Problem 2 Proximal Gradient for Lasso

Wei-Chih Chen
ID: 912448766

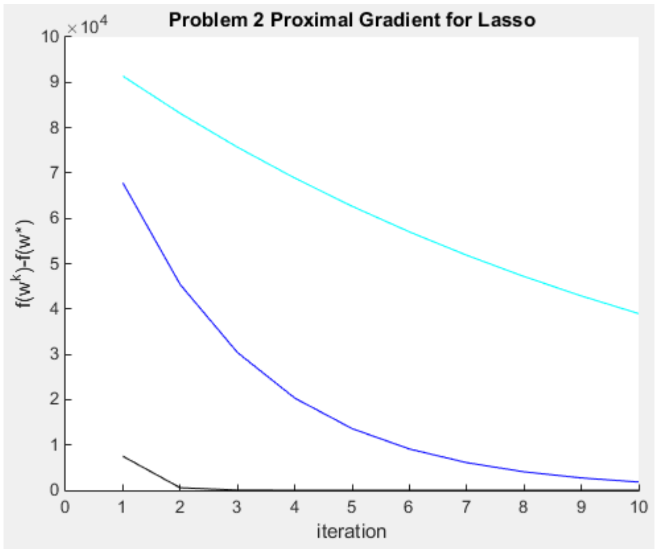
(a)

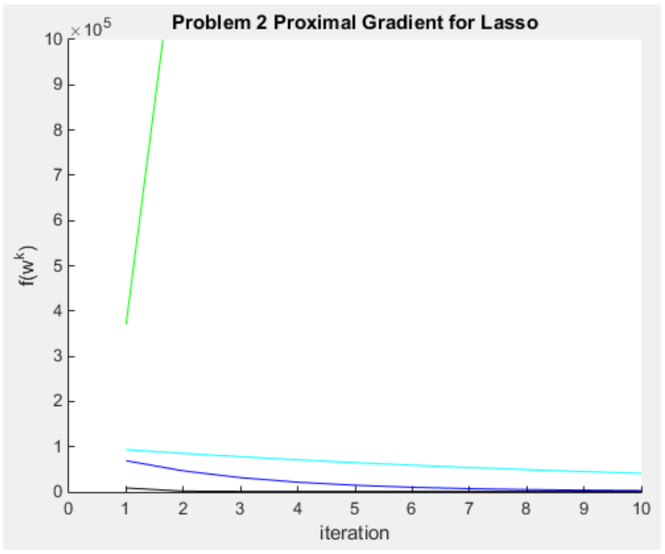
Step size	2^{-14}	2^{-16}	2^{-18}	2^{-20}	2^{-22}
$f(w^{50})$	5.0701×10^{107}	1.3067×10^{33}	1.1964×10^{3}	1.1964×10^{3}	2.1616×10^{3}
Elements in w^{50} nonzero	150348	150348	15	226	2772

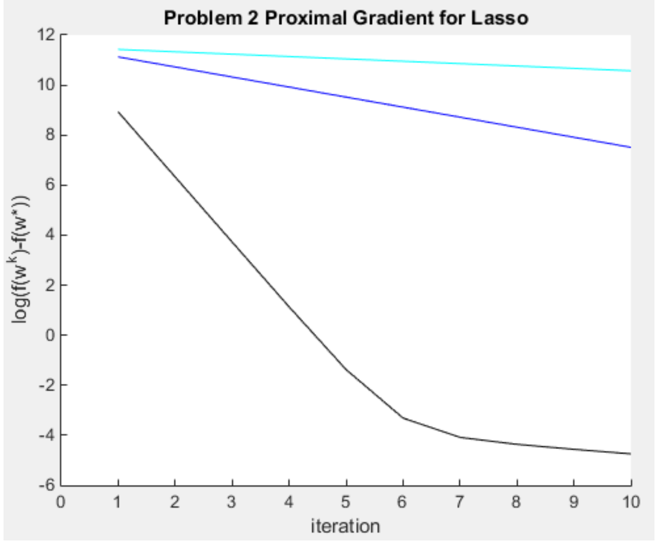
```
%Problem 2 Proximal Gradient for Lasso
load('E2006_matlab.mat');
lamda = 1;
dimnum = size(X,2); %x dimension number
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Problem 2 Proximal Gradient for Lasso
%(a)
eta =[2^-14, 2^-16, 2^-18, 2^-20, 2^-22];
countNonZero = zeros(5,1);
CostHis = zeros(5,50);
for etanum=1:5
    w = zeros(dimnum,1);
    compare = eta(etanum)*lamda;
    %Proximal Gradient Method
    for i=1:50
        %calculate dg
        dg1 = X * w - y;
        dg = X.' * dg1;
        wbar = w - eta(etanum) * dg;
        %soft Threshold
        w = sign(wbar).*max(abs(wbar)-compare,0);
        CostHis(etanum, i) = 0.5*norm(X * w - y,2)^2 + lamda*norm(w,1);
    end % end Proximal Gradient Method for 50 iterations

    for i=1:dimnum
        if w(i,1) ~= 0
            countNonZero(etanum) = countNonZero(etanum)+ 1;
        end
    end
end % end different eta
```

