

# 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 scalar.

Tightly expressed, we have:

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

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

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

### 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 :

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

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 method 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

```

1  %Gradient Method
2
3  % initialize
4  clear all;
5  alpha = 0.1;
6  beta = 0.6;
7  eta = 1e-5;
8
9
10 % generate the random instance
11
12 global A;
13 %load('A_200_100.mat');
14 load('A_500_400.mat');
15 [m,n] = size(A);
16
17 value = [];
18 step = [];
19 %main iteration
20
21
22 % at step 0
23 x = zeros(n,1);
24 grad = A*(1./(1 - A*x)) - 1./(1+x) + 1./(1-x);
25
26
27 while norm(grad,2) > eta
28
29     value = [value, func(x)];
30     Δ_x = -grad;
31     t = 1;
32
33     % constrain the x in dom(x) by changing t
34     while ((max(A*(x+t*Δ_x)) ≥ 1) || (max(abs(x+t*Δ_x)) ≥ 1))

```

```

35 t = t * beta;
36 end
37
38 % backtracking line search
39 while (func(x+t*Δ_x) - func(x) > alpha * t * grad' * Δ_x)
40 t = t * beta;
41 end
42 step = [step, t];
43 % update x by:
44 x = x + t * Δ_x;
45 % update new gradient at x by:
46 grad = A'*(1./(1 - A*x)) - 1./(1+x) + 1./(1-x);
47 end
48
49 %dump result
50 opt = min(value);
51
52 figure(1)
53 subplot(1,3,1);
54 plot([0:(length(value)-2)], value(1:length(value)-1), '-');
55 yl = '$f(\textbf{x}^k)$';
56 xlabel('iterative step');
57 ylim([min(value) max(value)]);
58 ylabel(yl, 'Interpreter', 'latex');
59 title('value - iterative step');
60
61 hold on;
62
63 subplot(1,3,2);
64 semilogy([0:(length(value)-2)], value(1:length(value)-1)-opt, '-');
65 xlabel('iterative step');
66 yl2 = '$f(\textbf{x}^k)-p^*$';
67 ylabel(yl2, 'Interpreter', 'latex');
68 title('value between opt - iterative step');
69
70 hold on;
71
72 subplot(1,3,3);
73 scatter([1:length(step)], step, 'filled', 'black');
74 xlabel('iterative step');
75 ylabel('$t^k$, 'Interpreter', 'latex');
76 title('step size - iterative step');
77 hold on;
78
79 function res = func(x)
80 global A;
81 res = -sum(log(1-A*x)) - sum(log(1+x)) - sum(log(1-x));
82 end
83
84 %
85 %

```

### 1.3 Result

In this case and the following experiment, 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.*

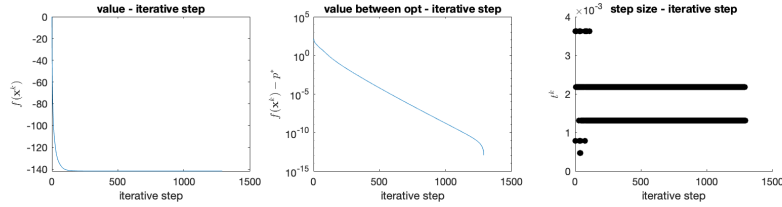


Figure 1: Instance 1, small instance

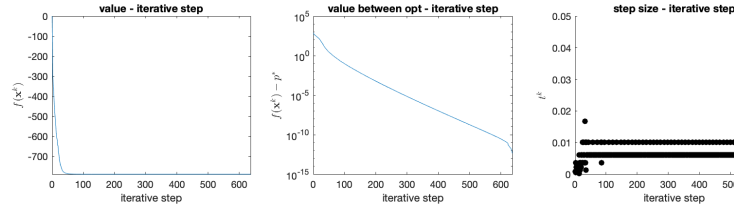


Figure 2: Instance 2, large instance

### 1.3.1 BLS parameter

The backtracking line search have two parameter  $\alpha, \beta$ . 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  $\alpha = 0.45, \beta = 0.85$ .

