

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:

```
1 clear all
2
3 syms x1 x2;
4 tol = 1e-6; %set the tolerance
5 max_iter = 50; %set the maximum iteration
6
7 %backtracking parameter
8 alpha = 1;
9 beta = 0.6;
10
11 %create the function f(x) and its gradient
12 func = exp(x1-x2-0.4)+exp(x1+x2-0.4)+(x1-1)^2+(x2+1)^2;
13 func_grad = (gradient(func))';
14 f = inline(func, 'x1', 'x2');
15 grad = inline(func_grad, 'x1', 'x2');
16
17 %initialization (denote x(0) by (0,0))
18 x = [0; 0];
19 k = [0:50];
20 x1_k = [0:50];
21 x2_k = [0:50];
22 f_k = [f(0,0), 1:50];
23 i = 0;
24 grad_norm = sqrt(grad(x(1), x(2)) * grad(x(1), x(2))');
25
```

```

26 %gradient method with backtracking line search
27 while grad_norm>tol && i<=max_iter
28     step = alpha;
29     tmp_x = x - step * (grad(x(1), x(2)))';
30     %backtracking line search
31     while f(tmp_x(1), tmp_x(2)) > f(x(1), x(2))
32         step = step * beta;
33         tmp_x = x - step * (grad(x(1), x(2)))';
34     end
35     x = x - step * (grad(x(1), x(2)))';
36     grad_norm = sqrt(grad(x(1), x(2)) * grad(x(1), x(2)))';
37     i = i + 1;
38     x1_k(i+1) = x(1);
39     x2_k(i+1) = x(2);
40     f_k(i+1) = f(x(1), x(2));
41 end
42
43 %consider the case that the function converges before reaching the maximum
    iteration
44 if i<max_iter
45     for j=i+1:max_iter+1
46         x1_k(j) = x1_k(i);
47         x2_k(j) = x2_k(i);
48         f_k(j) = f(x(1), x(2));
49     end
50 end
51
52 %visualization
53 %plot f(x(k)) versus k for k=0,1,2,...,50
54 plot(k, f_k, '.-k', 'MarkerSize', 15);
55 set(gca, 'FontSize', 15);
56 xlabel('k', 'FontSize', 15);
57 ylabel('f(x(k))', 'FontSize', 15);
58 %plot the trajectory of points x(0), x(1), ..., x(50)
59 figure;
60 plot(x1_k, x2_k, '.-k', 'MarkerSize', 15);
61 set(gca, 'FontSize', 15);
62 xlabel('x1', 'FontSize', 15);
63 ylabel('x2', 'FontSize', 15);
64 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 $x=(\mathbf{0.122218, -0.556434})$.

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

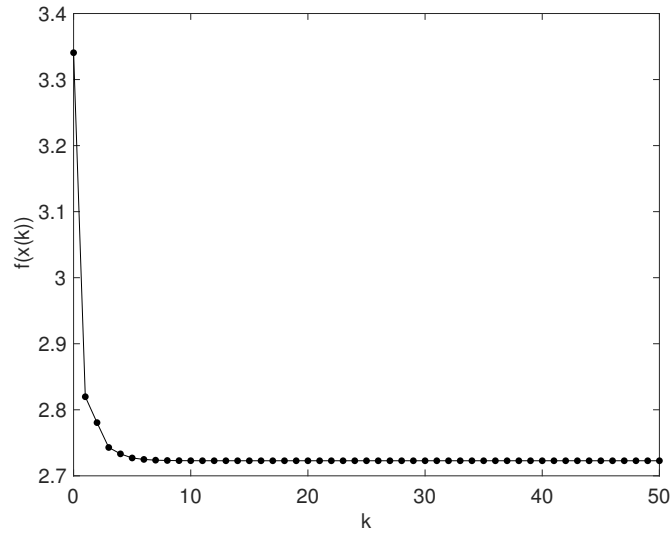


Fig. 1: $f(x^{(k)})$ versus k for $k = 0, 1, 2, \dots, 50$ (Gradient method)

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

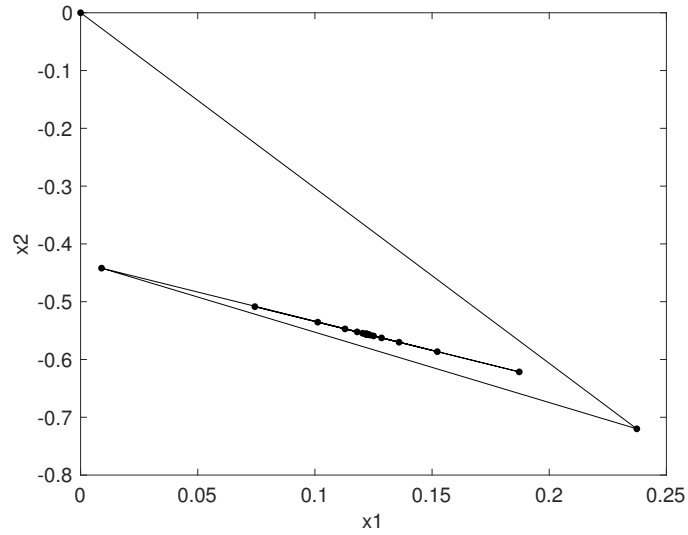


Fig. 2: The trajectory of points $x^{(0)}, x^{(1)}, \dots, x^{(50)}$ (Gradient method)

Problem 2

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

Code:

```
1 clear all
2
3 syms x1 x2;
4 tol = 1e-6; %set the tolerance
5 max_iter = 50; %set the maximum iteration
6
7 %backtracking parameter
8 alpha = 1;
9 beta = 0.6;
10
11 %create the function f(x) and its gradient and hessian
12 func = exp(x1-x2-0.4)+exp(x1+x2-0.4)+(x1-1)^2+(x2+1)^2;
13 func_grad = (gradient(func))';
14 func_hess = (hessian(func));
15 f = inline(func, 'x1', 'x2');
16 grad = inline(func_grad, 'x1', 'x2');
17 hess = inline(func_hess, 'x1', 'x2');
18
19 %initialization (denote x(0) by (0,0))
20 x = [0; 0];
21 k = [0:50];
22 x1_k = [0:50];
23 x2_k = [0:50];
24 f_k = [f(0,0), 1:50];
25 i = 0;
26 grad_norm = sqrt(grad(x(1), x(2)) * grad(x(1), x(2)))';
27
28 %Newton's method with backtracking line search
29 while grad_norm>tol && i<=max_iter
30     step = alpha;
31     tmp_x = x - step * hess(x(1), x(2)) \ (grad(x(1), x(2)))';
32     %backtracking line search
33     while f(tmp_x(1), tmp_x(2)) > f(x(1), x(2))
34         step = step * beta;
35         tmp_x = x - step * hess(x(1), x(2)) \ (grad(x(1), x(2)))';
36     end
37     x = x - step * hess(x(1), x(2)) \ (grad(x(1), x(2)))';
38     grad_norm = sqrt(grad(x(1), x(2)) * grad(x(1), x(2)))';
39     i = i + 1;
40     x1_k(i+1) = x(1);
41     x2_k(i+1) = x(2);
42     f_k(i+1) = f(x(1), x(2));
43 end
44
45 %consider the case that the function converges before reaching the maximum
    iteration
46 if i<max_iter
47     for j=i+1:max_iter+1
```

```

48         x1_k(j) = x1_k(i);
49         x2_k(j) = x2_k(i);
50         f_k(j) = f(x(1), x(2));
51     end
52 end
53
54 %visualization
55 %plot f(x(k)) versus k for k=0,1,2,...,50
56 plot(k, f_k, '.-k', 'MarkerSize', 15);
57 set(gca, 'FontSize', 15);
58 xlabel('k', 'FontSize', 15);
59 ylabel('f(x(k))', 'FontSize', 15);
60 %plot the trajectory of points x(0), x(1), ..., x(50)
61 figure;
62 plot(x1_k, x2_k, '.-k', 'MarkerSize', 15);
63 set(gca, 'FontSize', 15);
64 xlabel('x1', 'FontSize', 15);
65 ylabel('x2', 'FontSize', 15);
66 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 $k = 0, 1, 2, \dots, 50$ is shown in Figure 3.

