EE5138 Project

Submitted By:

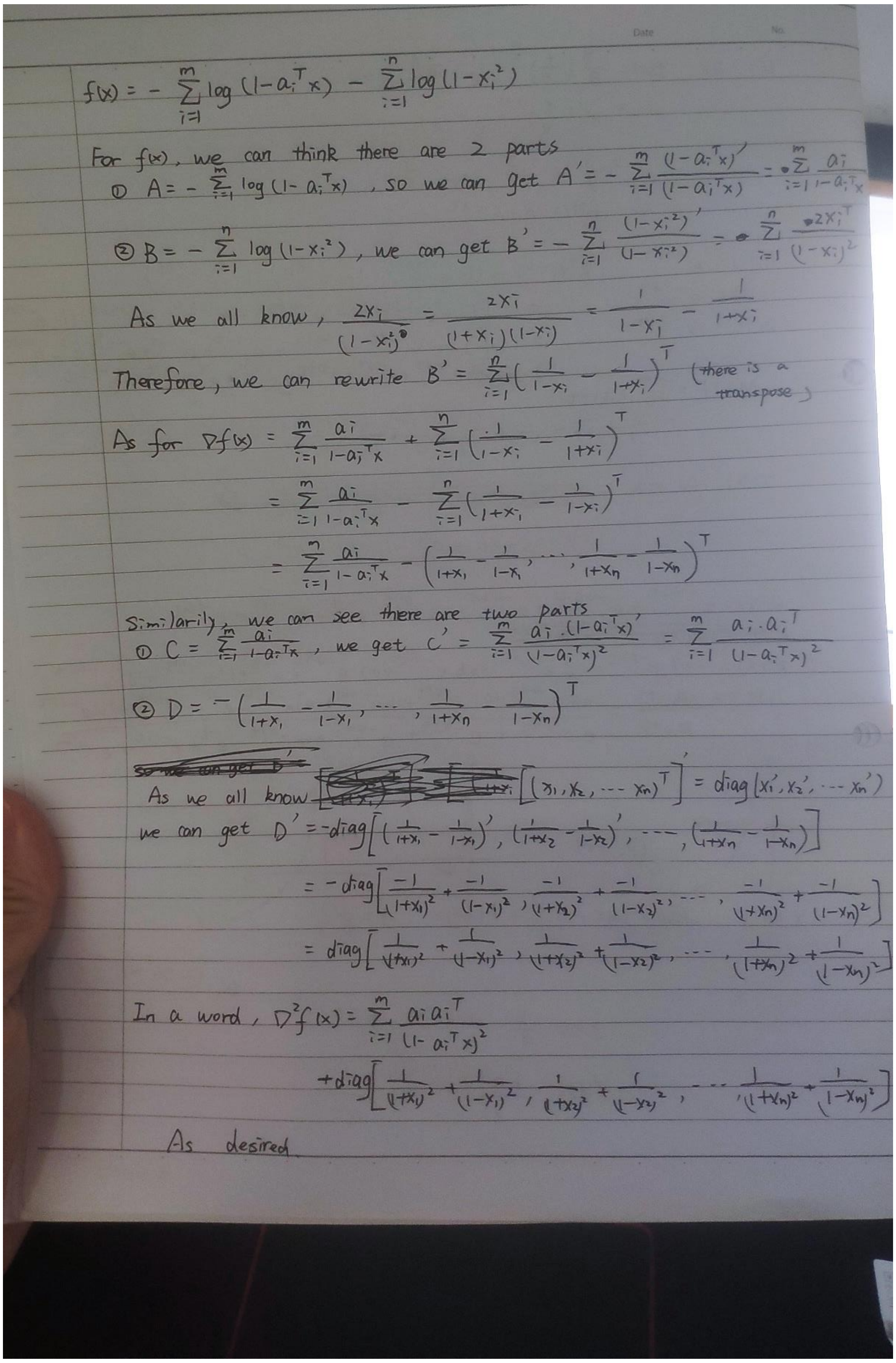
Name:	LUO ZIJIAN
Matric.No:	A0224725H
Mobile:	98912483

Electrical and Computer Engineering, School of Engineering

18 April 2021

1 (a)