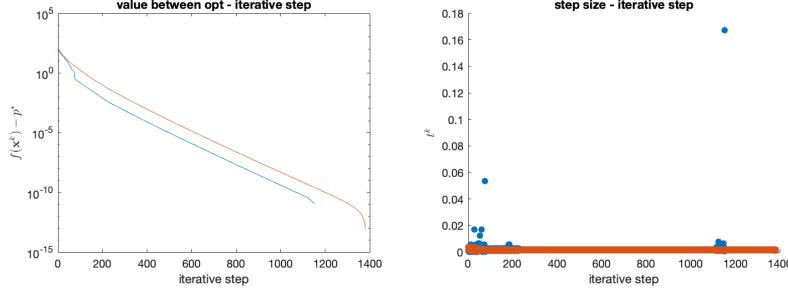


Figure 3: tryout 1, various  $\alpha$

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

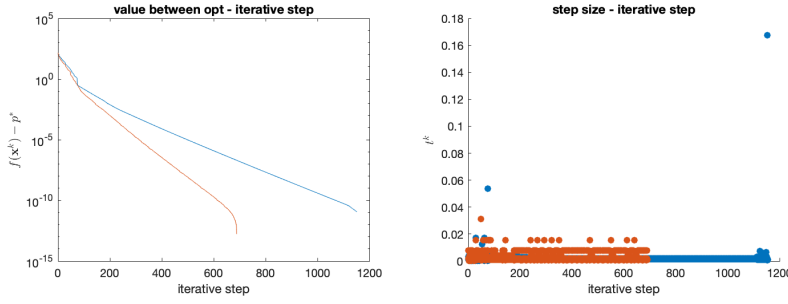


Figure 4: tryout 2, various  $\beta$

For the orange case, the  $\beta = 0.5$ . The smaller  $\beta$  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

```
1  % damped Newton Method
2
3  clear all;
4  alpha = 0.1;
5  beta = 0.6;
6  epsilon = 1e-8;
7
8
9  % generate the random instance
10 global A;
11 %load('A_200_100.mat');
12 load('A_500_400.mat');
13 [m,n] = size(A);
14 value = [];
15 step = [];
16 %main iteration
17
18
19 % at step 0
20 x = zeros(n,1);
21 grad = A'*(1./(1 - A*x)) - 1./(1+x) + 1./(1-x);
22
23 hessian = A'*diag((1./(1-A*x)).^2)*A + diag(1./(1+x).^2 + ...
24         1./(1-x).^2);
25
26 while 0.5 * lambda(hessian, grad) > epsilon
27     value = [value, func(x)];
28     Δ_x = -hessian^(-1) * grad;
29     t = 1;
30
31     % constrain the x in dom(x) by changing t
32     while ((max(A*(x+t*Δ_x)) ≥ 1) || (max(abs(x+t*Δ_x)) ≥ 1))
33         t = t * beta;
34     end
35
36     % backtracking line search
37     while (func(x+t*Δ_x) - func(x) > alpha * t * grad' * Δ_x)
38         t = t * beta;
39     end
40     step = [step, t];
41     % update x by:
```

```

42 x = x + t * Δ_x;
43 % update new gradient and hessian at x by:
44 grad = A'*(1./(1 - A*x)) - 1./(1+x) + 1./(1-x);
45 hessian = A'*diag((1./(1-A*x)).^2)*A + diag(1./(1+x).^2 + ...
    1./(1-x).^2);
46
47 end
48
49 %dump result
50 opt = min(value);
51
52 figure(1)
53 subplot(1,3,1);
54 plot([0:(length(value)-2)], value(1:length(value)-1), '-');
55 yl = '$f(\textbf{x}^k)$';
56 xlabel('iterative step');
57 ylim([min(value) max(value)]);
58 ylabel(yl, 'Interpreter', 'latex');
59 title('value - iterative step');
60 hold on;
61
62 subplot(1,3,2);
63 semilogy([0:(length(value)-2)], value(1:length(value)-1)-opt, '-');
64 xlabel('iterative step');
65 yl2 = '$f(\textbf{x}^k)-p^*$';
66 ylabel(yl2, 'Interpreter', 'latex');
67 title('value between opt - iterative step');
68 hold on;
69
70 subplot(1,3,3);
71 scatter([1:length(step)], step, 'filled', 'black');
72 xlabel('iterative step');
73 ylabel('$t^k$', 'Interpreter', 'latex');
74 title('step size - iterative step');
75 hold on;
76
77
78
79 function res = func(x)
80 global A;
81 res = -sum(log(1-A*x)) - sum(log(1+x)) - sum(log(1-x));
82 end
83
84 function res = lambda(hess, grad)
85 res = grad' * hess^(-1) * grad;
86 end
87
88 %
89 %

