Computer Project

Student

January 16, 2022

1 a.Gradient Method

1.1 Analytical expression for the gradient

For

$$f(\mathbf{x}) = -\sum_{i=1}^{m} \ln(1 - \mathbf{a}_i^T \mathbf{x}) - \sum_{i=1}^{n} \ln(1 - {x_i}^2)$$

the gradient of $f(\mathbf{x})$ is $\nabla f(\mathbf{x}) = \frac{\partial f(\mathbf{x})}{\partial x_i}\mathbf{e}_i$

For the k-th component of the gradient vector ,

$$\frac{\partial f(\mathbf{x})}{\partial x_k}\mathbf{e}_k = \left(\sum_{i=1}^{m} \frac{(\mathbf{a}_i)_k}{(1 - \mathbf{a}_i^T \mathbf{x})} + \frac{2x_k}{(1 - {x_k}^2)} \right) \mathbf{e}_k$$

where $(\mathbf{a}_i)_k$ denotes the k-th component of the vector $\mathbf{a}_i$, which is a scaler.

Tightly expressed, we have:

$$\nabla f(\mathbf{x}) = \sum_{i=1}^{m} \frac{\mathbf{a}_i}{(1 - \mathbf{a}_i^T \mathbf{x})} + 2(\mathbf{I} - \mathrm{diag}({x_k}^2))^{-1}\mathbf{x}$$

where k is the component parameter of the diagonal matrix, shown as:

$$\mathrm{diag}({x_k}^2) = \begin{pmatrix} {x_1}^2 & & & \\ & {x_2}^2 & & \\ & & \ddots & \\ & & & {x_n}^2 \end{pmatrix}$$

1.2 Solve the problem

For the Gradient Descent Method with backtracking line search, where $0 < \alpha < 0.5$ and $0 < \beta < 1$:

The iterative equations are :

$$\begin{cases} \mathbf{x}^0 = \mathbf{0} \\ \Delta \mathbf{x} = -\nabla f(\mathbf{x})\big|_{\mathbf{x}^i} \\ t_i = \text{find\_backtracking\_step}(\mathbf{x}^i) \\ \mathbf{x}^{i+1} = \mathbf{x}^i + t_i \Delta \mathbf{x} \end{cases}$$

The terminated condition is :

$$||\nabla f(\mathbf{x})||_2 \leq \eta \quad \text{at iterative step } i$$

The result then is

$$f(\mathbf{x}) = f(\mathbf{x}^i)$$

The methond of finding backtracking step given $\mathbf{x}^i$ are:

$$\begin{cases} t_0 = 1 \\ t_{k+1} = \beta t_k \quad \text{iff. } f(\mathbf{x}^i + t_k \Delta\mathbf{x}) > f(\mathbf{x^i}) + \alpha t_k \nabla f(\mathbf{x})^T \Delta\mathbf{x} \end{cases}$$

The terminated condition is :

$$f(\mathbf{x} + t_k \Delta\mathbf{x}) \leq f(\mathbf{x}) + \alpha t_k \nabla f(\mathbf{x})^T \Delta\mathbf{x} \quad \text{at step } k$$

And the return value of find_backtracking_step($\mathbf{x}_i$) is t_k.