(b)

First two step sizes are too large so it cannot show in graph. 2^{-18}, 2^{-20} and 2^{-22} step sizes are decree object function, but 2^{-18} decrease relatively fast. Based on the graph we got, step size $\mathbf{2^{-18}}$ is better for this algorithm.







Top left is $f(w^k)$- $f(w^*)$ VS iterations, top right is $f(w^k)$ VS iterations and bottom left is $\log(f(w^k)- f(w^*))$ VS iterations.

Red: 2^{-14}
Green: 2^{-16}
Black: 2^{-18}
Blue: 2^{-20}
Cyan: 2^{-22}

```
%(b)
err = zeros(5,10);
for j=1:5
    for i = 1 : 10
        err(j,i) = CostHis(j,i)-CostHis(j, 50);
    end
end
```

```
figure
hold all
axis([0 10 0 10e5]);
plot(err(1,:),'r')
plot(err(2,:),'g')
plot(err(3,:),'k')
plot(err(4,:),'b')
plot(err(5,:),'c')

title('Problem 2 Proximal Gradient for Lasso')
xlabel('iteration');
ylabel('f(w^k)-f(w*)')
```

Problem 3

a, $f(w) = \underbrace{\frac{1}{2}\|Xw - y\|_2^2}_{g(w)} + \underbrace{\lambda \|w_1\|}_{h(w)}$

$g(w) = \frac{1}{2}\|XW - y\|^2 = \frac{1}{2}(Xw-y)^T(Xw-y)$ where $s_i \in \partial |w_i + \delta e_i|$

$= \frac{1}{2} w^T X^T X w - w^T X^T y + \frac{1}{2} y^T y$

$g(w+\delta e_j) = \frac{1}{2}(w+\delta e_j)^T X^T X (w+\delta e_j) - (w+\delta e_j)^T X^T y + \frac{1}{2} y^T y$

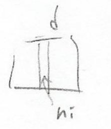
$= \frac{1}{2} w^T X^T X w + w^T X^T X \delta e_j + \frac{1}{2}\delta^2 \sum_{i=1}^n x_{ij}^2 - y^T X w - y^T X e_j \delta + \frac{1}{2} y^T y$

$\nabla_\delta g(w+\delta e_j) = \sum_{i=1}^n x_{ij}(x_i^T w) + \sum_{i=1}^n x_{ij}\delta - \sum_{i=1}^n x_{ij} y_i$

$\Rightarrow \delta^* = -\dfrac{\sum_{i=1}^n x_{ij}(x_i^T w - y_i)}{\sum_{i=1}^n x_{ij}^2}$ where j is the coordinate we want to deal (feature)
i is instance (data) in j's coordinate. $i \in 1 \sim n$

update rule:

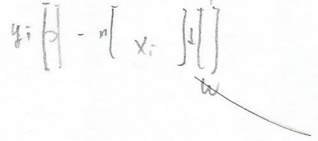
$$\boxed{\begin{aligned} \bar{w}_i &= w_i^k + \frac{-\sum_{i=1}^n x_{ij}(x_i^T w - y_i)}{\sum_{i=1}^n x_{ij}^2} \\ w_i^{k+1} &= S_{\lambda/(\sum_{i=1}^n x_{ij}^2)}(\bar{w}_i) = \begin{cases} \bar{w}_i - \lambda/(\sum_{i=1}^n x_{ij}^2) & \text{if } \bar{w}_i > \lambda/(\sum_{i=1}^n x_{ij}^2) \\ \bar{w}_i + \lambda/(\sum_{i=1}^n x_{ij}^2) & \text{if } \bar{w}_i < -\lambda/(\sum_{i=1}^n x_{ij}^2) \\ 0 & \text{if } |\bar{w}_i| < \lambda/(\sum_{i=1}^n x_{ij}^2) \end{cases} \end{aligned}}$$ ※



b, $\delta^* = -\dfrac{\sum_{i=1}^n x_{ij}(x_i^T w - y_i)}{\sum_{i=1}^n x_{ij}^2}$ Assume $X \in \mathbb{R}^{n\times d}$, each column of X has n_i nonzero element ($\sum_i n_i = nnz(X)$)