There is my proof



Date No

$$f(x) = -\sum_{i=1}^{m} \log(1-a_i^T x) - \sum_{i=1}^{n} \log(1-x_i^2)$$

For $f(x)$, we can think there are 2 parts

① $A = -\sum_{i=1}^{m} \log(1-a_i^T x)$, so we can get $A' = -\sum_{i=1}^{m} \frac{(1-a_i^T x)'}{(1-a_i^T x)} = \sum_{i=1}^{m} \frac{a_i}{1-a_i^T x}$

② $B = -\sum_{i=1}^{n} \log(1-x_i^2)$, we can get $B' = -\sum_{i=1}^{n} \frac{(1-x_i^2)'}{(1-x_i^2)} = \sum_{i=1}^{n} \frac{2x_i^T}{(1-x_i)^2}$

As we all know, $\frac{2x_i}{(1-x_i^2)} = \frac{2x_i}{(1+x_i)(1-x_i)} = \frac{1}{1-x_i} - \frac{1}{1+x_i}$

Therefore, we can rewrite $B' = \sum_{i=1}^{n} \left(\frac{1}{1-x_i} - \frac{1}{1+x_i}\right)^T$ (there is a transpose)

As for $\nabla f(x) = \sum_{i=1}^{m} \frac{a_i}{1-a_i^T x} + \sum_{i=1}^{n} \left(\frac{1}{1-x_i} - \frac{1}{1+x_i}\right)^T$

$$= \sum_{i=1}^{m} \frac{a_i}{1-a_i^T x} - \sum_{i=1}^{n} \left(\frac{1}{1+x_i} - \frac{1}{1-x_i}\right)^T$$

$$= \sum_{i=1}^{m} \frac{a_i}{1-a_i^T x} - \left(\frac{1}{1+x_1} - \frac{1}{1-x_1}, \cdots, \frac{1}{1+x_n} - \frac{1}{1-x_n}\right)^T$$

Similarily, we can see there are two parts

① $C = \sum_{i=1}^{m} \frac{a_i}{1-a_i^T x}$, we get $C' = \sum_{i=1}^{m} \frac{a_i \cdot (1-a_i^T x)'}{(1-a_i^T x)^2} = \sum_{i=1}^{m} \frac{a_i \cdot a_i^T}{(1-a_i^T x)^2}$

② $D = -\left(\frac{1}{1+x_1} - \frac{1}{1-x_1}, \cdots, \frac{1}{1+x_n} - \frac{1}{1-x_n}\right)^T$

~~so we can get D'~~

As we all know [illegible] $\left[(x_1, x_2, \cdots x_n)^T\right]' = \mathrm{diag}(x_1', x_2', \cdots x_n')$

we can get $D' = -\mathrm{diag}\left[\left(\frac{1}{1+x_1} - \frac{1}{1-x_1}\right)', \left(\frac{1}{1+x_2} - \frac{1}{1-x_2}\right)', \cdots, \left(\frac{1}{1+x_n} - \frac{1}{1-x_n}\right)\right]$

$$= -\mathrm{diag}\left[\frac{-1}{(1+x_1)^2} + \frac{-1}{(1-x_1)^2}, \frac{-1}{(1+x_2)^2} + \frac{-1}{(1-x_2)^2}, \cdots, \frac{-1}{(1+x_n)^2} + \frac{-1}{(1-x_n)^2}\right]$$

$$= \mathrm{diag}\left[\frac{1}{(1+x_1)^2} + \frac{1}{(1-x_1)^2}, \frac{1}{(1+x_2)^2} + \frac{1}{(1-x_2)^2}, \cdots, \frac{1}{(1+x_n)^2} + \frac{1}{(1-x_n)^2}\right]$$

In a word, $\nabla^2 f(x) = \sum_{i=1}^{m} \frac{a_i a_i^T}{(1-a_i^T x)^2}$

$$+\mathrm{diag}\left[\frac{1}{(1+x_1)^2} + \frac{1}{(1-x_1)^2}, \frac{1}{(1+x_2)^2} + \frac{1}{(1-x_2)^2}, \cdots, \frac{1}{(1+x_n)^2} + \frac{1}{(1-x_n)^2}\right]$$

