IEOR 160 Course Project

Qingan Zhao
SID: 3033030808

20 Oct. 2017

All plots and relevant code are generated in MATLAB.

Problem 1

This problem is solved using Gradient method with the backtracking line search ($\alpha = 1$ and $\beta = 0.6$).

Code:

```
clear all

syms x1 x2;
tol = 1e-6;    %set the tolerance
max_iter = 50;  %set the maximum iteration

%backtracking parameter
alpha = 1;
beta = 0.6;

%create the function f(x) and its gradient
func = exp(x1-x2-0.4)+exp(x1+x2-0.4)+(x1-1)^2+(x2+1)^2;
func_grad = (gradient(func))';
f = inline(func, 'x1', 'x2');
grad = inline(func_grad, 'x1', 'x2');

%initialization (denote x(0) by (0,0))
x = [0; 0];
k = [0:50];
x1_k = [0:50];
x2_k = [0:50];
f_k = [f(0,0), 1:50];
i = 0;
grad_norm = sqrt(grad(x(1), x(2)) * grad(x(1), x(2))');

```

```matlab
%gradient method with backtracking line search
while grad_norm>tol && i<=max_iter
    step = alpha;
    tmp_x = x - step * (grad(x(1), x(2)))';
    %backtracking line search
    while f(tmp_x(1), tmp_x(2)) > f(x(1), x(2))
        step = step * beta;
        tmp_x = x - step * (grad(x(1), x(2)))';
    end
    x = x - step * (grad(x(1), x(2)))';
    grad_norm = sqrt(grad(x(1), x(2)) * grad(x(1), x(2))');
    i = i + 1;
    x1_k(i+1) = x(1);
    x2_k(i+1) = x(2);
    f_k(i+1) = f(x(1), x(2));
end

%consider the case that the function converges before reaching the maximum iteration
if i<max_iter
    for j=i+1:max_iter+1
        x1_k(j) = x1_k(i);
        x2_k(j) = x2_k(i);
        f_k(j) = f(x(1), x(2));
    end
end

%visualization
%plot f(x(k)) versus k for k=0,1,2,...,50
plot(k, f_k, '.-k', 'MarkerSize', 15);
set(gca, 'FontSize', 15);
xlabel('k', 'FontSize', 15);
ylabel('f(x(k))', 'FontSize', 15);
%plot the trajectory of points x(0), x(1), ..., x(50)
figure;
plot(x1_k, x2_k, '.-k', 'MarkerSize', 15);
set(gca, 'FontSize', 15);
xlabel('x1', 'FontSize', 15);
ylabel('x2', 'FontSize', 15);
fprintf('The minimum value of the function is %f, where x=(%f, %f).\n', f(x(1), x(2)), x(1), x(2));
```

The minimum value of the function is **2.722814**, where x=**(0.122218, -0.556434)**.

The plot of $f(x^{(k)})$ versus k for $k = 0, 1, 2, ..., 50$ is shown in Figure 1.

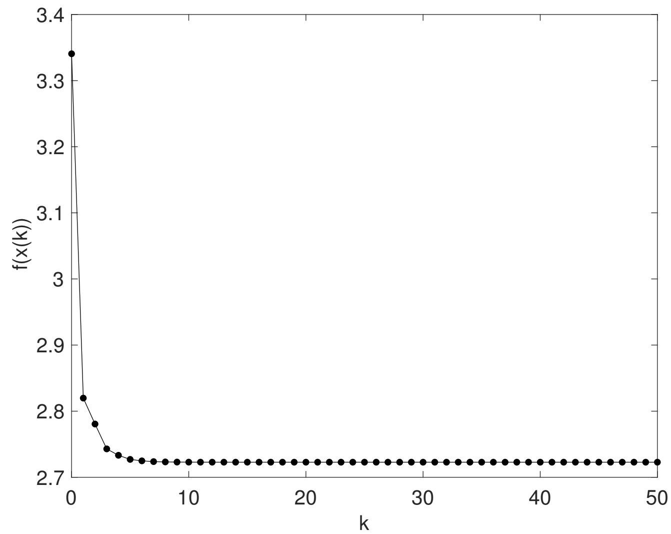

Fig. 1: $f(x^{(k)})$ versus k for $k = 0, 1, 2, ..., 50$ (Gradient method)

The trajectory of points $x^{(0)}$, $x^{(1)}$, ..., $x^{(50)}$ is shown in Figure 2.

