - \lambda\alpha & \beta_p^t > 0 \\ 0 & |\beta_p^t| < \lambda\alpha \\ \beta_p^t + \lambda\alpha & \beta_p^t < -\lambda\alpha \end{cases}$$

$\lambda$ is also chosen via cross validation. We chose $\lambda = 0.03$ in our range of $\lambda$s. The loss with respect to number of iterations is plotted below. With the same convergence criterion and fixed $\alpha = 0.1$, proximal method takes 34 iterations to converge. The selected non-zero $\beta$s are the same as previous methods with very close estimates. Tables are omitted.

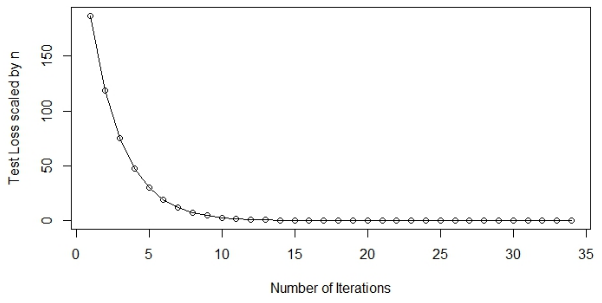

**Figure 4:** Loss function scaled by n for proximal method of fixed stepsize $\alpha = 0.1$

### 3.3 Part 3: proximal method with backtracking line search

[[**code in the appendix**]]
We still use the same Soft Threshold Operator as part 2. The only thing changes is that we do a backtracking line search in the gradient calculation part before the thresholding proximal, in which case the step size $r$ are searched through backtracking as:

```
    **Repeat r:=br, starting from r=1,**$a \in (0, 1/2)$,$b \in (0, 1)$
    **until** $f(x - r\nabla g(x)) \leq f(x) - \alpha r \nabla g(x)^T \nabla g(x)$
```

($\lambda$ is also chosen via cross validation. We chose $\lambda = 0.03$ in our range of $\lambda$s.) Though eventually only 8 mega-iterations on Soft Threshold Operator gives us the same *beta* estimates before, the actually time of iteration is longer since each mega iteration needs many small iterations to search for $r$. The eventual loss function is plotted below. We can also notice because of the exact line search in each mega iteration, the loss is much smaller than other methods at the beginning.

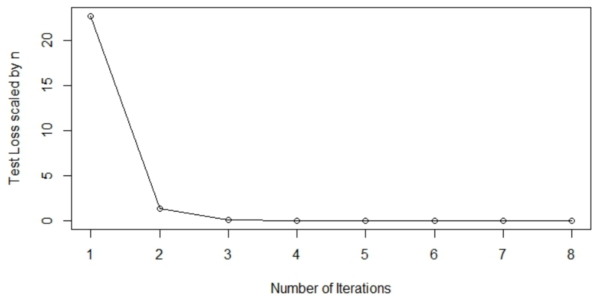

**Figure 5:** Loss function scaled by n for proximal method of backtracking line search

### 3.4 Par4: compare with glmnet in R

[[**code in the appendix**]]
We compare what we have in previous sections with the glmnet and cv.glmnet function from glmnet package in R, which is commonly used in lasso regression. glmnet and cv.glmnet returns the same $\beta$ estimate as in Table 1, but with more choices of $\lambda$. Our choice of $\lambda$ is restricted by the computational cost. We eventually choose very close $\lambda$s as the cv.glm function with its default grid. However, with the package, we cannot visualize the loss with each iteration and cannot know how many iterations it takes exactly for convergence. But we can see the result for the whole $\lambda$ grid and easily choose the best fit $\lambda$ potentially with AIC, BIC or other rules and do not need to manually repeated choose the $\lambda$.