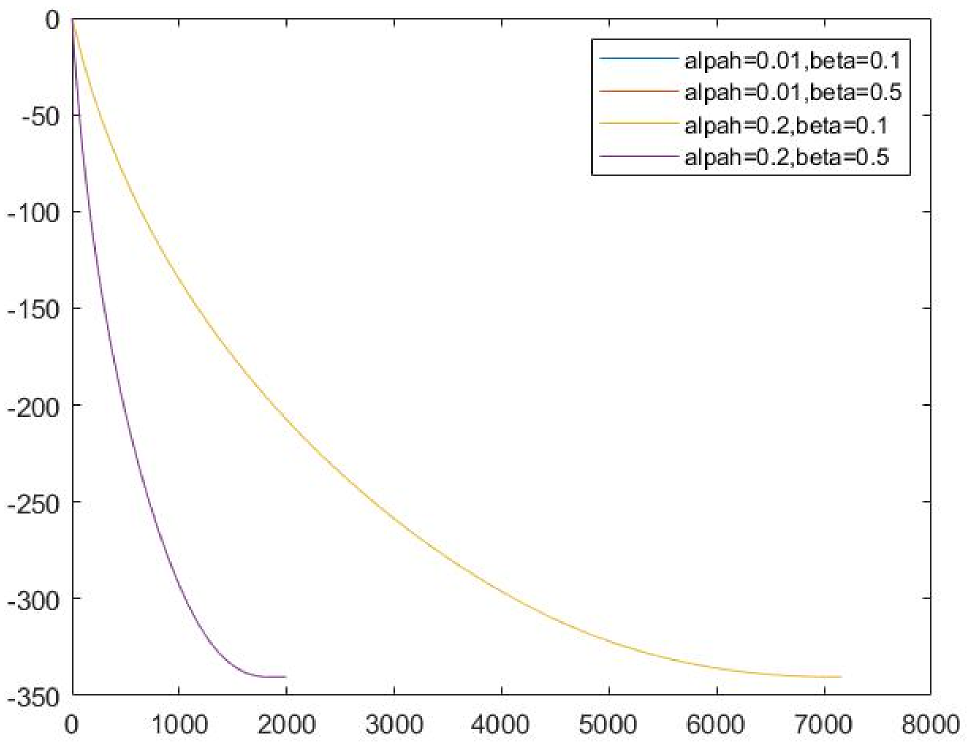
As desired

(b) Based on the proof of the gradient function of f(x), we get the first order and second order gradient function, and in this part, we use the gradient method to finish this problem. There are four groups of parameters.

(1) We get four optimal value from four groups

optimal value	parameters
-340.4403	Alpha=0.01, beta=0.1
-340.4402	Alpha=0.01, beta=0.5
-340.4403	Alpha=0.2, beta=0.1
-340.4402	Alpha=0.2, beta=0.5

(2) I plot four cases in one figure, here is the performance of gradient method in different groups of parameters.



In conclusions:

From the figure, we found the performance of same alpha is similar because the curve is covered. However, in fact, this suggest that the gradient method works better with fairly large alpha. That is because the first order of object function can more easily reach the optimal point.

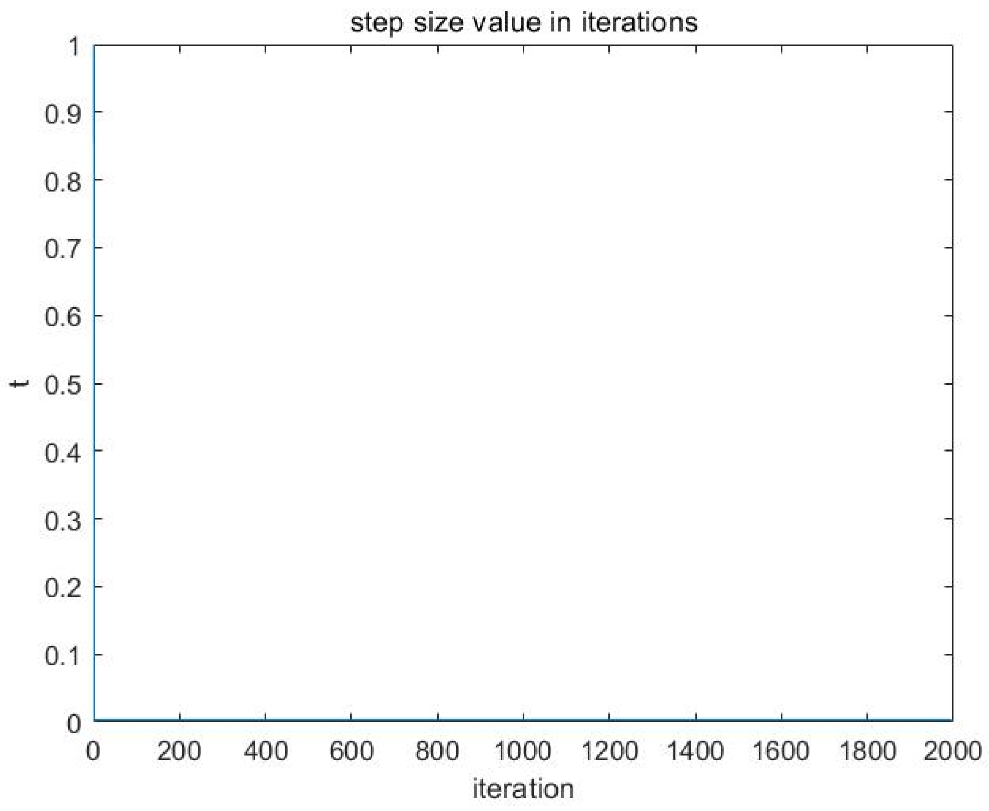
As for beta, bigger beta can reduce the number of iterations because it can make the step size fairly bigger than that of smaller beta. Therefore, beta make a great difference in performance of reducing iterations.