1.2.1 code

```
%Gradient Method

% initialize
clear all;
alpha = 0.1;
beta = 0.6;
eta = 1e-5;


% generate the random instance

global A;
%load('A_200_100.mat');
load('A_500_400.mat')
[m ,n] = size(A);

value = [];
step = [];
%main iteration


% at step 0
x = zeros(n,1);
grad = A'*(1./(1 - A*x)) - 1./(1+x) + 1./(1-x);


while norm(grad,2) > eta

value = [value, func(x)];
△_x = -grad;
t = 1;

% constrain the x in dom(x) by changing t
while ((max(A*(x+t*△_x)) ≥ 1) || (max(abs(x+t*△_x)) ≥ 1))
```

```
t = t * beta;
end

% backtracking line search
while (func(x+t*Δ_x) - func(x) > alpha * t * grad' * Δ_x)
t = t * beta;
end
step = [step, t];
% update x by:
x = x + t * Δ_x;
% update new gradient at x by:
grad = A'*(1./(1 - A*x)) - 1./(1+x) + 1./(1-x);
end

%dump result
opt = min(value);

figure(1)
subplot(1,3,1);
plot([0:(length(value)-2)], value(1:length(value)-1), '-');
yl = '$f(\textbf{x}^k)$';
xlabel('iterative step');
ylim([min(value) max(value)]);
ylabel(yl,'Interpreter','latex');
title('value - iterative step');

hold on;

subplot(1,3,2);
semilogy([0:(length(value)-2)], value(1:length(value)-1)-opt, '-');
xlabel('iterative step');
yl2 = '$f(\textbf{x}^k)-p^*$';
ylabel(yl2,'Interpreter','latex');
title('value between opt - iterative step');

hold on;

subplot(1,3,3);
scatter([1:length(step)], step,'filled','black');
xlabel('iterative step');
ylabel('$t^k$','Interpreter','latex');
title('step size - iterative step')
hold on;

function res = func(x)
global A;
res = -sum(log(1-A*x)) - sum(log(1+x)) - sum(log(1-x));
end

%
%
```

1.3 Result

In this case and the following experimemt, we use totally two normal instance (not testing the parameter of BLS or the convergency). The small instance is with $m = 200, n = 100$ and the large instance is with $m = 500, n = 400$, the vector set of $\mathbf{a}_i$ are equipped as matrix, and each element are generated as random number.

Both case have the BLS parametera at $\alpha = 0.1, \beta = 0.6$.

The following result shows the normal testing case with three subfigure. These are function value , value towards optimal value and time step size to iterative step number. The second figure value towards optimal value to iterative step number are plot in y - logarithmic axis. Because the function value have negative true value, we can not plot the first figure in y - logarithmic axis.

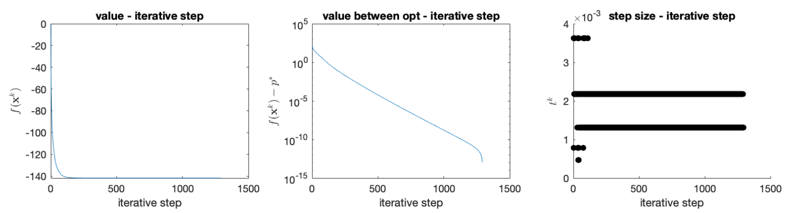

Figure 1: Instance 1, small instance

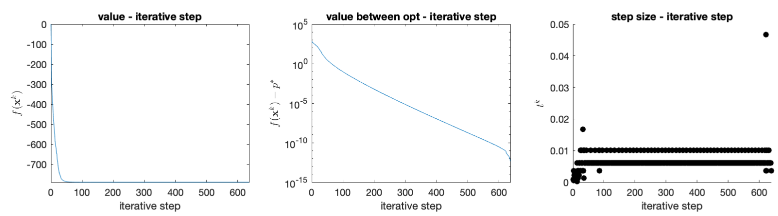

Figure 2: Instance 2, large instance

1.3.1 BLS parameter

The backtracking line search have two parameter α, β. The BLS is a algorithm finding the maximize t with a constraint which ensure the next step closer enough towards the optimal without marching to large step in a specific step direction.

The following results show the parameter impact.

We use blue line as base line, with $\alpha = 0.45, \beta = 0.85$.