```

## 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.

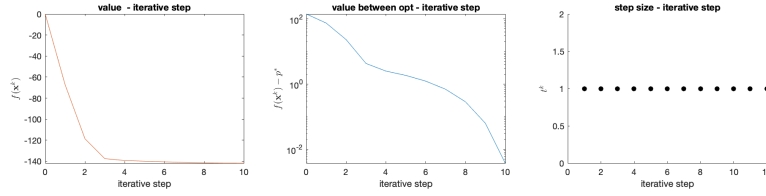


Figure 5: Instance 1, small instance

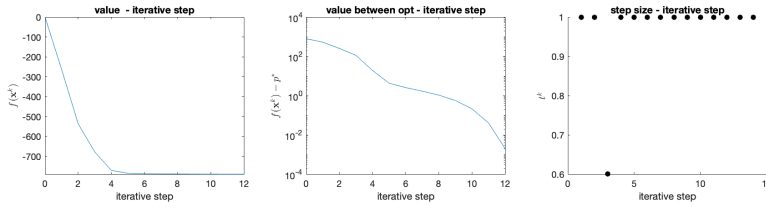


Figure 6: Instance 2, large instance

### 2.3.1 Convergency

The choose of  $\eta$  will depend convergency and iterative step.

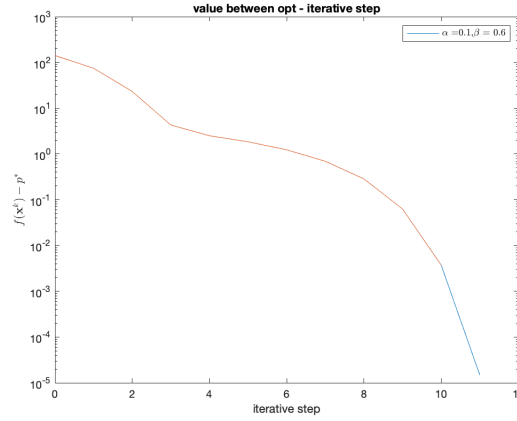


Figure 7: Different  $\eta$

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 evaluated.

### 3.2 code

```
1 %approximated damped Newton Method by reduce hessian calc
2 clear all;
3 % initialize
4 alpha = 0.1;
5 beta = 0.6;
6 epsilon = 1e-8;
7
8
9 % generate the random instance
10 global A;
11 %load('A_200_100.mat');
12 load('A_500_400.mat');
13 [m,n] = size(A);
14 value = {};
15 step = {};
16 %main iteration
17
18 index = 1;
19 %do three time
20 for N = [1,10,40]
21 % at step 0
22
23 value{index} = [];
24 step{index} = [];
25 inner_step = 0;
26 x = zeros(n,1);
27 grad = A'*(1./(1 - A*x)) - 1./(1+x) + 1./(1-x);
28 hessian = A'*diag((1./(1-A*x)).^2)*A + diag(1./(1+x).^2 + ...
29     1./(1-x).^2);
30 hessian_i = hessian ^(-1);
31 while 0.5*lambda(hessian_i, grad) > epsilon
32
33 value{index} = [value{index}, func(x)];
34 Δ_x = -hessian_i * grad;
35 t = 1;
36
37 % constrain the x in dom(x) by changing t
38 while ((max(A*(x+t*Δ_x)) ≥ 1) || (max(abs(x+t*Δ_x)) ≥ 1))
39 t = t * beta;
40 end
41
42 % backtracking line search
43 while (func(x+t*Δ_x) - func(x) > alpha * t * grad' * Δ_x)
```

```

44 t = t * beta;
45 end
46 step{index} = [step{index}, t];
47 % update x by:
48 x = x + t * Δ_x;
49 % update new gradient at x by:
50 grad = A*(1./(1 - A*x)) - 1./(1+x) + 1./(1-x);
51
52 % update new hessian and inverse at x by every N step:
53 if (mod(inner_step, N) == 0)
54 hessian = A'*diag((1./(1-A*x)).^2)*A + diag(1./(1+x).^2 + ...
55     1./(1-x).^2);
56 hessian_i = hessian ^ (-1);
57 end
58 inner_step = inner_step + 1;
59 index = index + 1;
60
61 end
62
63 %dump result
64 opt = min(value{3});
65
66
67 figure(1)
68 N = [1,10,40];
69 for i = 1:3
70 subplot(3,3,i);
71 plot([0:(length(value{i})-2)], value{i}(1:length(value{i})-1), '-');
72 yl = '$f(\textbf{x}^k)$';
73 xlabel('iterative step');
74 ylim([min(value{i}) max(value{i})]);
75 ylabel(yl, 'Interpreter', 'latex');
76 title(['N=' num2str(N(i))]);
77 hold on;
78 end
79
80 for i = 1:3
81 subplot(3,3,i+3)
82 semilogy([0:(length(value{i})-2)], ...
83     value{i}(1:length(value{i})-1)-opt, '-');
84 xlabel('iterative step');
85 yl2 = '$f(\textbf{x}^k)-p^*$';
86 ylabel(yl2, 'Interpreter', 'latex');
87 title(['N=' num2str(N(i))]);
88 hold on;
89 end
90
91 for i = 1:3
92 subplot(3,3,i+6)
93 scatter([1:length(step{i})], step{i}, 'filled', 'black');
94 xlabel('iterative step');
95 ylabel('$t^k$', 'Interpreter', 'latex');
96 title(['N=' num2str(N(i))]);
97 hold on;
98 end