① Precompute $h_i := \sum_{i=1}^n X_{ij}^2 \longrightarrow O(nnz(X))$ operation

② Precompute $r_i := x_i^T w - y_i$ for all i (instance i from $1 \sim n$) $\to O(nnz(X))$ operation



③ For each coordinate update.

- compute $\delta^* = \dfrac{\sum_{i=1}^n x_{ij}(y_i - x_i^T w)}{\sum_{i=1}^n x_{ij}^2} = \dfrac{\sum_{i=1}^n r_i X_{ij}}{h_i} \to O(n_i)$ operation
- For all $i = 1 \cdots n$

let $\frac{\lambda}{\sum_{i=1}^n x_{ij}^2} = h$

```
if $\bar{w}_i > h$,    $r_i \leftarrow r_i + (\delta^* - h) * X_{ij}$
if $\bar{w}_i < -h$    $r_i \leftarrow r_i + (\delta^* + h) * X_{ij}$    } → $O(n_i)$ operation
if $|\bar{w}_i| < h$   $r_i = r_i$
```

$\Rightarrow$ for total n coordinate update : $\sum_{i=1}^n n_i = O(nnz(X))$

From ①, ② and ③, each outer iteration require $O(nnz(X))$ operation ※

Problem 3 Coordinate Descent for Lasso

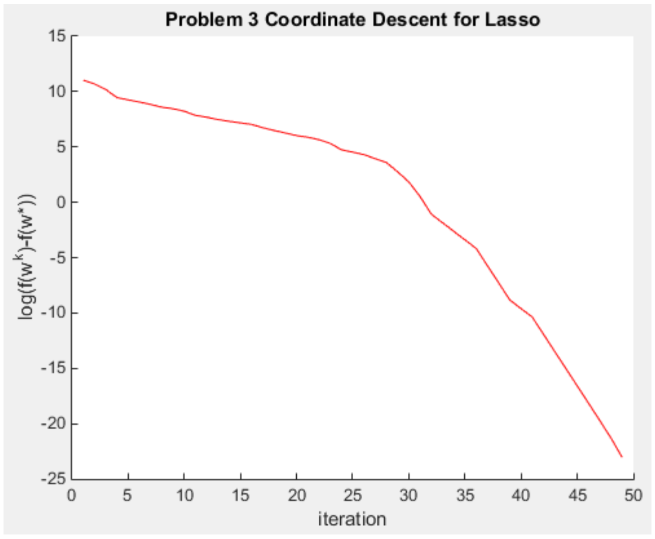
(a) On written paper

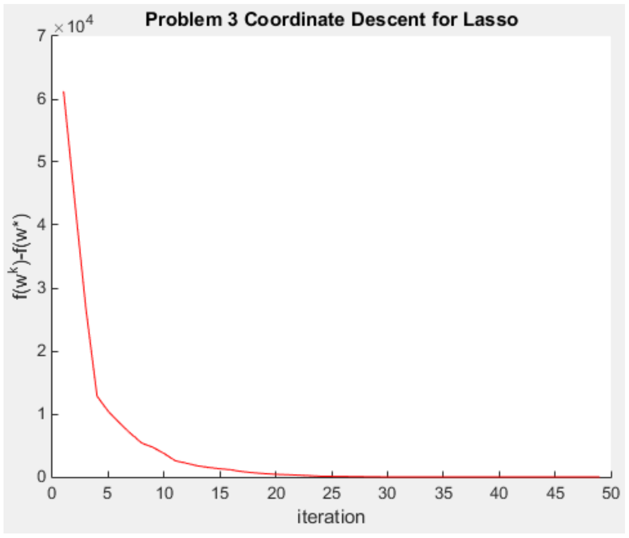
(b) On written paper

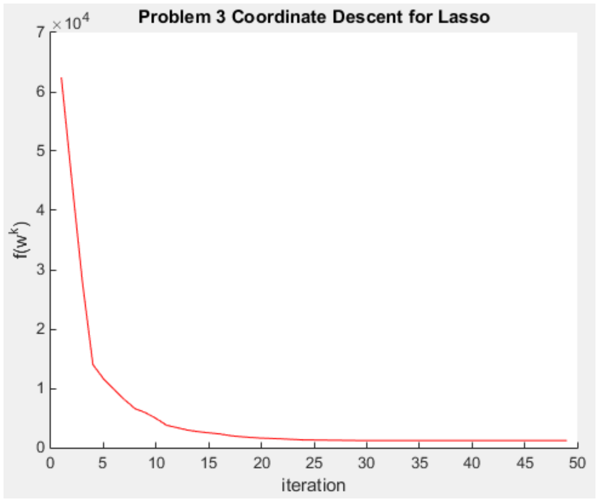
(c) $\boldsymbol{f}(\boldsymbol{w}^{\mathbf{20}}) = \mathbf{1.6080} \times \mathbf{10^3}$ Three graphs for reference.

Since simulation result may be different by time, I listed 5 simulation results for $\boldsymbol{f}(\boldsymbol{w}^{\mathbf{20}})$:

1.6882×10^3	1.3513×10^3	1.3786×10^3	1.4629×10^3	1.7559×10^3







```
wCoordi = zeros(dimnum,1);
iterNum = 50;
CostHisCoordi = zeros(1,iterNum);
h = zeros(1,dimnum);
for j=1:dimnum
    h(1,j) = sum(X(:,j).^2);
    h(1,j) = h(1,j) + 1e-50;
end
```

```
for k=1:iterNum
   p = randperm(dimnum);
   %precompute ri
   r = X * wCoordi -y;