## 4 Appendix

```
set.seed(123)
library(MASS)
library(glmnet)
X=mvrnorm(n=10^5,mu=rep(0,500),Sigma=diag(500))
b_true=rep(0,500)
b_true[sample(1:500,15)]=1
b_true[sample(1:500,10)]=-1
```

```
y=X%*%b_true

trainlabel <- sample(1:10^5,0.8*10^5,replace = FALSE)
xtrain <- X[trainlabel,]
ytrain <- y[trainlabel]
xtest <- X[-trainlabel,]
ytest <- y[-trainlabel]


lambdagrid <- c(0.03,0.05,0.1,0.25,0.5)
alpha_fixed=0.1
alpha_decrs <- NULL

L <- function(b,lambda,y,X){
  sum((y-X%*%b)^2) + lambda*sum(abs(b))
}

#subgradient with alphafixed
iter =10^4
n=10^5
b_subg <- matrix(runif(n=500*(iter+1),-1,1),500,
ncol = (iter+1))
cv <- matrix(0,length(lambdagrid),5)
for(l in 1:length(lambdagrid)){
  lambda <- lambdagrid[l]
 for(k in 1:5){
   validlabel <- (0.8*n/5*(k-1)+1):(0.8*n/5*k)
   xvalid <- xtrain[validlabel,]
   yvalid <- ytrain[validlabel]
   xtrcv <- xtrain[-validlabel,]
   ytrcv <- ytrain[-validlabel]
   b_subg <- matrix(runif(n=500*(iter+1),-1,1),500,
   ncol = (iter+1))
  for (i in 1:iter){
  subg <- NULL
  for (j in 1:500){
    if (b_subg[j,i]<0){
      subg[j]=-1}
    else if (b_subg[j,i]>0){
      subg[j]=1
    }
    else if(b_subg[j,]==0){
      subg[j]=runif(1,-1,1)
    }
  }
  g=(-2*t(xtrcv)%*%(ytrcv-xtrcv%*%b_subg[,i])
   +lambda*subg)/n
  b_subg[,i+1]=b_subg[,i]-alpha_fixed*g
  if (sum((b_subg[,i+1]-b_subg[,i])^2)< 10^-6){
    break
  }
  }
   cv[l,k] <- L(b_subg[,i+1],lambda,yvalid,xvalid)
}
}

#plot out cv of lambdas
plot(rowMeans(cv),pch=20,xlab='lambda',
ylab='CV(lambda)',xaxt='n')
axis(1,at=c(1:5),label=lambdagrid)
abline(h=(rowSums((cv-rowMeans(cv))^2)/5)
[which.min(rowMeans(cv))] +min(rowMeans(cv)))

# from cv,we choose lambda=0.03
lambda=0.03
b_subg <- matrix(runif(n=500*(iter+1),-1,1),
500,ncol = (iter+1))
for (i in 1:iter){
  subg <- NULL
  for (j in 1:500){
    if (b_subg[j,i]<0){
      subg[j]=-1}
    else if (b_subg[j,i]>0){
      subg[j]=1
    }
    else if(b_subg[j,]==0){
      subg[j]=runif(1,-1,1)
    }
  }
  g=(-2*t(xtrain)%*%(ytrain-xtrain%*%b_subg[,i])
  +lambda*subg)/n
  b_subg[,i+1]=b_subg[,i]-alpha_fixed*g
  if (sum((b_subg[,i+1]-b_subg[,i])^2)< 10^-6){
    break
  }
}
i_subg=i
print(c(which(b_subg[,i+1]>1e-1),
b_subg[which(b_subg[,i+1]>1e-1),i+1]))
print(c(which(b_subg[,i+1]< -0.1),
b_subg[which(b_subg[,i+1]< -0.1),i+1]))
kable(cbind(which(b_subg[,i+1]>1e-1),
b_subg[which(b_subg[,i+1]>1e-1),i+1],
which(b_subg[,i+1]< -0.1),
b_subg[which(b_subg[,i+1]< -0.1),i+1])
,format='latex')


#decreasing stepsize
iter =10^4
b_subgad <- matrix(runif(n=500*(iter+1),-1,1),
500,ncol = (iter+1))
cv_subgad <- matrix(0,length(lambdagrid),5)
for(l in 1:length(lambdagrid)){
  lambda <- lambdagrid[l]
  for(k in 1:5){
    validlabel <- (0.8*n/5*(k-1)+1):(0.8*n/5*k)
    xvalid <- xtrain[validlabel,]