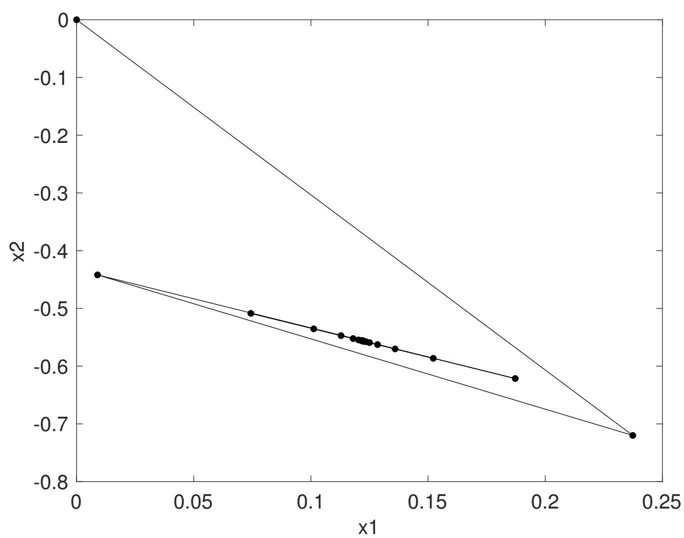

Fig. 2: The trajectory of points $x^{(0)}$, $x^{(1)}$, ..., $x^{(50)}$ (Gradient method)

Problem 2

This problem is solved using Newton's method with the backtracking line search ($\alpha = 1$ and $\beta = 0.6$).

Code:

```
clear all

syms x1 x2;
tol = 1e-6;    %set the tolerance
max_iter = 50;  %set the maximum iteration

%backtracking parameter
alpha = 1;
beta = 0.6;

%create the function f(x) and its gradient and hessian
func = exp(x1-x2-0.4)+exp(x1+x2-0.4)+(x1-1)^2+(x2+1)^2;
func_grad = (gradient(func))';
func_hess = (hessian(func));
f = inline(func, 'x1', 'x2');
grad = inline(func_grad, 'x1', 'x2');
hess = inline(func_hess, 'x1', 'x2');

%initialization (denote x(0) by (0,0))
x = [0; 0];
k = [0:50];
x1_k = [0:50];
x2_k = [0:50];
f_k = [f(0,0), 1:50];
i = 0;
grad_norm = sqrt(grad(x(1), x(2)) * grad(x(1), x(2))');

%Newton's method with backtracking line search
while grad_norm>tol && i<=max_iter
    step = alpha;
    tmp_x = x - step * hess(x(1), x(2)) \ (grad(x(1), x(2)))';
    %backtracking line search
    while f(tmp_x(1), tmp_x(2)) > f(x(1), x(2))
        step = step * beta;
        tmp_x = x - step * hess(x(1), x(2)) \ (grad(x(1), x(2)))';
    end
    x = x - step * hess(x(1), x(2)) \ (grad(x(1), x(2)))';
    grad_norm = sqrt(grad(x(1), x(2)) * grad(x(1), x(2))');
    i = i + 1;
    x1_k(i+1) = x(1);
    x2_k(i+1) = x(2);
    f_k(i+1) = f(x(1), x(2));
end

%consider the case that the function converges before reaching the maximum iteration
if i<max_iter
    for j=i+1:max_iter+1
```

```
        x1_k(j) = x1_k(i);
        x2_k(j) = x2_k(i);
        f_k(j) = f(x(1), x(2));
    end
end

%visualization
%plot f(x(k)) versus k for k=0,1,2,...,50
plot(k, f_k, '.-k', 'MarkerSize', 15);
set(gca, 'FontSize', 15);
xlabel('k', 'FontSize', 15);
ylabel('f(x(k))', 'FontSize', 15);
%plot the trajectory of points x(0), x(1), ..., x(50)
figure;
plot(x1_k, x2_k, '.-k', 'MarkerSize', 15);
set(gca, 'FontSize', 15);
xlabel('x1', 'FontSize', 15);
ylabel('x2', 'FontSize', 15);
fprintf('The minimum value of the function is %f, where x=(%f, %f).\n', f(x(1)
    , x(2)), x(1), x(2));
```

The minimum value of the function is **2.722814**, where x=**(0.122219, -0.556434)**.