Here is the code

```
%% Clear all variables and close all
close all
clear
clc
alpha=0.01;
beta=0.1;
[f_recording1]=get_optimal(alpha,beta);
alpha=0.01;
beta=0.5;
[f_recording2]=get_optimal(alpha,beta);
alpha=0.2;
beta=0.1;
[f_recording3]=get_optimal(alpha,beta);
alpha=0.2;
beta=0.5;
[f_recording4]=get_optimal(alpha,beta);
%% plot the performance
figure(1)
x=1:length(f_recording1);
plot(x',f_recording1);
hold on
x=1:length(f_recording2);
plot(x',f_recording2);
hold on
x=1:length(f_recording3);
plot(x',f_recording3);
hold on
x=1:length(f_recording4);
plot(x',f_recording4);
hold on
legend('alpah=0.01,beta=0.1','alpah=0.01,beta=0.5',
'alpah=0.2,beta=0.1','alpah=0.2,beta=0.5');
function f_recording=get_optimal(alpha,beta)
%% intialize
m = 300;
n = 200;
x = zeros(n,1);
[f0,g] = gradient_function(x);
p_star=-340.4402;
t = 1;
f_recording = f0;
while norm(g) > 10^(-3)
    dx = -g/norm(g);
    f = gradient_function(x+t*dx);
    f1 = f0 + alpha*t*g'*dx;
    while abs(f-f1) >= 1
        t = beta*t;
        f = gradient_function(x+t*dx);
        f1 = f0 + alpha*t*g'*dx;
    end
    x = x+t*dx;
    [f0,g] = gradient_function(x);
    f_recording = [f_recording;f0];
    f0
    if f0<p_star
        break
    end
    if length(f_recording) > 10000
        break
    end
end
end
%% get the gradient function
function [f f_1] = gradient_function(x);
    A=dlmread('A.txt');
    m = 300;
    n = 200;
    fi = - log(1 - x.^2);
    for i = 1 : m
        di(i) = - log(1 - A(i,:)*x);
    end
    % get the intial f(x)
    f = sum(fi) + sum(di);
    for i = 1 : m
        fii(i,:) = A(i,:)/(1 - A(i,:)*x);
    end
    dii = 1./(1+x)-1./(1-x);
    % get the first order gradient
    f_1 = sum(fii)' - dii;
end
```

(3) For the case of alpha=0.01, beta=0.5

Using the result from part (2), I plot the change of step size in iterations.