    yvalid <- ytrain[validlabel]
    xtrcv <- xtrain[-validlabel,]
    ytrcv <- ytrain[-validlabel]
    b_subgad <- matrix(runif(n=500*(iter+1),-1,1),
    500,ncol = (iter+1))
for (i in 1:iter){
```

```
  subgad <- NULL
  for (j in 1:500){
    if (b_subgad[j,i]<0){
      subgad[j]=-1}
    else if (b_subgad[j,i]>0){
      subgad[j]=1
    }
    else if(b_subgad[j,]==0){
      subgad[j]=runif(1,-1,1)
    }
  }
  gad=(-2*t(X)%*%(y-X%*%b_subgad[,i])+lambda*subgad)/n
  #alpha_decrs[i] <- 0.1/sqrt(i)
  alpha_decrs[i] <- 1/i
  b_subgad[,i+1]=b_subgad[,i]-alpha_decrs[i]*gad
  if (sum((b_subgad[,i+1]-b_subgad[,i])^2)< 10^-6){
    break
  }
}
    cv_subgad[l,k] <- L(b_subgad[,i+1],
    lambda,yvalid,xvalid)
  }
}

plot(rowMeans(cv_subgad),pch=20,xlab='lambda',
ylab='CV(lambda)',xaxt='n')
axis(1,at=c(1:5),label=lambdagrid)
abline(h=(rowSums((cv_subgad-rowMeans(cv_subgad))^2)/5)
[which.min(rowMeans(cv_subgad))]+min(rowMeans(cv_subgad)))

#we choose lambda=0.03
for (i in 1:iter){
  subgad <- NULL
  for (j in 1:500){
    if (b_subgad[j,i]<0){
      subgad[j]=-1}
    else if (b_subgad[j,i]>0){
      subgad[j]=1
    }
    else  if(b_subgad[j,]==0){
      subgad[j]=runif(1,-1,1)
    }
  }
  gad=(-2*t(xtrain)%*%(ytrain-xtrain%*%b_subgad[,i])
  +lambda*subgad)/n
  alpha_decrs[i] <- 1/i
  b_subgad[,i+1]=b_subgad[,i]-alpha_decrs[i]*gad
  if  (sum((b_subgad[,i+1]-b_subgad[,i])^2)<  10^-6){
    break
  }
}
i_subgad <- i
print(c(which(b_subgad[,i+1]>  0.1),
b_subgad[which(b_subgad[,i+1]>0.1),i+1]))
print(c(which(b_subgad[,i+1]<  -0.1),
b_subgad[which(b_subgad[,i+1]<  -0.1),i+1]))



#proximal  method
iter  =10^4
b_prox<- matrix(runif(n=500*(iter+1),-1,1),
500,ncol = (iter+1))
z_prox<- matrix(0,500,ncol = (iter+1))
cv_prox <- matrix(0,length(lambdagrid),5)
for(l  in  1:length(lambdagrid)){
  lambda <- lambdagrid[l]
  for(k  in  1:5){
    validlabel <- (0.8*n/5*(k-1)+1):(0.8*n/5*k)
    xvalid <- xtrain[validlabel,]
    yvalid <- ytrain[validlabel]
    xtrcv <- xtrain[-validlabel,]
    ytrcv <- ytrain[-validlabel]
    b_prox <- matrix(runif(n=500*(iter+1),-1,1),
    500,ncol = (iter+1))
    z_prox<-  matrix(runif(n=500*(iter+1),-1,1),
    500,ncol = (iter+1))
for (i in 1:iter){
  g=-2*t(xtrcv)%*%(ytrcv-xtrcv%*%z_prox[,i])/n
  b_prox[,i+1]=z_prox[,i]-alpha_fixed*g
  for (j in 1:500){
    if (b_prox[j,i+1]>lambda*alpha_fixed){
      z_prox[j,i+1]=b_prox[j,i+1]-lambda*alpha_fixed
    }
    else if(b_prox[j,i+1]< -lambda*alpha_fixed){
      z_prox[j,i+1]=b_prox[j,i+1]+lambda*alpha_fixed
    }
    else if(abs(b_prox[j,i+1]) <= lambda*alpha_fixed){
      z_prox[j,i+1]=0
    }
  }
  if (sum((z_prox[,i+1]-z_prox[,i])^2)< 10^-6){
    break
  }
}
    cv_prox[l,k] <- L(z_prox[,i+1],lambda,yvalid,xvalid)
  }
}

