Convex and Nonsmooth Optimization - HW 5

Yunfei Ge (yg2047)

March 30, 2020

Problem 1

Answer:

(a) For positive definite matrix A, the minimizer of $\frac{1}{2}x^TAx + x^Tb$ can be compute in Matlab as

$$x^* = -A^{-1}b = \begin{bmatrix}-5 & 120 & -630 & 1120 & -630\end{bmatrix}^T$$

The optimal value

$$p^* = \frac{1}{2}x^{*T}Ax^* + x^{*T}b = -12.5$$

With gradient method, we obtain

$$f(x^{(0)}) = 8.2282 \qquad f(x^{(100)}) = -6.0571$$

Thus, we have

$$k = \frac{f(x^{(100)}) - p^*}{f(x^{(0)}) - p^*} = 0.31$$

The algorithm reduces $f(x) - p^*$ by $k = 0.31$ using the starting point provided.
Theoretically, we have

$$f(x^{(l)}) - p^* \leq c^l(f(x^{(0)}) - p^*) \qquad (c = 1 - 2m\alpha\min\{1, \beta/M\})$$

By computing the eigenvalues of A, we get

$$M = \lambda_{max}(A) = 1.5671 \qquad m = \lambda_{min}(A) = 3.2879e-06$$

Let $l = 100$, we got

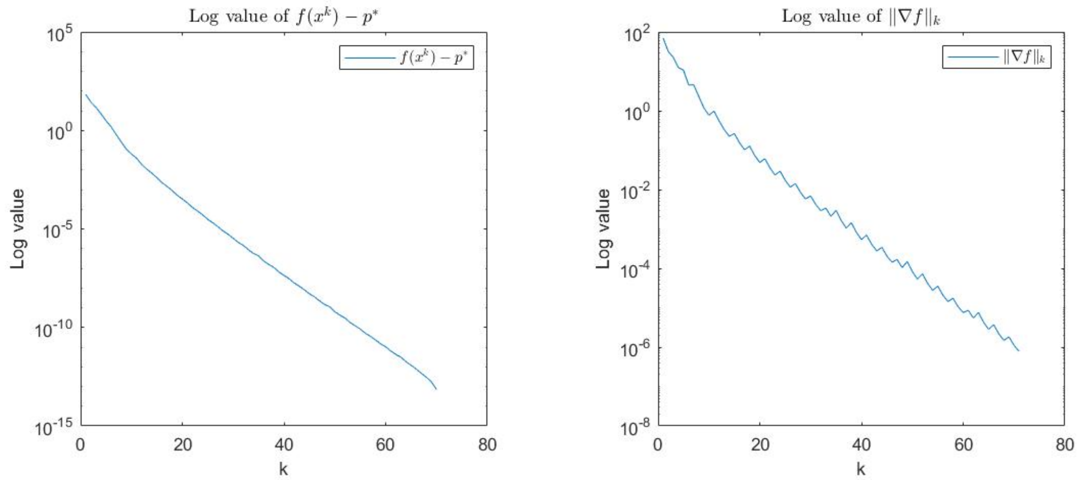
$$\frac{f(x^{(100)}) - p^*}{f(x^{(0)}) - p^*} \leq c^{100} = 0.9999$$

We can see that the theory in class provides an upper bound of convergence. In practice, the algorithm will converge faster than the upper bound in theory.

(b) With gradient method, the estimate of the minimal value

$$p^* = -67.4637$$

The figure below shows the log plots of $f(x^{(k)}) - p^*$ and the gradient norms. We can observe linear convergence in the figure.



If we use the following theory in the class to estimate M/m,

$$\frac{f(x^{(j)}) - p^*}{f(x^{(k)}) - p^*} \leq c^{j-k}$$

we'll find that the bound is too loose that the estimated bound for M/m does not provide any useful information.

To estimate the condition number M/m, we can compute the Hessian of different points and find the the largest and smallest eigenvalues of the Hessian $\|\nabla^2 f(x)\|$. With this method, we obtain that

$$\frac{M}{m} \approx 6.408 \times 10^4$$

Problem 2

Answer:

(a) If we use Newton's method in 1(a), the descent direction is

$$\Delta x_{NT} = -(\nabla^2 f(x))^{-1} \nabla f(x) = -A^{-1}(Ax + b) = -x - A^{-1}b$$

If we set the step size to $t = 1$, we have

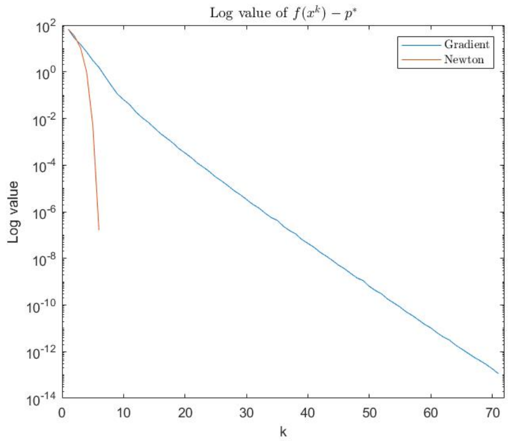
$$x' = x + t\Delta x_{NT} = x - x - A^{-1}b = -A^{-1}b$$