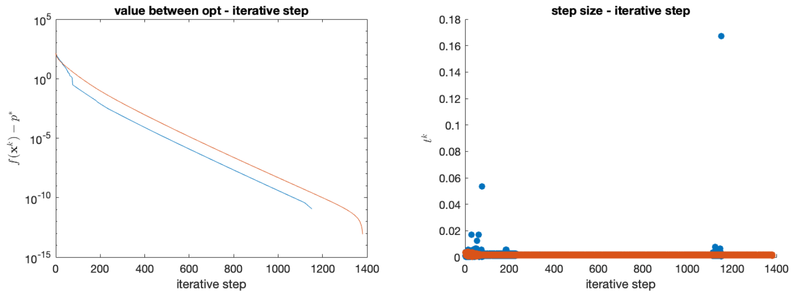

Figure 3: tryout 1, various α

For the orange case, the $\alpha = 0.1$. The smaller α makes the step along a step direction closer to the coutor line, and this leads small step size. So the iterative step will grow.

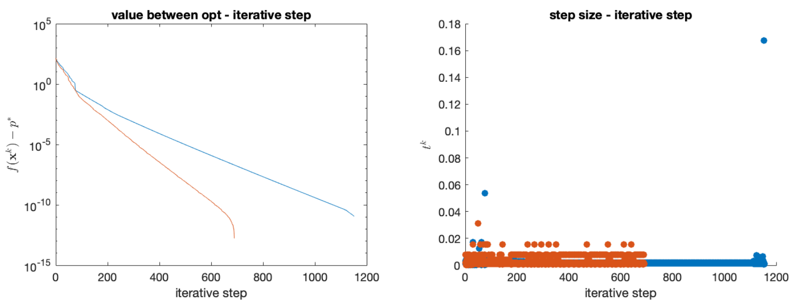

Figure 4: tryout 2, various β

For the orange case, the $\beta = 0.5$. The smaller β have faster run time, because it iterates the t with large damping. And because the t damped fast, which means a great progress, the stop condition of BLS maybe reach with rough accuracy, which makes step size fall into a larger value(because when reach the condition, the t will not update and stay in the last t) than its exact value. So the total iterative step become smaller for a specific error.

2 b.Damped Newton's Method

2.1 Analytical expression for the Hessian

The Hessian of this problem is

$$\mathbf{H} := \nabla^2 f(\mathbf{x}) = \sum_{i=1}^{m} \frac{\mathbf{a}_i \mathbf{a}_i^T}{(1 - \mathbf{a}_i^T \mathbf{x})} + \text{diag}\left(\frac{2(1 + {x_k}^2)}{(1 - {x_k}^2)^2} \right)$$

note that $\mathbf{a}_i \mathbf{a}_i^T$ is outer product and returns a matrix.

2.2 Solve the problem

The only difference towards case **1 a** is that $\Delta \mathbf{x}$ is given as

$$\Delta \mathbf{x}_{\text{dnt}} = -\mathbf{H}^{-1} \nabla f(\mathbf{x})$$

And the stopping criterion is $\frac{1}{2}\lambda(\mathbf{x})^2 \leq \epsilon$, where

$$\lambda(\mathbf{x})^2 := \nabla f(\mathbf{x})^T \nabla^2 f(\mathbf{x})^{-1} \nabla f(\mathbf{x})$$