plot(rowMeans(cv_prox),pch=20,xlab='lambda',
ylab='CV(lambda)',xaxt='n')
axis(1,at=c(1:5),label=lambdagrid)
abline(h=(rowSums((cv_prox-rowMeans(cv_prox))^2)/5)
[which.min(rowMeans(cv_prox))]+min(rowMeans(cv_prox)))



# We choose lambda=0.03
lambda=0.03
b_prox<- matrix(runif(n=500*(iter+1),-1,1),
500,ncol = (iter+1))
z_prox<- matrix(runif(n=500*(iter+1),-1,1),
500,ncol = (iter+1))
for (i in 1:iter){
  g=-2*t(X)%*%(y-X%*%z_prox[,i])/n
  b_prox[,i+1]=z_prox[,i]-alpha_fixed*g
  for (j in 1:500){
    if (b_prox[j,i+1]>lambda*alpha_fixed){
      z_prox[j,i+1]=b_prox[j,i+1]-lambda*alpha_fixed
    }
    else if(b_prox[j,i+1]< -lambda*alpha_fixed){
      z_prox[j,i+1]=b_prox[j,i+1]+lambda*alpha_fixed
    }
    else if(abs(b_prox[j,i+1]) <= lambda*alpha_fixed){
      z_prox[j,i+1]=0
    }
  }
  if (sum((z_prox[,i+1]-z_prox[,i])^2)< 10^-6){
    break
  }
}
i_prox <- i
print(c(which(z_prox[,i_prox+1]>1e-1),
z_prox[which(z_prox[,i_prox+1]>1e-1),i_prox+1]))
print(c(which(z_prox[,i_prox+1]< -0.1),
z_prox[which(z_prox[,i_prox+1] < -0.1),i_prox+1]))


#proximal method with backtracking line search
bt_a=0.3
bt_b=0.7
f <- function(beta,y,X){
  sum((y-X%*%beta)^2)
}
cv_prox_bt <- matrix(0,length(lambdagrid),5)
for(l in 1:length(lambdagrid)){
  lambda <- lambdagrid[l]
  for(k in 1:5){
    validlabel <- (0.8*n/5*(k-1)+1):(0.8*n/5*k)
    xvalid <- xtrain[validlabel,]
    yvalid <- ytrain[validlabel]
    xtrcv <- xtrain[-validlabel,]
    ytrcv <- ytrain[-validlabel]
    b_prox_bt <- matrix(runif(n=500*(iter+1),-1,1),
    500,ncol = (iter+1))
    z_prox_bt <- matrix(runif(n=500*(iter+1),-1,1),
    500,ncol = (iter+1))
for (i in 1:iter){
  g=-2*t(xtrcv)%*%(ytrcv-xtrcv%*%z_prox_bt[,i])
  t=1
  for (m in 1:100){
    if(f(z_prox_bt[,i]-t*g,ytrcv,xtrcv)<
    f(z_prox_bt[,i],ytrcv,xtrcv)-(bt_a*t)*(t(g)%*%g)[1][1])
    {
      break
    }
    t=t*bt_b
    m=m+1
  }
  b_prox_bt[,i+1]=z_prox_bt[,i]-t*g
  for (j in 1:500){
    if (b_prox_bt[j,i+1]>lambda*alpha_fixed){
      z_prox_bt[j,i+1]=b_prox_bt[j,i+1]
                       -lambda*alpha_fixed
    }
    else if(b_prox_bt[j,i+1]< -lambda*alpha_fixed){
      z_prox_bt[j,i+1]=b_prox_bt[j,i+1]
                          +lambda*alpha_fixed
    }
    else if(abs(b_prox_bt[j,i+1]) <= lambda*alpha_fixed){
      z_prox_bt[j,i+1]=0
    }
  }
  if (sum((z_prox_bt[,i+1]-z_prox_bt[,i])^2)< 10^-6){
    break
  }
}
    cv_prox_bt[l,k] <- L(z_prox_bt[,i+1],lambda,
                          yvalid,xvalid)
  }
```

```
}

plot(rowMeans(cv_prox_bt),pch=20,
xlab='lambda',ylab='CV(lambda)',xaxt='n')
axis(1,at=c(1:5),label=lambdagrid)
abline(h=(rowSums((cv_prox_bt-rowMeans(cv_prox_bt))^2)/5)
[which.min(rowMeans(cv_prox_bt))]+min(rowMeans(cv_prox_bt)))



#we choose lambda=0.03
lambda=0.03
