수치해석 HW4

21700242 문선빈

Problem 1.

(a) $f(x) = \frac{1}{2}(x_1^3 - x_2)^2 + \frac{1}{2}(x_1 - 1)^2$

$f'(x_1) = (x_1^3 - x_2)\cdot(3x_1^2) + (x_1 - 1)$

$= 3x_1^5 - 3x_1^2 x_2 + x_1 - 1$ — ①

$f'(x_2) = (x_1^3 - x_2)\cdot(-1) = -x_1^3 + x_2$ — ②

minimum

② - $x_1^3 = x_2$

① - $3x_1^5 - 3x_1^2 x_2 + x_1 - 1 = 0$

$\cancel{3x_1^5} - \cancel{3x_1^2 x_1^3} + x_1 - 1$

$= x_1 - 1 = 0$

$\therefore x_1 = 1 \quad x_2 = 1$

(b)

$H_f(x_k) s_k = -\nabla f(x_k)$

$x_{k+1} = x_k + s_k$

$x_{k+1} = x_k - H_f^{-1}(x_k)\nabla f(x_k)$

$$Hf(x) = \nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_2^2} \end{bmatrix}$$

$\frac{\partial^2 f(x)}{\partial x_1^2} = 15x_1^4 - 6x_1 x_2 + 1$

$\frac{\partial^2 f(x)}{\partial x_1 x_2} = -3x_1^2$

$\frac{\partial^2 f(x)}{\partial x_2 \partial x_1} = -3x_1^2$

$\frac{\partial^2 f(x)}{\partial x_2^2} = 1$

$x_0 = \begin{bmatrix} 2 \\ 2 \end{bmatrix} \Rightarrow \begin{bmatrix} 217 & -12 \\ -12 & 1 \end{bmatrix} = H_f$

$H_f^{-1} = \frac{1}{217 \times 1 - (-12)(-12)} \begin{bmatrix} 1 & 12 \\ 12 & 217 \end{bmatrix}$

$= \frac{1}{73}\begin{bmatrix} 1 & 12 \\ 12 & 217 \end{bmatrix} = \begin{bmatrix} 0.014 & 0.0164 \\ 0.0164 & 35.671 \end{bmatrix}$

$x_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix} - \begin{bmatrix} 0.014 & 0.0164 \\ 0.0164 & 35.671 \end{bmatrix}\begin{bmatrix} 73 \\ -6 \end{bmatrix}$

$= \begin{bmatrix} 1.986 \\ 7.826 \end{bmatrix}$

$x_1 = 1.986$

$x_2 = 7.871$

(c) x_0가 더 가깝다

(d) $f(x_1) = 0.486$

$f(x_0) = 18.5$

$f(x_1)$이 더 작다

Problem 2

Gradient Descent Algorithm

- 순서

1. gradient를 initialize를 해준다
2. 2-norm gradient가 tolerance에 보다 큰 경우 아래의 경우를 수행한다.
3. step size는 0.1로 고정해준다
4. x를 아래 수식에 따라 update 해준다.

$$x_{k+1} = x_k - t_k * \nabla f(x_k)$$

2~5를 반복한다.

- sudo code

initialize x

initialize gradient

while $\|\nabla f(x_k)\| > tol$ do

$t_k = 0.1$

$x_{k+1} = x_k - t_k * \nabla f(x_k)$

k = k+1

end while

- Calculate gradient

gradient 계산식은 다음과 같다.

$$\nabla f(x_k) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 3x_1 + x_2 - 4 \\ x_1 + 2 * x_2 + 2 \end{bmatrix}$$

로 표현할 수 있다.