   for s =1 : dimnum
      i=p(s);
      %calculate coordinate update rule
      delta = X(:,i).' * r;
      delta = -delta / h(1,i);
      wbar(i) = wCoordi(i) + delta;
      wCoordi(i) = sign(wbar(i)).*max(abs(wbar(i))-lamda / h(1,i),0);
      if wbar(i) > lamda / h(1,i)
         r = r + (delta - lamda / h(1,i)) * X(:,i);
      elseif wbar(i) < - lamda / h(1,i)
         r = r + (delta + lamda / h(1,i)) * X(:,i);
      end
   end
   CostHisCoordi(1,k) = 0.5*norm(X * wCoordi - y,2)^2 + lamda*norm(wCoordi,1);
end

errCoordi = zeros(1,iterNum - 1);
for i = 1 : iterNum - 1
   errCoordi(1,i) = CostHisCoordi(1,i);
end

figure
hold all
plot(errCoordi,'r')
title('Problem 3 Coordinate Descent for Lasso')
xlabel('iteration');
ylabel('f(w^k)')
```

Problem 4

a, For 3, : $\min_w \frac{1}{2}\|Xw - y\|_2^2 + \lambda\|w\|_1$

For 6, : $\min_{w_1, w_2} \frac{1}{2}\|Xw_1 - y\|_2^2 + \lambda\|w_2\|_1 + \frac{1}{2}\|w_1 - w_2\|^2$ s.t. $w_1 = w_2$

replace $w_2 = w_1$ $\Rightarrow$ $\min_{w_1} \frac{1}{2}\|Xw_1 - y\|_2^2 + \lambda\|w_1\|_1 + 0$

which equtal problem to 3,

$\therefore$ Solution to 3, is equal to solution to 6,

b,

$w_1^k = \arg\min_w L(w, w_2^{k-1}, u^{k-1})$

$= \arg\min_w \frac{1}{2}\|Xw - y\|_2^2 + u^T w = f(w)$

$\Rightarrow \nabla f(w) = X^T(Xw^* - y) + u = 0$

$\Rightarrow x^* = (X^TX)^{-1}(X^Ty - u)$

Time complexity : X^TX : $d[X^T]n[\overset{d}{X}] \Rightarrow O(nd^2)$

$(X^TX)^{-1}$: $O(d^3)$ which $150K \times 150K \times 150K \doteq 3 \times 10^{16}$ operation

therefore it is almost impossible to compute close form solution using a single computer.