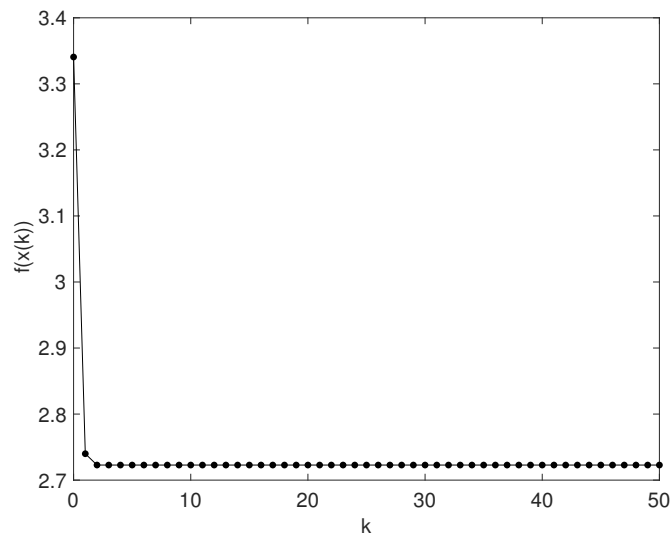


Fig. 3: $f(x^{(k)})$ versus k for $k = 0, 1, 2, \dots, 50$ (Newton's method)

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

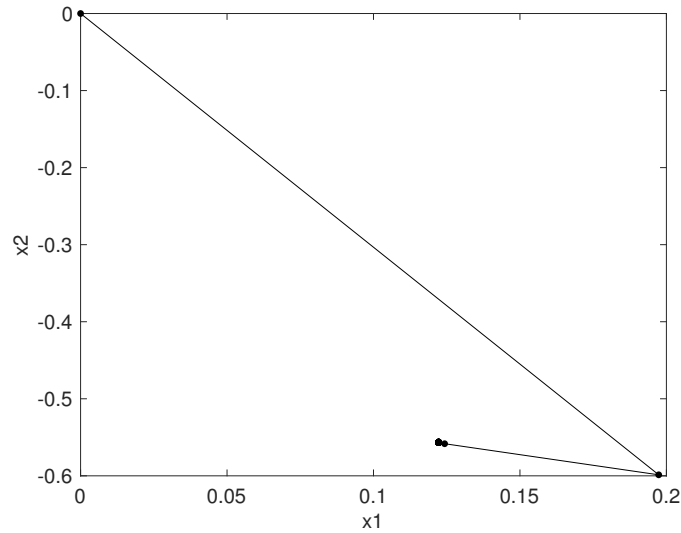


Fig. 4: The trajectory of points $x^{(0)}, x^{(1)}, \dots, x^{(50)}$ (Newton's method)

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)

```

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

```

(2) The gradient function

```
1 %this function is to calculate gradient of the function in problem 3
2 function [grad] = grad_fun(x, a)
3
4 grad = zeros(1,100);
5 %x is the input vector, a is the random matrix generated in the main script
6 for i=1:100
7     tmp = 0;
8     for k=1:100
9         tmp = tmp + 2 * a(i,k) / (log(10) * (1 - 2 * (a(:,k))' * x));
10    end
11    grad(i) = 2 * x(i) / (log(10) * (2-x(i)^2)) + tmp;
12 end
```