The plot of $f(x^{(k)})$ versus k for $k = 0, 1, 2, ..., 50$ is shown in Figure 3.

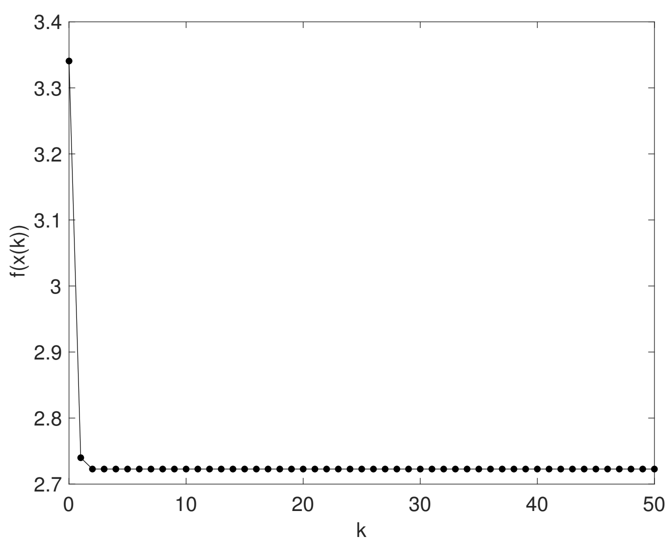

Fig. 3: $f(x^{(k)})$ versus k for $k = 0, 1, 2, ..., 50$ (Newton's method)

The trajectory of points $x^{(0)}$, $x^{(1)}$, ..., $x^{(50)}$ is shown in Figure 4.

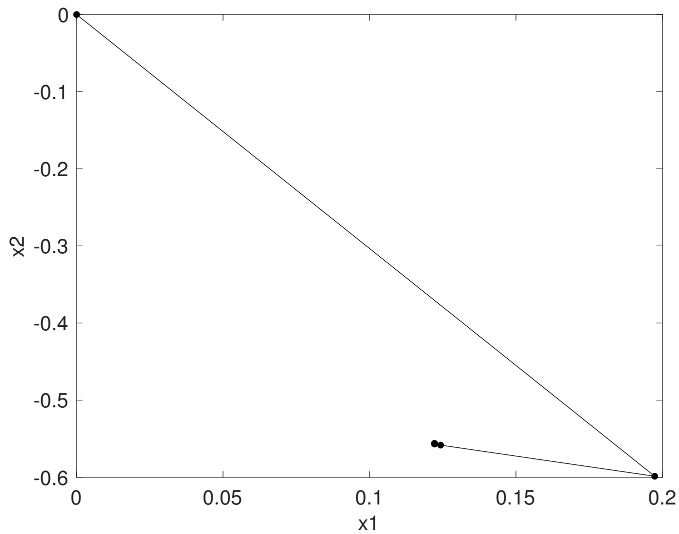

Fig. 4: The trajectory of points $x^{(0)}$, $x^{(1)}$, ..., $x^{(50)}$ (Newton's method)

Problem 3

This problem is solved using Newton's method with the backtracking line search ($\alpha = 0.4$ and $\beta = 0.4$).

The function in this problem is much more complex than the previous one. Hence, to speed up the code, let's first calculate the gradient and hessian by hand instead of using *gradient*() and *hessian*() functions in MATLAB. Assume the *log*() in the problem is the logarithm to the base 10.

Another thing needs to mention is that the code is not efficient enough to run with 500 variables, so let's just run with 100 variables instead.

Code:

(1) The f function (the function in the problem)

```
%this function is to calculate the function in problem 3
function [fun] = f(x, a)
%x is the input vector, a is the random matrix generated in the main script
fun = 0;
for i=1:100
    fun = fun + log10(2-x(i)^2) + log10(1 - 2 * (a(:,i))' * x);
end
fun = -fun;
```

(2) The gradient function

```
%this function is to calculate gradient of the function in problem 3
function [grad] = grad_fun(x, a)

grad = zeros(1,100);
%x is the input vector, a is the random matrix generated in the main script
for i=1:100
    tmp = 0;
    for k=1:100
        tmp = tmp + 2 * a(i,k) / (log(10) * (1 - 2 * (a(:,k))' * x));
    end
    grad(i) = 2 * x(i) / (log(10) * (2-x(i)^2)) + tmp;
end
```