c, $w_1^k = \arg\min_w L(w, w_2^{k-1}, u^{k-1})$

$= \arg\min_w \frac{1}{2}\|Xw - y\|_2^2 + u^T w + \frac{1}{2}\|w - w_2\|_2^2$

$\Rightarrow X^T(Xw^* - y) + u + (w^* - w_2) = 0$

$\Rightarrow \underbrace{(X^TX + I)}_{A} w^* = \underbrace{X^Ty - u + w_2}_{b}$

rest show in code

Problem 4 ADMM for Lasso

(a) On written paper

(b) On written paper

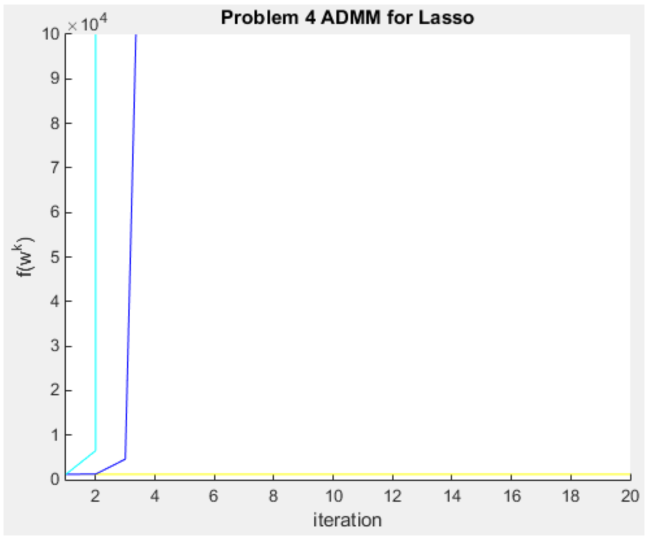
(c)

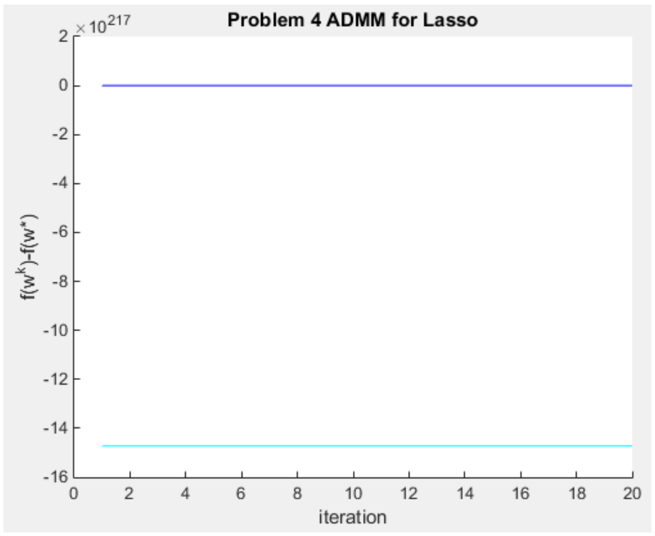
```
%(c) Calculate Ax=b
wADMM1 = zeros(dimnum,1);
wADMM2 = zeros(dimnum,1);
u = zeros(dimnum,1);
b = X.' * y - u + wADMM2;
wADMM1 = fast_conjugate_gradient_solve(X, b);
function d = fast_conjugate_gradient_solve(datax, b)
%Using Conjugate gradient to solve Aw=b
%w : input weight(x) (No need wait to kill)
%d: optimal solution that we want to compute
%b: input that need to calculate before input
%datax: to faster compute A  = X^T * X + I
sizex = size(datax,2);
d = zeros(sizex,1);
r = b; %r0 = b - X*d but initial d = 0 so r = b;
p = r;
k=0;
while true
    FastComp1 = datax * p;
    FastComp2 = datax.' * FastComp1 + p; % result of A * p
    rk_square = r.' * r;
    alpha_k = rk_square / (p.' * FastComp2);
    d = d + alpha_k * p;
    r = r - alpha_k * FastComp2;
    if norm(r,2)/norm(b,2) <= 1e-3
        break
    end
    belta = (r.' * r) / rk_square;
    p = r + belta * p;
    k = k+1;
end
end
```

(d)

For step size = 0.01, 0.1 and 1, they converge pretty fast after several iterations going to minimizer which f(w^{20})=1.1964X10^{3}. In which step size is 1, it converges to this point after 6 iterations. For step size = 10 and 100, they are explosive large.