하지만 $x^T A x > 0$ 크고, $\mathrm{x} \neq 0$ 일 경우

주어진 수식은,

$$f(x) = \frac{1}{2}x^T Ax - b^T x$$

임으로 gradient는 다음과 같이 나타낼 수 있다.

$$\nabla f(x_k) = Ax - b$$

- Code

```
void Gradient_Descent() {

    int num = 0;
    printf("\n\nGradient Method.......\n");

    Gradient(init_hw4);

    pre_x[0] = init_hw4[0];
    pre_x[1] = init_hw4[1];

    while (NORM2(grad[0], grad[1]) >= TOL_ERROR) {
        Next_GD(pre_x, tk);
        Gradient(pre_x);
        output_grad[num][0] = ans_x[0];
        output_grad[num][1] = ans_x[1];
        printf("%2d %3.2f %3.2f\n", num+1, output_grad[num][0], output_grad[num][1]);
        num++;
    }

}
```

Fig. Gradient Descent

sudo code에 따라서, 먼저 gradient와 pre_x에 initialize 해준다. while문에 따라서, gradient 2norm이 TOL_ERROR보다 클 경우에 프로그램이 돌아간다. 프로그램 안에서 수식에 따른, 다음 ans_x를 계산한다. 그 이후, gradient를 새로 계산한다. 그리고 output_grad에 계산된 ans_x를 update 해준다.

```
void Gradient(double input[no_hw4]){
    double x = input[0];
    double y = input[1];

    grad[0] = 3 * x + y - 4;
    grad[1] = x + 2 * y + 2;
}
```

Fig. Gradient

gradient를 계산하는 함수는 다음과 같다. input 값을 풀어 쓴 gradient 수식에 넣었다.

```
double NORM2(double x1, double x2) {
    double temp = pow(x1, 2) + pow(x2, 2);
    return sqrt(temp);
}
```

Fig. 2-norm

2-norm를 계산하는 함수를 다음과 같이 만들었다.

```
void Next_GD(double input[no_hw4], double tk) {
    ans_x[0] = input[0] - tk * grad[0];
    ans_x[1] = input[1] - tk * grad[1];

    pre_x[0] = ans_x[0];
    pre_x[1] = ans_x[1];
}
```

Fig. Next_GD

다음 gradient를 계산하고, update하는 함수이다. 수식에 따라 계산해주고 함수내에서 x를 update 해준다.

steepest gradient model

- 순서

5. gradient를 initialize를 해준다
6. 2-norm gradient가 tolerance에 보다 큰 경우 아래의 경우를 수행한다.
7. step size는 update 해준다.
8. x를 아래 수식에 따라 update 해준다.

$$x_{k+1} = x_k - t_k * \nabla f(x_k)$$

2~5를 반복한다.

- sudo code

initialize x

initialize gradient

while $\|\nabla f(x_k)\| > tol$ do

update t_k

$x_{k+1} = x_k - t_k * \nabla f(x_k)$

k = k+1

end while

- Calculate gradient

gradient 계산식은 다음과 같다.

$$\nabla f(x_k) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 3x_1 + x_2 - 4 \\ x_1 + 2 * x_2 + 2 \end{bmatrix}$$

로 표현할 수 있다.

하지만 $x^T A x > 0$ 크고, $\mathrm{x} \neq 0$ 일 경우

주어진 수식은,

$$f(x) = \frac{1}{2}x^T A x - b^T x$$

임으로 gradient는 다음과 같이 나타낼 수 있다.

$$\nabla f(x_k) = Ax - b$$

- Update Step size

step size를 다음과 같이 표현할 수 있다.

$$t_k = argmin\, f(x_k - t_k * \nabla f(x_k))$$

$$t_k = \frac{\nabla f(x_k)^T * \nabla f(x_k)}{\nabla f(x_k)^T * A * \nabla f(x_k)}$$

이다.

- Code

```
void Steepest_Gradient_Descent() {

    int num = 0;
    printf("\n\nSteepest Method.......\n");

    Gradient(init_hw4);
    tk = FindStepsize(grad);

    pre_x[0] = init_hw4[0];
    pre_x[1] = init_hw4[1];

    while (NORM2(grad[0], grad[1]) >= TOL_ERROR) {
        Next_GD(pre_x, tk);
        Gradient(pre_x);
        tk = FindStepsize(grad);
        output_steep[num][0] = ans_x[0];
        output_steep[num][1] = ans_x[1];
        printf("%2d %3.2f %3.2f\n", num + 1, output_steep[num][0], output_steep[num][1]);
        num++;
    }

}
```

Fig. steepest gradient descent

gradient descent와 다른 점은 step size를 initialize 해주고 update해주는 거 외에는 gradient와 동일하다. 따라서, 함수의 flow는 동일하다.

```
double FindStepsize(double input[no_hw4]) {
    double temp[no_hw4] = { 0, };
    double den = 0, num = 0;

    num = grad[0] * grad[0] + grad[1] * grad[1];

    for (int i = 0; i < no_hw4; i++) {
        for (int j = 0; j < no_hw4; j++) {
            temp[i] = temp[i] + grad[j] * A[i][j];
        }
        den = den + temp[i] * grad[i];
    }


    double output = num / den;

    return output;
}
```

Fig. Find Step size

step size를 update 해주는 함수이다. 수식의 분모와 분자를 각각 계산해준다. return해준다.