(3) The hessian function

```
%this function is to calculate hessian of the function in problem 3
function [hess] = hess_fun(x, a)
%x is the input vector, a is the random matrix generated in the main script

%divide the hessian matrix into two parts for convenience
hess_1 = zeros(100);
hess_2 = zeros(100);

for i=1:100
    hess_1(i,i) = 4 + 2 * x(i)^2 / (log(10) * (2 - x(i)^2)^2);
end

for i=1:100
    for j=1:100
        for k=1:100
            hess_2(i,j) = hess_2(i,j) + 2 * a(i,k) * 2 * a(j,k) / (log(10) * (1 - 2 * (a(:,k))' * x)^2);
        end
    end
end
hess = hess_1 + hess_2;
```

(4) Main script

```
clear all
%this code is not efficient enough to run 500 variables
%run it only 100 variables

rand_num = rand(100,100); %randomly generated matrix (n vectors)
tol = 1e-6;    %set the tolerance
max_iter = 300;  %set the maximum iteration
```

```
%backtracking parameter
alpha = 0.4;
beta = 0.4;

%initialization (denote x(0) by (0,0,...,0))
x = zeros(100,1);
k = [0:300];
f_k = [f(x, rand_num), 1:50];
i = 0;
grad_norm = sqrt(grad_fun(x, rand_num) * grad_fun(x, rand_num)');

%Newton's method with backtracking line search
while grad_norm>tol && i<=max_iter
    step = alpha;
    tmp_x = x - step * hess_fun(x, rand_num) \ (grad_fun(x, rand_num))';
    %backtracking line search
    while f(tmp_x, rand_num) > f(x, rand_num)
        step = step * beta;
        tmp_x = x - step * hess_fun(x, rand_num) \ (grad_fun(x, rand_num))';
    end
    x = x - step * hess_fun(x, rand_num) \ (grad_fun(x, rand_num))';
    grad_norm = sqrt(grad_fun(x, rand_num) * grad_fun(x, rand_num)');
    fprintf('The %dth iteration completed\n', i);   %monitor the iteration
    i = i + 1;
    f_k(i+1) = f(x, rand_num);
end
fprintf('All iterations completed\n');
%consider the case that the function converges before reaching the maximum
    iteration
if i<max_iter
    for j=i+1:max_iter+1
        f_k(j) = f(x, rand_num);
    end
end

%visualization
%plot f(x(k)) versus k for k=0,1,2,...,50
plot(k, f_k, '.-k', 'MarkerSize', 15);
set(gca, 'FontSize', 15);
xlabel('k', 'FontSize', 15);
ylabel('f(x(k))', 'FontSize', 15);
fprintf('The minimum value of the function is %f.\n', f(x, rand_num));
```

The minimum value of the function is **-204.012227.**

The plot of $f(x^{(k)})$ versus k for $k = 0, 1, 2, ..., 300$ is shown in Figure 5.

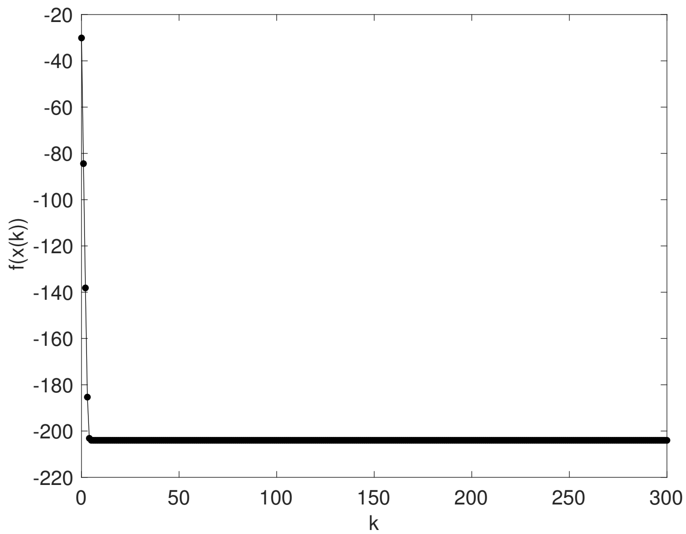

Fig. 5: $f(x^{(k)})$ versus k for $k = 0, 1, 2, ..., 300$ (Newton's method)