# 수치해석 과제#4

**2015111113 김준기**

## 6.3(c)

Newton_Rapson.m

```
1   function result = Newton_Rapson(func, dfunc, xi, es, maxit)
2       % func =  x^3 - 6x^2 + 11x - 6.1
3
4       iteration = 0;
5       display();
6       while(1)
7
8           xi_1 = xi - func(xi)/dfunc(xi);
9
10          ea = abs((xi_1 - xi)/xi_1)*100;
11
12          xi = xi_1;
13          iteration = iteration + 1;
14
15          display(iteration, xi, ea);
16
17          % loop문 탈출 조건
18          if ea <= es, break, end
19          if iteration >= maxit, break, end
20      end
21      disp(xi);
22      result_2 = roots([1 -6 11 -6.1]);
23      display_answer(xi, result_2);
24
25      result = xi;
26
27
28
29  end
30  function display_answer(result1, result2)
31      fprintf("root by using Newton-Rapson method : %f #n", result1);
32      fprintf("root by buil-in root function : %f #n", result2);
33  end
34
```

실질적으로 Newton-Rapson method를 이용하여 계산하는 부분입니다.

```
35      function display(number, x1, x2)
36 -    if nargin==0
37 -        disp("|------|-----------------------------|-----------------------------|");
38 -        disp("|number|            x_i              | approximation relative error |");
39 -        disp("|------|-----------------------------|-----------------------------|");
40
41 -    else
42 -        fprintf("|%3d   |%20.10f         |      %15.10f         |\n", number, x1, x2);
43 -        disp("|------|-----------------------------|-----------------------------|");
44 -    end
45
46 -    end
```

단순히 보기 좋게 출력만 담당하는 함수입니다. 인수의 개수가 0개이면, 도표의 머리 부분과 틀만 출력합니다.

명령 창

```
>> Newton_Rapson(@(x)x^3 - 6*x^2 + 11*x - 6.1, @(x) 3*x^2 - 12*x + 11, 3.5, 0.001, 100 )
|------|-----------------------------|-----------------------------|
|number|            x_i              | approximation relative error |
|------|-----------------------------|-----------------------------|
|  1   |        3.1913043478         |         9.6730245232        |
|------|-----------------------------|-----------------------------|
|  2   |        3.0686988211         |         3.9953587472        |
|------|-----------------------------|-----------------------------|
|  3   |        3.0473167369         |         0.7016692386        |
|------|-----------------------------|-----------------------------|
|  4   |        3.0466810869         |         0.0208636877        |
|------|-----------------------------|-----------------------------|
|  5   |        3.0466805318         |         0.0000182191        |
|------|-----------------------------|-----------------------------|
    3.0467

root by using Newton-Rapson method : 3.046681
root by buil-in root function : 3.046681
root by buil-in root function : 1.898969
root by buil-in root function : 1.054351

ans =

    3.0467

fx >>
```

뉴턴-랩슨 방법을 이용한 경우 해를 하나밖에 못 찾았지만, roots( ) 매트랩 내장 함수를 사용한 경우 3개의 해를 찾았습니다. 이는 3차 방정식이 최대 3개의 해가 존재할 수 있는데, 이 경우 그 사실에 부합합니다.

## 6.12

Newton_Rapson.m

```
function result = Newton_Rapson(func, dfunc, xi, es, maxit)
    % func =  x^3 - 6x^2 + 11x - 6.1

    iteration = 0;
    display();
    while(1)

        xi_1 = xi - func(xi)/dfunc(xi);

        ea = abs((xi_1 - xi)/xi_1)*100;

        xi = xi_1;
        iteration = iteration + 1;

        display(iteration, xi, ea);

        % loop문 탈출 조건
        if ea <= es, break, end
        if iteration >= maxit, break, end
    end
    disp(xi);
    result_2 = roots([0.0074 -0.284 3.355 -12.183 5]);
    display_answer(xi, result_2);

    result = xi;



end
function display_answer(result1, result2)
    fprintf("root by using Newton-Rapson method : %f #n", result1);
    fprintf("root by buil-in root function : %f #n", result2);
end