두 model의 output을 plot 해보면 steepest descent model은 18까지 iteration이 가고 descent model은 90까지 간다. 가장 효율적인 step size를 정해주는 steepest descent model의 효율이 더 좋다.

Compare Output

```
/******* HW4 *******/

Gradient Method.......
 1 -1.10 0.80
 2 -0.45 0.55
 3 0.03 0.29
 4 0.39 0.03
 5 0.67 -0.22
 6 0.89 -0.44
 7 1.07 -0.64
 8 1.21 -0.82
 9 1.33 -0.98
10 1.43 -1.12
11 1.51 -1.24
12 1.58 -1.34
13 1.64 -1.43
14 1.69 -1.51
15 1.74 -1.58
16 1.77 -1.63
17 1.80 -1.68
18 1.83 -1.73
19 1.85 -1.77
20 1.87 -1.80
21 1.89 -1.83
22 1.91 -1.85
23 1.92 -1.87
24 1.93 -1.89
25 1.94 -1.90
26 1.95 -1.92
27 1.96 -1.93
28 1.96 -1.94
29 1.97 -1.95
30 1.97 -1.95
31 1.98 -1.96
32 1.98 -1.97
33 1.98 -1.97
34 1.98 -1.97
35 1.99 -1.98
36 1.99 -1.98
37 1.99 -1.98
38 1.99 -1.99
39 1.99 -1.99
40 1.99 -1.99
41 1.99 -1.99
42 2.00 -1.99
43 2.00 -1.99
44 2.00 -1.99
45 2.00 -2.00
```

```
46 2.00 -2.00
47 2.00 -2.00
48 2.00 -2.00
49 2.00 -2.00
50 2.00 -2.00
51 2.00 -2.00
52 2.00 -2.00
53 2.00 -2.00
54 2.00 -2.00
55 2.00 -2.00
56 2.00 -2.00
57 2.00 -2.00
58 2.00 -2.00
59 2.00 -2.00
60 2.00 -2.00
61 2.00 -2.00
62 2.00 -2.00
63 2.00 -2.00
64 2.00 -2.00
65 2.00 -2.00
66 2.00 -2.00
67 2.00 -2.00
68 2.00 -2.00
69 2.00 -2.00
70 2.00 -2.00
71 2.00 -2.00
72 2.00 -2.00
73 2.00 -2.00
74 2.00 -2.00
75 2.00 -2.00
76 2.00 -2.00
77 2.00 -2.00
78 2.00 -2.00
79 2.00 -2.00
80 2.00 -2.00
81 2.00 -2.00
82 2.00 -2.00
83 2.00 -2.00
84 2.00 -2.00
85 2.00 -2.00
86 2.00 -2.00
87 2.00 -2.00
88 2.00 -2.00
89 2.00 -2.00
90 2.00 -2.00
```

Fig. output of descent model

```
Steepest Method.......
 1 1.56 0.21
 2 1.20 -1.40
 3 1.91 -1.56
 4 1.84 -1.88
 5 1.98 -1.91
 6 1.97 -1.98
 7 2.00 -1.98
 8 1.99 -2.00
 9 2.00 -2.00
10 2.00 -2.00
11 2.00 -2.00
12 2.00 -2.00
13 2.00 -2.00
14 2.00 -2.00
15 2.00 -2.00
16 2.00 -2.00
17 2.00 -2.00
18 2.00 -2.00
계속하려면 아무 키나 누르십시오 . . .
```

Fig. output of steepest descent model

두 model의 output을 plot 해보면 steepest descent model은 18까지 iteration이 가고 descent model은 90까지 간다. 가장 효율적인 step size를 정해주는 steepest descent model의 효율이 더 좋다.