(3) The hessian function

```
1 %this function is to calculate hessian of the function in problem 3
2 function [hess] = hess_fun(x, a)
3 %x is the input vector, a is the random matrix generated in the main script
4
5 %divide the hessian matrix into two parts for convenience
6 hess_1 = zeros(100);
7 hess_2 = zeros(100);
8
9 for i=1:100
10    hess_1(i,i) = 4 + 2 * x(i)^2 / (log(10) * (2 - x(i)^2)^2);
11 end
12
13 for i=1:100
14    for j=1:100
15        for k=1:100
16            hess_2(i,j) = hess_2(i,j) + 2 * a(i,k) * 2 * a(j,k) / (log(10) *
17            (1 - 2 * (a(:,k))' * x)^2);
18        end
19    end
20 end
21 hess = hess_1 + hess_2;
```

(4) Main script

```
1 clear all
2 %this code is not efficient enough to run 500 variables
3 %run it with only 100 variables
4
5 rand_num = rand(100,100); %randomly generated matrix (n vectors)
6 tol = 1e-6; %set the tolerance
7 max_iter = 300; %set the maximum iteration
```

```

8
9 %backtracking parameter
10 alpha = 0.4;
11 beta = 0.4;
12
13 %initialization (denote x(0) by (0,0,...,0))
14 x = zeros(100,1);
15 k = [0:300];
16 f_k = [f(x, rand_num), 1:50];
17 i = 0;
18 grad_norm = sqrt(grad_fun(x, rand_num) * grad_fun(x, rand_num)');
19
20 %Newton's method with backtracking line search
21 while grad_norm>tol && i<=max_iter
22     step = alpha;
23     tmp_x = x - step * hess_fun(x, rand_num) \ (grad_fun(x, rand_num))';
24     %backtracking line search
25     while f(tmp_x, rand_num) > f(x, rand_num)
26         step = step * beta;
27         tmp_x = x - step * hess_fun(x, rand_num) \ (grad_fun(x, rand_num))';
28     end
29     x = x - step * hess_fun(x, rand_num) \ (grad_fun(x, rand_num))';
30     grad_norm = sqrt(grad_fun(x, rand_num) * grad_fun(x, rand_num)');
31     fprintf('The %dth iteration completed\n', i); %monitor the iteration
32     i = i + 1;
33     f_k(i+1) = f(x, rand_num);
34 end
35 fprintf('All iterations completed\n');
36 %consider the case that the function converges before reaching the maximum
    iteration
37 if i<max_iter
38     for j=i+1:max_iter+1
39         f_k(j) = f(x, rand_num);
40     end
41 end
42
43 %visualization
44 %plot f(x(k)) versus k for k=0,1,2,...,50
45 plot(k, f_k, '-k', 'MarkerSize', 15);
46 set(gca, 'FontSize', 15);
47 xlabel('k', 'FontSize', 15);
48 ylabel('f(x(k))', 'FontSize', 15);
49 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 $k = 0, 1, 2, \dots, 300$ is shown in Figure 5.

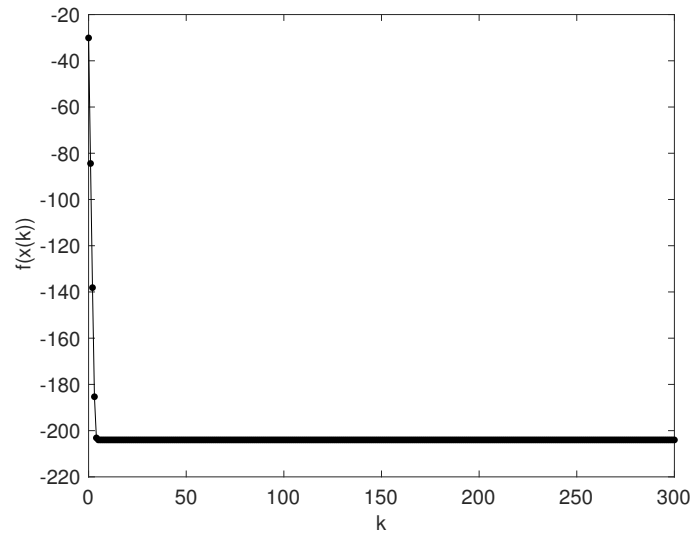


Fig. 5: $f(x^{(k)})$ versus k for $k = 0, 1, 2, \dots, 300$ (Newton's method)