2.2.1 code

```
% damped Newton Method

clear all;
alpha = 0.1;
beta = 0.6;
epsilon = 1e-8;


% generate the random instance
global A;
%load('A_200_100.mat');
load('A_500_400.mat');
[m ,n] = size(A);
value = [];
step = [];
%main iteration


% at step 0
x = zeros(n,1);
grad = A'*(1./(1 - A*x)) - 1./(1+x) + 1./(1-x);

hessian = A'*diag((1./(1-A*x)).^2)*A + diag(1./(1+x).^2 + ...
    1./(1-x).^2);

while 0.5 * lambda(hessian, grad) > epsilon

value = [value, func(x)];
Δ_x = -hessian^(-1) * grad;
t = 1;

% constrain the x in dom(x) by changing t
while ((max(A*(x+t*Δ_x)) ≥ 1) || (max(abs(x+t*Δ_x)) ≥ 1))
t = t * beta;
end

% backtracking line search
while (func(x+t*Δ_x) - func(x) > alpha * t * grad' * Δ_x)
t = t * beta;
end
step = [step, t];
% update x by:
```

```
x = x + t * Δ_x;
% update new gradient and hessian at x by:
grad = A'*(1./(1 - A*x)) - 1./(1+x) + 1./(1-x);
hessian = A'*diag((1./(1-A*x)).^2)*A + diag(1./(1+x).^2 + ...
    1./(1-x).^2);

end

%dump result
opt = min(value);

figure(1)
subplot(1,3,1);
plot([0:(length(value)-2)], value(1:length(value)-1), '-');
yl = '$f(\textbf{x}^k)$';
xlabel('iterative step');
ylim([min(value) max(value)]);
ylabel(yl,'Interpreter','latex');
title('value  - iterative step');
hold on;

subplot(1,3,2);
semilogy([0:(length(value)-2)], value(1:length(value)-1)-opt, '-');
xlabel('iterative step');
yl2 = '$f(\textbf{x}^k)-p^*$';
ylabel(yl2,'Interpreter','latex');
title('value between opt - iterative step');
hold on;

subplot(1,3,3);
scatter([1:length(step)], step,'filled','black');
xlabel('iterative step');
ylabel('$t^k$','Interpreter','latex');
title('step size - iterative step');
hold on;



function res = func(x)
global A;
res = -sum(log(1-A*x)) - sum(log(1+x)) - sum(log(1-x));
end

function res = lambda(hess, grad)
res = grad' * hess^(-1) * grad;
end

%
%
```

2.3 Result

The following figure show the result of same two instances in **case 1a**, with $\eta = 1e-8$.There are only fews iterative step remains to reach convergency.

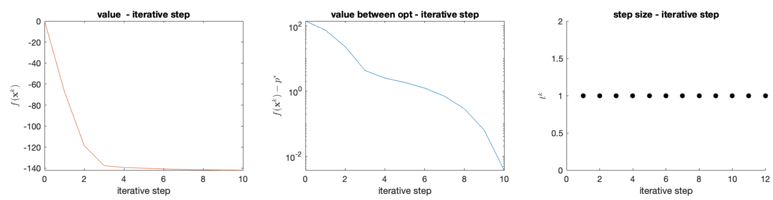

Figure 5: Instance 1, small instance

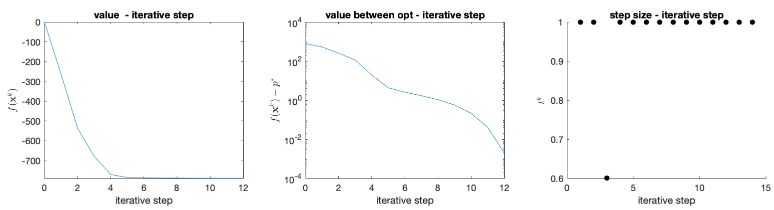

Figure 6: Instance 2, large instance

2.3.1 Convergency

The choose of η will depend convergency and iterative step.

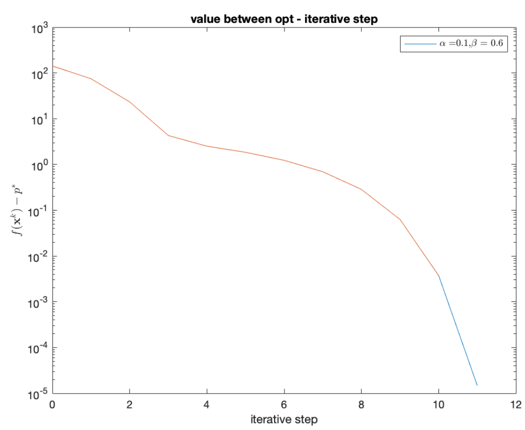

Figure 7: Different η

We choose $\eta = 1e-5$ and $\eta = 1e-15$, the result shows that few extra step are needed to reach higher error magnitude, because damped Newton Method have quadratic convergency.