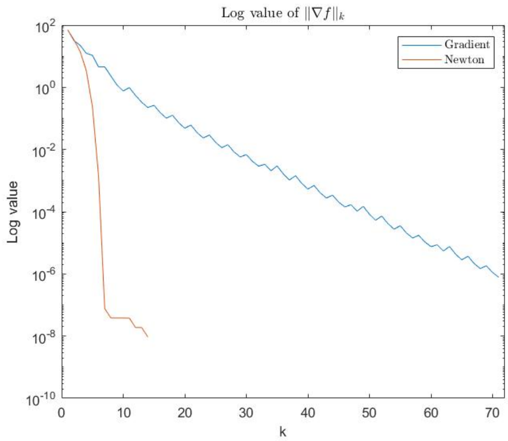
We know that for the problem in 1(a), the minimizer is exactly

$$x^* = -A^{-1}b$$

This means we can find the minimizer in only one step if we use Newton's method with problem 1(a). Thus, it is trivial to minimize the quadratic function in question 1(a) by Newton's method.

(b) The following figure shows the result.





We can observe quadratic convergence with Newton's method.

With Newton's method, we need total 14 iterations in Matlab to satisfy the tolerance of 10−8 on the norm of gradient. In fact, we can see that the function value converges after only 7 iterations. The rest of the iterations are only trying to reduce the norm of the gradient.

To compare the result with the theory discussed in the lecture, we first estimate the value of M, m, L. With Matlab computation, we obtain

$$M = 1.2828 \times 10^5 \qquad m = 2 \qquad L = 1.9251 \times 10^5$$

From the theory, we know that

$$f(x^{(l)}) - p^* \leq \frac{2m^3}{L^2} \left(\frac{1}{2}\right)^{2^{l-k+1}}$$

If we want the LHS $\leq \epsilon$, we need

$$\left(\frac{1}{2}\right)^{2^{l-k+1}} \leq \frac{\epsilon L^2}{2m^3}$$

Let $\epsilon_0 = \frac{2m^3}{L^2}$, The number of iterations should satisfy

$$T = l - k + 1 \geq log_2 log_2 \frac{\epsilon_0}{\epsilon}$$

Apply the values above, we find that

$$\epsilon_0 = \frac{2m^3}{L^2} = 8.3 \times 10^{-5}$$

From the graph, we also notice that after $\|\nabla f(x)\| \leq \eta = 10^{-8}$, the value $f(x) - p^*$ is very small. Set $\epsilon = 10^{-15}$. We obtain

$$T \geq 5.18$$

This implies we need at least 6 iterations for the Newton's method to converge, which is correct with our algorithm result.

Matlab Code Problem 1

1.(a) gradmeth.m

```
function [f_all,gnorm_all] = gradmeth(fun, x0, tol, maxit)
% code for gradient method, including backtracking line search
% Input
%       fun: anonymous function
%       x0: starting point
%       tol: tolerance on the norm of the gradient
%       maxit: max number of iterations
%
% Output
%       f_all: vector of function values f(x(k))
%       gnorm_all: corresponding vector of gradient norms


% Initialization
i = 1; % number of iterations
f_all = [];
gnorm_all = [];
x = x0;

while i <= maxit % control the number of iterations
    [f g] = fun(x);
    dx = -g; % use negative gradient as descent direction
    f_all(i) = f;
    gnorm_all(i) = norm(g);


    % Backtraking Line Search
    t = BTLS(fun, x, f, g, dx); % determine proper step size
    x = x + t.*dx; % update x

    if gnorm_all(i) < tol % check tolerance on the norm of the gradient
        break
    end

    i = i+1;
end
end


function t = BTLS(fun, x, f, g, dx)
% code for backtracking line search
% Input
%       fun: anonymous function
%       x: current position x
%       f: function value at x
%       g: gradient at x
%       dx: descent direction
%
% Output
%       t: step size in gradient descent

% Parameters
```

```
alpha = 0.25;
beta = 0.5;

% Default step size
t = 1;

while 1
    f_new = fun(x+t.*dx);

    if f_new > f + alpha.*t.*g'*dx
        t = beta*t;
    else
        break;
    end
end
end
```

1.(b) Objective function for 9.30

```
function [f,g] = funb(x,A)
% objective function for 9.30
[n m] = size(A);

if max(A'*x) < 1 && norm(x, inf) < 1 % x in dom f

    % function value
    f = -sum(log(1-A'*x)) - sum(log(1+x)) - sum(log(1-x));
    % gradient
    g = A*(1./(1-A'*x)) - 1./(1+x) + 1./(1-x);

else % x not in dom f

    % function value
    f = inf;
    % gradient
    g = nan*zeros(n, 1);
end
end
```