Step size	0.01	0.1	1	10	100
$\boldsymbol{f(w^{20})}$	1.1959×10^3	1.1962×10^3	1.1964×10^3	4.9559×10^{40}	1.9071×10^{80}





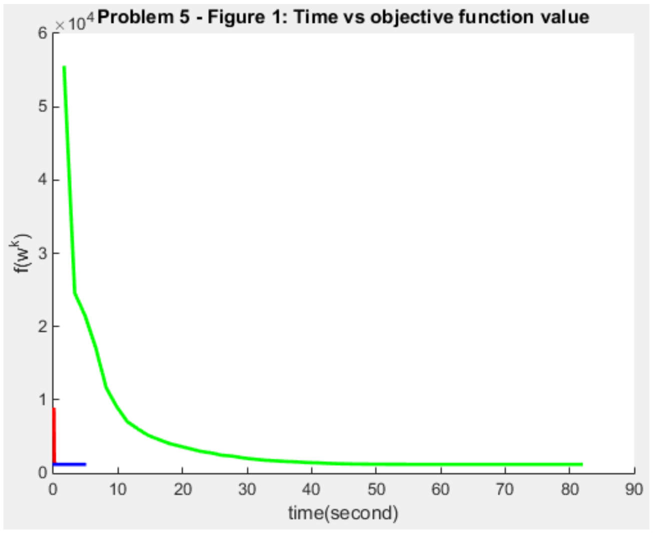
```
Red: 0.01
Green: 0.1
Black: 1
Blue: 10
Cyan: 100

%(d) ADMM for Lasso
eta = [0.01,0.1,1,10,100];
iter = 50;
CostHistADMM = zeros(5,iter);
for etanum=1:5
    wADMM1 = zeros(dimnum,1);
    wADMM2 = zeros(dimnum,1);
    u = zeros(dimnum,1);
    for k =1: iter
        %w1^k update
        b = X.' * y - u + wADMM2; %u^k-1 and w2^k-1
        wADMM1_k = fast_conjugate_gradient_solve(X, b); %record new w1^k
        %w2^k update
        inner = wADMM1 + u;
        wADMM2 = sign(inner).*max(abs(inner)-lamda,0);
        %u^k update
        u = u + eta(etanum) * (wADMM1_k - wADMM2);
        wADMM1 = wADMM1_k;
        CostHistADMM(etanum, k) = 0.5*norm(X * wADMM1 - y,2)^2 + lamda*norm(wADMM2,1)
+ 0.5*norm(wADMM1-wADMM2,2)^2;
    end
end

errADMM = zeros(5,20);
for j=1:5
    for i = 1 : 20
        errADMM(j,i) = CostHistADMM(j,i);
    end
end
figure
```

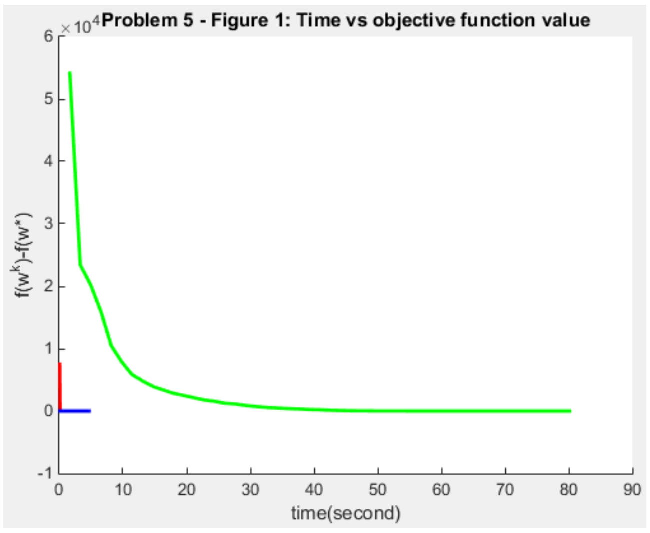
```
hold all
axis([1 20 0 1e5]);
plot(errADMM(1,:),'r')
plot(errADMM(2,:),'g')
plot(errADMM(3,:),'y')
plot(errADMM(4,:),'b')
plot(errADMM(5,:),'c')

title('Problem 4 ADMM for Lasso')
xlabel('iteration');
ylabel('f(w^k)')
```