Here is the code.

```
%% Clear all variables and close all
close all
clear
clc
%% intialize
m = 300;
n = 200;
x = zeros(n,1);
alpha=0.01;
beta=0.5;
[f0,g] = gradient_function(x);
p_star=-340.4402;
t = 1;
t_recording = t;
while norm(g) > 10^(-3)
    dx = -g/norm(g);
    f = gradient_function(x+t*dx);
    f1 = f0 + alpha*t*g'*dx;
    while abs(f-f1) >= 1
        t = beta*t;
        f = gradient_function(x+t*dx);
        f1 = f0 + alpha*t*g'*dx;
    end
    x = x+t*dx;
    [f0,g] = gradient_function(x);
    t_recording = [t_recording;t];
    f0
    if f0<p_star
        break
    end
end
%% plot the performance
figure(1)
x=1:length(t_recording);
plot(x',t_recoding);
xlabel("iteration");
ylabel("t");
title("step size value in iterations");
%% Get the gradient function
function [f f_1] = gradient_function(x);
    A=dlmread('A.txt');
    m = 300;
    n = 200;
    fi = - log(1 - x.^2);
    for i = 1 : m
        di(i) = - log(1 - A(i,:)*x);
    end
    f = sum(fi) + sum(di);
    for i = 1 : m
        fii(i,:) = A(i,:)/(1 - A(i,:)*x);
    end
    dii = 1./(1+x)-1./(1-x);
    f_1 = sum(fii)' - dii;
end
```

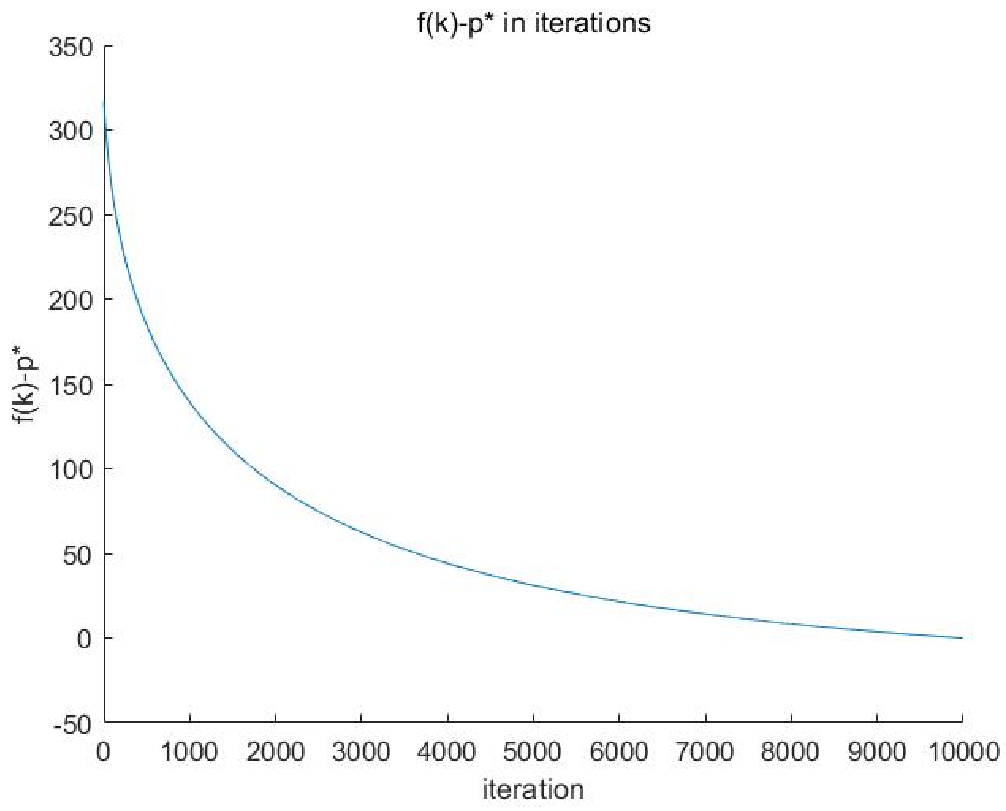
(c) Based on the proof of the gradient function of f(x), we get the first order and second order gradient function, and in this part, we use the Newton method to finish this problem.

(1) From Newton’s method, I got this optimal value

optimal value	parameters
-316.3919	Alpha=0.01, beta=0.5

(2)

Using the result from part (1), I plot the change of f(k)-p* in iterations.