1.(b) Main Code

```
%% 1.(b)
clear all;
close all;
clc;
load Adata.mat

% Initial size
n = 100;
m = 50;

fun = @(x)funb(x, A);
```

```
x0 = zeros(n,1); % Starting point
tol = 1e-6;
maxit = 100;

% Run gradient method
[f_all,gnorm_all] = gradmeth(fun, x0, tol, maxit);
p_star = f_all(end) % optimal value

% plot the function values
tiledlayout(1,2);
nexttile
semilogy(f_all-p_star);
xlim([0 80]);
xlabel('k');
ylabel('Log value');
title('Log value of $f(x^k)-p^*$','Interpreter','latex');
legend('$f(x^k)-p^*$','Interpreter','latex');

% plot the gradient
nexttile
semilogy(gnorm_all);
xlim([0 80]);
xlabel('k');
ylabel('Log value');
title('Log value of $\Vert \nabla f \Vert_k$','Interpreter','latex');
legend('$\Vert \nabla f \Vert_k$','Interpreter','latex');
```

Matlab Code Problem 2

2.(b) Objective function (with Hessian)

```
function [f,g,h] = fun2b(x,A)
% objective function for 9.30
[n m] = size(A);

if max(A'*x) < 1 && norm(x, inf) < 1 % x in dom f

    % function value
    f = -sum(log(1-A'*x)) - sum(log(1+x)) - sum(log(1-x));

    % gradient
    g = A*(1./(1-A'*x)) - 1./(1+x) + 1./(1-x);

    % hessian
    h = zeros(n, n);
    for i = 1: m
        h = h + (A(:, i)*A(:, i)') / (1-A(:, i)'*x)^2;
    end
    h = h + 2*diag( (ones(n,1)+diag(x)*x) ./ (ones(n,1)-diag(x)*x).^2 );


else % x not in dom f

    % function value
    f = inf;
    % gradient
    g = nan*zeros(n, 1);
    % hessian
    h = nan*zeros(n, n);
end
end
```

2.(b) Main Code

```
%% 2.(b)
close all;
load Adata.mat

% Initial size
n = 100;
m = 50;

fun = @(x)fun2b(x, A);
x0 = zeros(n,1); % Starting point
tol = 1e-8;
maxit = 100; % Terminates when the norm of the gradient is 1e?8

[f2_all,gnorm2_all] = newtmeth(fun, x0, tol, maxit);
p_star = f2_all(end) % optimal value

% plot the function values
```

```
tiledlayout(1,2);
nexttile
semilogy(f_all-p_star);
hold on
semilogy(f2_all-p_star);
xlim([0 72]);
xlabel('k');
ylabel('Log value');
title('Log value of $f(x^k)-p^*$','Interpreter','latex');
legend('Gradient','Newton','Interpreter','latex');

% plot the gradient
nexttile
semilogy(gnorm_all);
hold on
semilogy(gnorm2_all)
xlim([0 72]);
xlabel('k');
ylabel('Log value');
title('Log value of $\Vert \nabla f \Vert_k$','Interpreter','latex');
legend('Gradient','Newton','Interpreter','latex');
```

2.(b) Estimate M, m, L

```
function [M_max, m_min, L_max] = estimate(fun,x_all,A)
% code to estimate the value of M, m and L
% Input
%       x_all: the point x at each iteration
%
% Output
%       M_max: estimated value of M
%       m_min: estimated value of m
%       L_max: estimated value of L

[n m] = size(A);

% Obtain k feasible points (along with the inital / final point)
x = cell(0);
xfin = x_all(end);
x(1) = {x0};
x(2) = {xfin{1}};
k = 50;

for i = 3: k+2
    xrad = rand(n, 1) - 0.5;
    while 1
        if max(A'*xrad) >= 1 || norm(xrad, inf) >= 1 % not feasible
            xrad = 0.1 * xrad;
        else
            break
        end
    end
    x(i) = {xrad};
end
```

```

% Compute M and m for all x
M = zeros(k+2, 1);
m = zeros(k+2, 1);
for i = 1: k+2
    [~,~,h] = fun(x{i});
    m(i) = min(eig(h));
    M(i) = max(eig(h));
end
% Find the largest M and smallest m
m_min = min(m);
M_max = max(M);


% Compute L
C = combnk(1:k+2, 2);
size_C = size(C, 1);
L = zeros(size_C, 1);
for i = 1: size_C
    p1 = x{C(i, 1)};
    p2 = x{C(i, 2)};
    [~, ~, h1] = fun(p1);
    [~, ~, h2] = fun(p2);
    L(i) = norm(h1-h2, 2) / norm(p1-p2, 2);
end
L_max = max(L);
end
```

Acknowledgement:

Reference: Convex Optimization solutions manual.