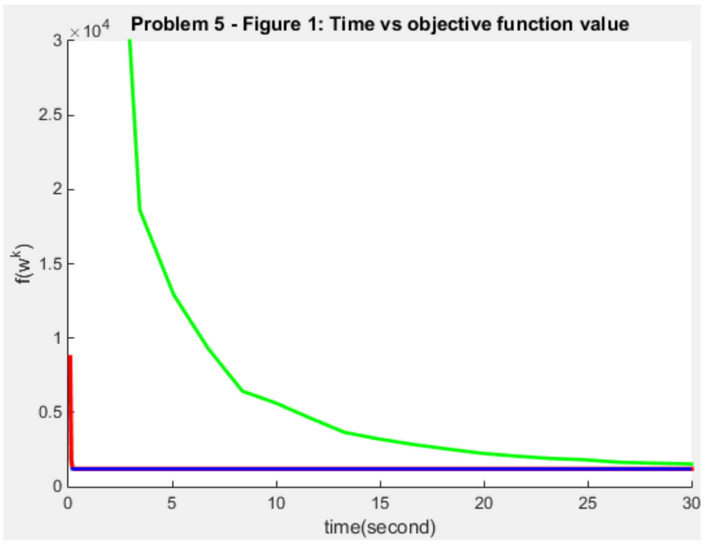
Problem 5 Comparison of proximal gradient, ADMM and coordinate descent

1. Time vs objective function value for E2006 dataset
   Run three algorithms in **50** iterations. I calculated time after initializing parameters for each algorithm. And set a time stamp on the end of each iteration (using `toc` command in matlab).
   Step Size: Proximal gradient is set 2^{-18}.
   ADMM's step size is set 1.

```
Red: Proximal
Green: Coordinate
Blue: ADMM
```



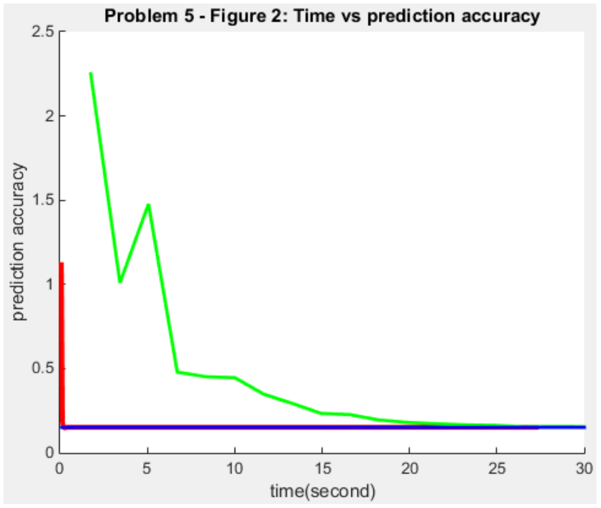




In this graph, I use <u>different iteration time</u> on different algorithm. It will show a better result on x axis. Below is each algorithm that executed time arrived 30 seconds:

Proximal gradient **496** iterations
Coordinate descent **19** iterations
ADMM **328** iterations

2. Time vs prediction accuracy for E2006 dataset
   The mean square error will converge to **0.1519** among all algorithms when iteration times arrive 30 seconds.



<u>**E2006 dataset conclusion**</u>:
First, step size of proximal gradient and ADMM are $\mathbf{2^{-18}}$ and **0.1** separately.
Second, all of three algorithms converge to optimal solution after 30 seconds.
Third, proximal gradient computes fast on each iteration but it needs more iterations to converge and

its objective function value is large at beginning but converge pretty fast to optimal solution. Coordinate descent takes longer time on each iteration and converge slow to optimal solution. However, its prediction accuracy is unstable on the way to optimal solution. ADMM is also fast on each iteration and general get good predict result. In E2006 dataset, **ADMM is a better algorithm to apply**.

3. Time vs objective function value for real-sim dataset

   Step size:

   Proximal gradient is set $\mathbf{2^{-14}}$.

   (It will converge to higher objective function value if we do not have enough step size.)

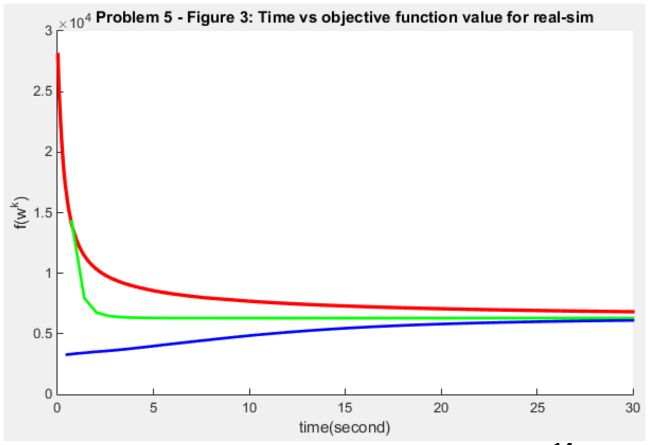
   ADMM's step size is set **0.1**. (It diverges when step size is 1.)