b_prox_bt<- matrix(runif(n=500*(iter+1),-1,1),
500,ncol = (iter+1))
z_prox_bt<- matrix(runif(n=500*(iter+1),-1,1),
500,ncol = (iter+1))
for (i in 1:iter){
  g=-2*t(xtrain)%*%(ytrain-xtrain%*%z_prox_bt[,i])
  t=1
  for (m in 1:100){
    if(f(z_prox_bt[,i]-t*g,ytrain,xtrain)
    <f(z_prox_bt[,i],ytrain,xtrain)-(bt_a*t)*(t(g)%*%g)[1][1]){
      break
    }
    t=t*bt_b
    m=m+1
  }
  b_prox_bt[,i+1]=z_prox_bt[,i]-t*g
  for (j in 1:500){
    if (b_prox_bt[j,i+1]>lambda*alpha_fixed){
      z_prox_bt[j,i+1]=b_prox_bt[j,i+1]-lambda*alpha_fixed
    }
    else if(b_prox_bt[j,i+1]< -lambda*alpha_fixed){
      z_prox_bt[j,i+1]=b_prox_bt[j,i+1]+lambda*alpha_fixed
    }
    else if(abs(b_prox_bt[j,i+1]) <= lambda*alpha_fixed){
      z_prox_bt[j,i+1]=0
    }
  }
  if (sum((z_prox_bt[,i+1]-z_prox_bt[,i])^2)< 10^-6){
    break
  }
}

i_prox_bt <- i
print(c(which(z_prox_bt[,i_prox_bt+1]>1e-1),
z_prox_bt[which(z_prox_bt[,i_prox_bt+1]>1e-1)
,i_prox_bt+1]))
print(c(which(z_prox_bt[,i_prox_bt+1]< -0.1),
z_prox_bt[which(z_prox_bt[,i_prox_bt+1] < -0.1)
,i_prox_bt+1]))


#compare with glmnet
cvfit <- cv.glmnet(x=X,y=y)
fit1 <- glmnet(x=X,y=y)


#plot all the objective functions
L_subgtr <- NULL
for (i in 1:i_subg){
  L_subgtr[i] <- L(b_subg[,i],lambda,ytrain,xtrain)
}
plot(L_subgtr/(0.8*n),type='o',
xlab='Number of Iterations',ylab='Training Loss scaled by n')

L_subgts <- NULL
for (i in 1:i_subg){
  L_subgts[i] <- L(b_subg[,i],lambda,ytest,xtest)
}
plot(L_subgts/(0.2*n),type='o',
xlab='Number of Iterations',ylab='Test Loss scaled by n')

L_subgadtr <- NULL
for (i in 1:i_subgad){
  L_subgadtr[i] <- L(b_subgad[,i],lambda,ytrain,xtrain)
```

```
}
plot(L_subgadtr/(0.8*n),type='o',
xlab='Number of Iterations',ylab='Training Loss scaled by n')

L_subgadts <- NULL
for (i in 1:i_subgad){
  L_subgadts[i] <- L(b_subgad[,i],lambda,ytest,xtest)
}
plot(L_subgadts/(0.2*n),type='o',
xlab='Number of Iterations',ylab='Test Loss scaled by n')

#Proximal
L_proxtr <- NULL
for (i in 1:i_prox){
  L_proxtr[i] <- L(z_prox[,i],lambda,ytrain,xtrain)
}
plot(L_proxtr/(0.8*n),type='o',
xlab='Number of Iterations',ylab='Training Loss scaled by n')

L_proxts <- NULL
for (i in 1:i_prox){
  L_proxts[i] <- L(z_prox[,i],lambda,ytest,xtest)
}
plot(L_proxts/(0.2*n),type='o',
xlab='Number of Iterations',ylab='Test Loss scaled by n')

#Proximal BT
L_prox_bttr <- NULL
for (i in 1:i_prox_bt){
  L_prox_bttr[i] <- L(z_prox_bt[,i],lambda,ytrain,xtrain)
}
plot(L_prox_bttr/(0.8*n),type='o',
xlab='Number of Iterations',ylab='Training Loss scaled by n')

L_prox_btts <- NULL
for (i in 1:i_prox_bt){
  L_prox_btts[i] <- L(z_prox_bt[,i],lambda,ytest,xtest)
}
```

```
plot(L_prox_btts/(0.2*n),type='o',
xlab='Number of Iterations',ylab='Test Loss scaled by n')
```