3 c.Approximated Newton's Method

3.1 Re-using the Hessian

In this case, the iterative equations and terminated condition are the same as case **2 b**. The only difference is that the Hessian are updated every N step, with $\Delta\mathbf{x} = -\mathbf{H}^{-1}\nabla f(\mathbf{x})$, where $\mathbf{H}$ is the last Hessian evaluted.

3.2 code

```
%approximated damped Newton Method by reduce hessian calc
clear all;
% initialize
alpha = 0.1;
beta = 0.6;
epsilon = 1e-8;


% generate the random instance
global A;
%load('A_200_100.mat');
load('A_500_400.mat');
[m,n] = size(A);
value = {};
step = {};
%main iteration

index = 1;
%do three time
for N = [1,10,40]
% at step 0

value{index} = [];
step{index} = [];
inner_step = 0;
x = zeros(n,1);
grad = A'*(1./(1 - A*x)) - 1./(1+x) + 1./(1-x);
hessian = A'*diag((1./(1-A*x)).^2)*A + diag(1./(1+x).^2 + ...
    1./(1-x).^2);
hessian_i = hessian ^(-1);
while 0.5*lambda(hessian_i, grad) > epsilon


value{index} = [value{index}, func(x)];
Δ_x = -hessian_i * grad;
t = 1;

% constrain the x in dom(x) by changing t
while ((max(A*(x+t*Δ_x)) ≥ 1) || (max(abs(x+t*Δ_x)) ≥ 1))
t = t * beta;
end

% backtracking line search
while (func(x+t*Δ_x) - func(x) > alpha * t * grad' * Δ_x)
```

```
t = t * beta;
end
step{index} = [step{index}, t];
% update x by:
x = x + t * ∆_x;
% update new gradient at x by:
grad = A'*(1./(1 - A*x)) - 1./(1+x) + 1./(1-x);

% update new hessian and inverse at x by every N step:
if (mod(inner_step ,N) == 0)
hessian = A'*diag((1./(1-A*x)).^2)*A + diag(1./(1+x).^2 + ...
    1./(1-x).^2);
hessian_i = hessian ^ (-1);
end
inner_step = inner_step + 1;
end
index = index + 1;

end

%dump result
opt = min(value{3});


figure(1)
N = [1,10,40];
for i = 1:3
subplot(3,3,i);
plot([0:(length(value{i})-2)], value{i}(1:length(value{i})-1), '-');
yl = '$f(\textbf{x}^k)$';
xlabel('iterative step');
ylim([min(value{i}) max(value{i})]);
ylabel(yl,'Interpreter','latex');
title(['N=' num2str(N(i))]);
hold on;
end

for i = 1:3
subplot(3,3,i+3)
semilogy([0:(length(value{i})-2)], ...
    value{i}(1:length(value{i})-1)-opt, '-');
xlabel('iterative step');
yl2 = '$f(\textbf{x}^k)-p^*$';
ylabel(yl2,'Interpreter','latex');
title(['N=' num2str(N(i))]);
hold on;
end

for i = 1:3
subplot(3,3,i+6)
scatter([1:length(step{i})], step{i},'filled','black');
xlabel('iterative step');
ylabel('$t^k$','Interpreter','latex');
title(['N=' num2str(N(i))]);
hold on;
end
```

```
function res = func(x)
global A;
res = -sum(log(1-A*x)) - sum(log(1+x)) - sum(log(1-x));
end

function res = lambda(hess_i, grad)
res = grad' * hess_i * grad;
end

%
%
```

3.2.1 Result

This figures shows the result using different N as interval to update the hessian and its inverse.The are taken $1, 10, 40$. Both small and large instance are used. When $N = 1$, this case is the same as **case 2b** , the damped Newton method.