```

```

99 function res = func(x)
100 global A;
101 res = -sum(log(1-A*x)) - sum(log(1+x)) - sum(log(1-x));
102 end
103
104 function res = lambda(hess_i, grad)
105 res = grad' * hess_i * grad;
106 end
107
108 %
109 %

```

### 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.

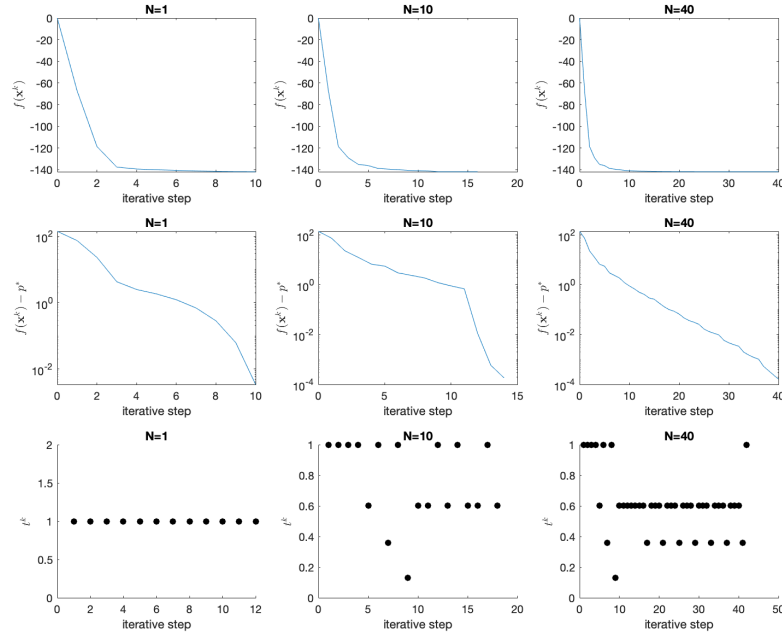


Figure 8: Instance 1, small instance

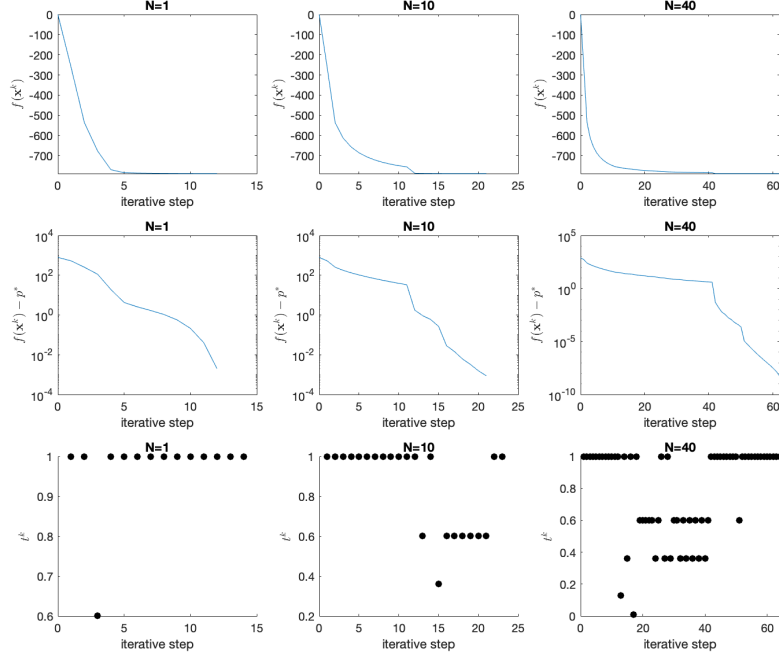


Figure 9: Instance 2, large instance

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

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

$k$  is the component parameter of diagonal matrix.