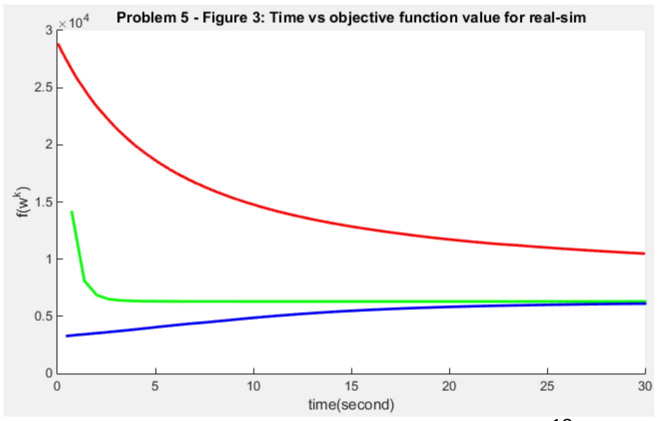
   Iteration time:

   Proximal gradient **2000** iterations

   Coordinate descent **50** iterations

   ADMM **64** iterations





Left hand side is Proximal gradient is set $\mathbf{2^{-14}}$. And right hand side is Proximal gradient is set 2^{-18} which is not suitable for this dataset.

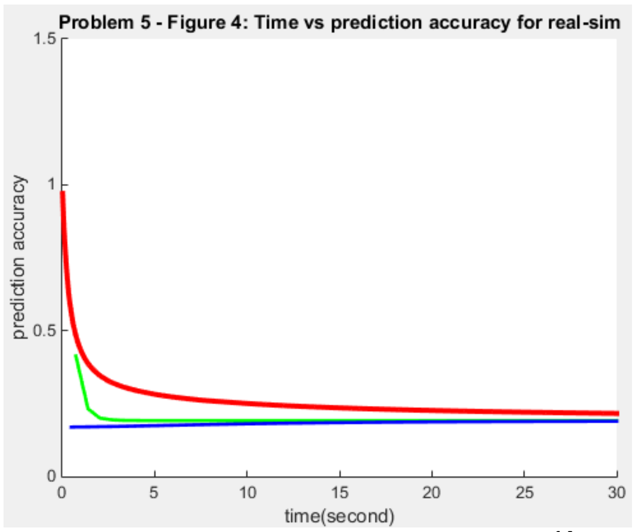
4. Time vs prediction accuracy for real-sim dataset

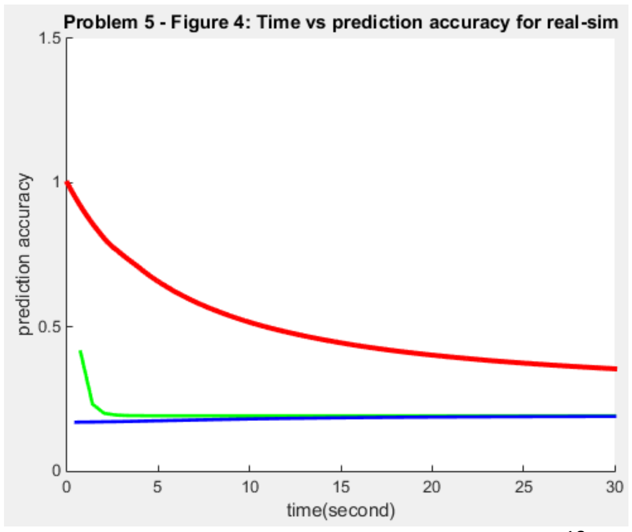
   The mean square error when iteration times arrive 30 seconds:

   Proximal gradient: 0.2144 (If step size equal to 2^{-18}, accuracy is 0.3467 shown left right)

   Coordinate descent: 0.1917

   ADMM: 0.1897





Left hand side is Proximal gradient is set $\mathbf{2^{-14}}$. And right hand side is Proximal gradient is set 2^{-18}.

Real-sim dataset conclusion:

First, step size of proximal gradient and ADMM is different from E2006 dataset which are 2^{-14} and 1 separately.

Second, proximal gradient computes fast on each iteration but it needs more iterations to converge and its predict error is slightly higher than other two algorithms. Coordinate descent takes longer time on each iteration but converge fast to optimal solution. ADMM is also fast on each iteration and general get good predict result, but in this dataset it converges relatively slower than coordinate descent. In real-sim dataset, **coordinate descent is a better algorithm to apply**.