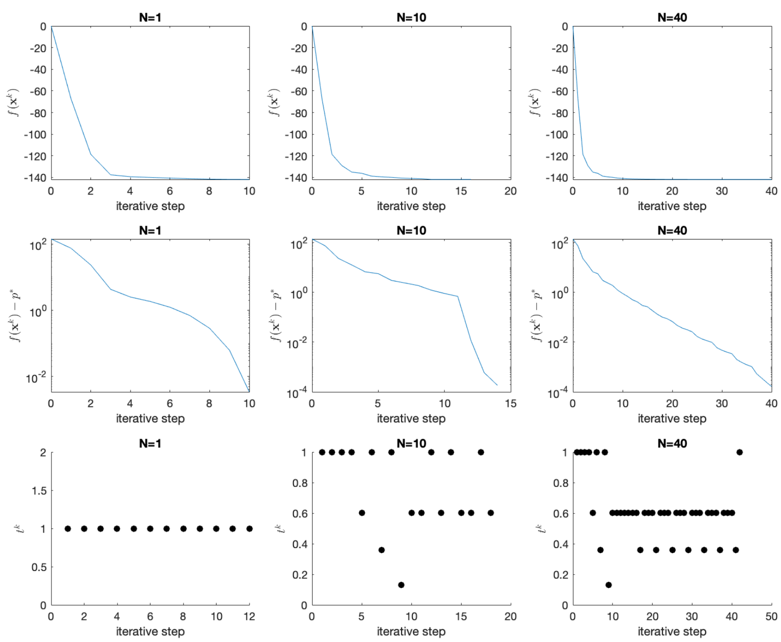

Figure 8: Instance 1, small instance

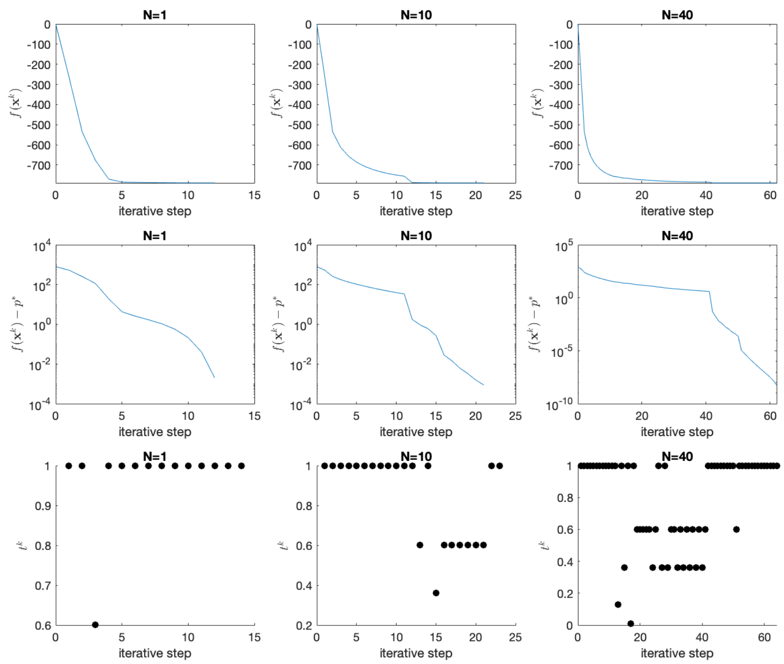

Figure 9: Instance 2, large instance

From the result can we know that, the large interval remains long iterative step for a specific error limit.

3.3 Diagonal approximation

In this case, the iterative equations and terminated condition are the same as case **2 b**. The only difference is that the Hessian $\mathbf{H}$ is approximated as its diagonal. That is

$$\tilde{\mathbf{H}} = \operatorname{diag}\left(\sum_{i=1}^{m} \frac{(\mathbf{a}_i)_k{}^2}{(1-\mathbf{a}_i^T\mathbf{x})} + \left(\frac{2(1+x_k{}^2)}{(1-x_k{}^2)^2}\right)\right)$$

k is the component parameter of diagonal matrix.

3.3.1 code

```
%approximated damped Newton Method by reduce hessian to its ...
    diagonal .