#### 3.3.1 code

```

1 %approximated damped Newton Method by reduce hessian to its ...
  diagonal .
2

```

```

3 % initialize
4 alpha = 0.1;
5 beta = 0.6;
6 epsilon = 1e-8;
7
8
9 % generate the random instance
10 global A;
11 %load('A_200_100.mat');
12 load('A_500_400.mat');
13 [m,n] = size(A);
14 value = [];
15 step = [];
16
17 %main iteration
18
19
20 % at step 0
21 x = zeros(n,1);
22 grad = A'*(1./(1 - A*x)) - 1./(1+x) + 1./(1-x);
23 hessian = A'*diag((1./(1-A*x)).^2)*A + diag(1./(1+x).^2 + ...
24     1./(1-x).^2);
25 hd = diag(diag(hessian));
26 while 0.5 * lambda(hd, grad) > epsilon
27     value = [value, func(x)];
28     Δ_x = -hd^(-1) * grad;
29     t = 1;
30
31 % constrain the x in dom(x) by changing t
32 while ((max(A*(x+t*Δ_x)) ≥ 1) || (max(abs(x+t*Δ_x)) ≥ 1))
33     t = t * beta;
34 end
35
36 % backtracking line search
37 while (func(x+t*Δ_x) - func(x) > alpha * t * grad' * Δ_x)
38     t = t * beta;
39 end
40 step = [step, t];
41 % update x by:
42 x = x + t * Δ_x;
43 % update new gradient and hessian at x by:
44 grad = A'*(1./(1 - A*x)) - 1./(1+x) + 1./(1-x);
45 hessian = A'*diag((1./(1-A*x)).^2)*A + diag(1./(1+x).^2 + ...
46     1./(1-x).^2);
47 hd = diag(diag(hessian));
48 end
49 %dump result
50 opt = min(value);
51
52
53 figure(1)
54 subplot(1,3,1);
55 plot([0:(length(value)-2)], value(1:length(value)-1), '-');
56 yl = '$f(\textbf{x}^k)$';
57 xlabel('iterative step');

```

```

58 ylim([min(value) max(value)]);
59 ylabel(y1,'Interpreter','latex');
60 title('value - iterative step');
61 hold on;
62
63 subplot(1,3,2);
64 semilogy([0:(length(value)-2)], value(1:length(value)-1)-opt, '-');
65 xlabel('iterative step');
66 yl2 = '$f(\textbf{x}^k)-p^*$';
67 ylabel(yl2,'Interpreter','latex');
68 title('value between opt - iterative step');
69 hold on;
70
71 subplot(1,3,3);
72 scatter([1:length(step)], step, 'filled', 'black');
73 xlabel('iterative step');
74 ylabel('$t^k$', 'Interpreter', 'latex');
75 title('step size - iterative step');
76 hold on;
77
78 function res = func(x)
79 global A;
80 res = -sum(log(1-A*x)) - sum(log(1+x)) - sum(log(1-x));
81 end
82
83
84 function res = lambda(hess, grad)
85 res = grad' * hess^(-1) * grad;
86 end

```

### 3.4 Result

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

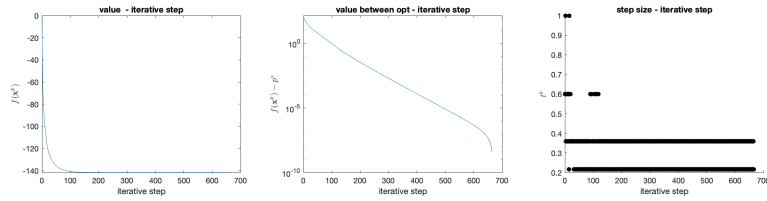


Figure 10: Instance 1,small instance

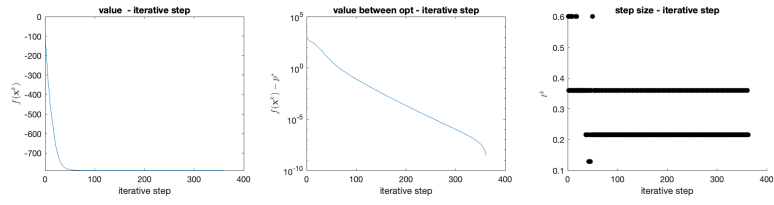


Figure 11: Instance 2, large instance

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