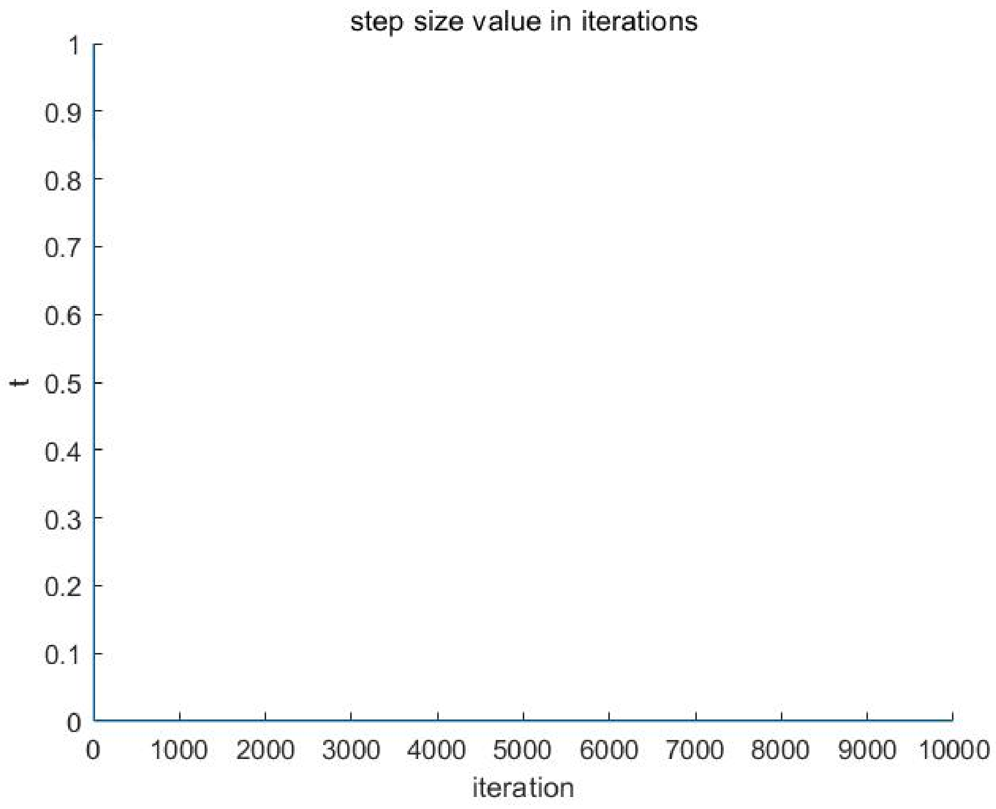


Here is the code

```
%% Clear all variables and close all
close all
clear
clc
%% intialize
m = 300;
n = 200;
x = zeros(n,1);
alpha=0.01;
beta=0.5;
[f0,g1,g2] = gradient_function(x);
t = 1;
p_star = -316.3885;
f_recording = f0 - p_star;
t_recording = t;
lambda = g1'*pinv(g2)*g1;
%% Newton's method
while lambda > 10^(-8)
    dx = -pinv(g2)*g1;
    f = gradient_function(x+t*dx);
    f1 = f0 + alpha*t*g1'*dx;
    while abs(f-f1) >= 1
        t = beta*t;
        f = gradient_function(x+t*dx);
        f1 = f0 + alpha*t*g1'*dx;
    end
    x = x + t*dx;
    [f0 g1 g2] = gradient_function(x);
    lambda = g1'*pinv(g2)*g1;

    f0
    t_recording = [t_recording;t];
    f_recording = [f_recording;f0 - p_star];

    if length(t_recording) > 10000
        break
    end
    if f0<p_star
        break
    end
end
%% plot the performance
figure(1);
hold on;
x=1:length(f_recording);
plot(x',f_recording);
xlabel("iteration");
ylabel("f(k)-p*");
title("f(k)-p* in iterations");
hold off;

figure(2);
hold on;
x=1:length(t_recording);
plot(x',t_recording);
xlabel("iteration");
ylabel("t");
title("step size value in iterations");
hold off;
%% get the gradient function
function [f df ddf] = gradient_function(x);
    A=dlmread('A.txt');
    m = 300;
    n = 200;
    fi = - log(1 - x.^2);
    for i = 1 : m
        di(i) = - log(1 - A(i,:)*x);
    end
    f = sum(fi) + sum(di);
    for i = 1 : m
        fii(i,:) = A(i,:)/(1 - A(i,:)*x);
    end
    dii = 1./(1+x)-1./(1-x);
    df = sum(fii)' - dii;
    for i = 1 : m
        fiii(:,:,i) = A(i,:)*A(i,:)'/(1 - 
A(i,:)*x).^2;
    end
    diii = (1./(1+x).^2) + (1./(1-x).^2);
    ddf = sum(fiii,3) + diag(diii);
end
```

(3)

I plot the change of step size versus iterations.



As for this part of code, I use part(2) code to plot this performance.