```

```
% initialize
alpha = 0.1;
beta = 0.6;
epsilon = 1e-8;


% generate the random instance
global A;
%load('A_200_100.mat');
load('A_500_400.mat');
[m ,n] = size(A);
value = [];
step = [];

%main iteration


% at step 0
x = zeros(n,1);
grad = A'*(1./(1 - A*x)) - 1./(1+x) + 1./(1-x);
hessian = A'*diag((1./(1-A*x)).^2)*A + diag(1./(1+x).^2 + ...
    1./(1-x).^2);
hd = diag(diag(hessian));
while  0.5 * lambda(hd, grad) > epsilon

value = [value, func(x)];
Δ_x = -hd^(-1) * grad;
t = 1;

% constrain the x in dom(x) by changing t
while ((max(A*(x+t*Δ_x)) ≥ 1) || (max(abs(x+t*Δ_x)) ≥ 1))
t = t * beta;
end

% backtracking line search
while (func(x+t*Δ_x) - func(x) > alpha * t * grad' * Δ_x)
t = t * beta;
end
step = [step, t];
% update x by:
x = x + t * Δ_x;
% update new gradient and hessian at x by:
grad = A'*(1./(1 - A*x)) - 1./(1+x) + 1./(1-x);
hessian = A'*diag((1./(1-A*x)).^2)*A + diag(1./(1+x).^2 + ...
    1./(1-x).^2);
hd = diag(diag(hessian));
end

%dump result
opt = min(value);


figure(1)
subplot(1,3,1);
plot([0:(length(value)-2)], value(1:length(value)-1), '-');
yl = '$f(\textbf{x}^k)$';
xlabel('iterative step');
```

```
ylim([min(value) max(value)]);
ylabel(yl,'Interpreter','latex');
title('value  - iterative step');
hold on;

subplot(1,3,2);
semilogy([0:(length(value)-2)], value(1:length(value)-1)-opt, '-');
xlabel('iterative step');
yl2 = '$f(\textbf{x}^k)-p^*$';
ylabel(yl2,'Interpreter','latex');
title('value between opt - iterative step');
hold on;

subplot(1,3,3);
scatter([1:length(step)], step,'filled','black');
xlabel('iterative step');
ylabel('$t^k$','Interpreter','latex');
title('step size - iterative step');
hold on;

function res = func(x)
global A;
res = -sum(log(1-A*x)) - sum(log(1+x)) - sum(log(1-x));
end


function res = lambda(hess, grad)
res = grad' * hess^(-1) * grad;
end
```

3.4 Result

This figures show the result of instance 1 the same as **case a**.

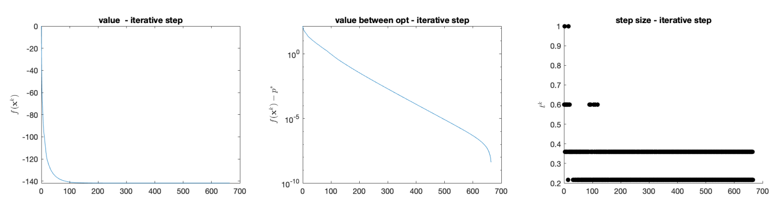

Figure 10: Instance 1,small instance

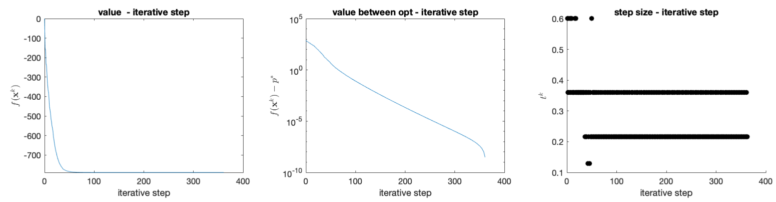

Figure 11: Instance 2,large instance

From the resule can we know that , the iterative step remains large compared with damped Newton Method.