```

```
35      function display(number, x1, x2)
36 -    if nargin==0
37 -        disp("|------|------------------------------|------------------------------|");
38 -        disp("|number|            x_i               | approximation relative error |");
39 -        disp("|------|------------------------------|------------------------------|");
40
41 -    else
42 -        fprintf("|%3d   |%20.10f          |      %15.10f          |\n", number, x1, x2);
43 -        disp("|------|------------------------------|------------------------------|");
44 -    end
45
46 -    end
```

명령 창

```
>> Newton_Rapson(@(x)0.0074*x^4 - 0.284*x^3 + 3.355*x^2 - 12.183*x + 5, @(x)(0.0074*4)*x^3 - (0.284*3)*x^2 + (3.355*2)*x - 12.183, 16.15, 0.001, 100 )
|------|------------------------------|------------------------------|
|number|            x_i               | approximation relative error |
|------|------------------------------|------------------------------|
|  1   |         9.0771024166         |         77.9202135080        |
|------|------------------------------|------------------------------|
|  2   |        -4.0210096030         |        325.7418736284        |
|------|------------------------------|------------------------------|
|  3   |        -1.6764510805         |        139.8524865867        |
|------|------------------------------|------------------------------|
|  4   |        -0.2803959330         |        497.8870886340        |
|------|------------------------------|------------------------------|
|  5   |         0.3342437645         |        183.8896526473        |
|------|------------------------------|------------------------------|
|  6   |         0.4630232579         |         27.8127483250        |
|------|------------------------------|------------------------------|
|  7   |         0.4684701966         |          1.1627076427        |
|------|------------------------------|------------------------------|
|  8   |         0.4684797455         |          0.0020382746        |
|------|------------------------------|------------------------------|
|  9   |         0.4684797456         |          0.0000000063        |
|------|------------------------------|------------------------------|
    0.4685

root by using Newton-Rapson method : 0.468480
root by buil-in root function : 18.894766
root by buil-in root function : 13.257491
root by buil-in root function : 5.757641
root by buil-in root function : 0.468480

ans =

    0.4685

fx >>
```

뉴턴-랩슨 방법을 이용한 경우 해를 하나밖에 못 찾았지만, roots( ) 매트랩 내장 함수를 사용한 경우 4개의 해를 찾았습니다. 이는 4차 방정식이 최대 4개의 해가 존재할 수 있는데, 이 경우 그 사실에 부합합니다.

## 6.19

Newton_Rapson.m | impedence.m

```
1   function impedence(bracket)
2       % 저항, 인덕터, 커패시터가 병렬로 연결된 회로에서
3       % 각 주파수(w)를 구하는 함수 입니다.
4       % R, L, C값은 이미 주어져 있습니다.
5 –     R = 225;
6 –     L = 0.5;
7 –     C = 0.6 * 10^-6;
8
9 –     func = @(w) sqrt(R^-2 + (w*C - 1/(w*L))^2) - 0.01;
10
11 –    [w_result, fw] = fzero(func, bracket);
12
13 –    disp(w_result);
14 –    disp(fw);
15
16 –  end
```

임피던스(Z) 값이 100옴일 때, 각 주파수 W를 구하는 문제로 이는 왼쪽과 같은 방정식이 0 일때의 해 w를 찾는 문제로 생각할 수 있습니다.

명령 창

```
>> impedence([1 1000])
  220.0202

   1.7347e-18

fx >>
```

각 주파수(w)의 값은 220.02로 그 때의 함수 값이 0에 거의 근접하는 값이 나옴을 확인하였습니다.

# 7.11

Newton_Rapson.m | impedence.m | optimizationfunc.m

```
function optimizationfunc(bracket, func, maxit, es)
    drowfunc();
    [x_min, f_val] = fminbnd(func, bracket(1), bracket(2));
    disp(x_min);
    disp(f_val);
    golden_section_search(bracket, func, maxit, es);
end
```

7.11(a)을 위한 그래프 그리기

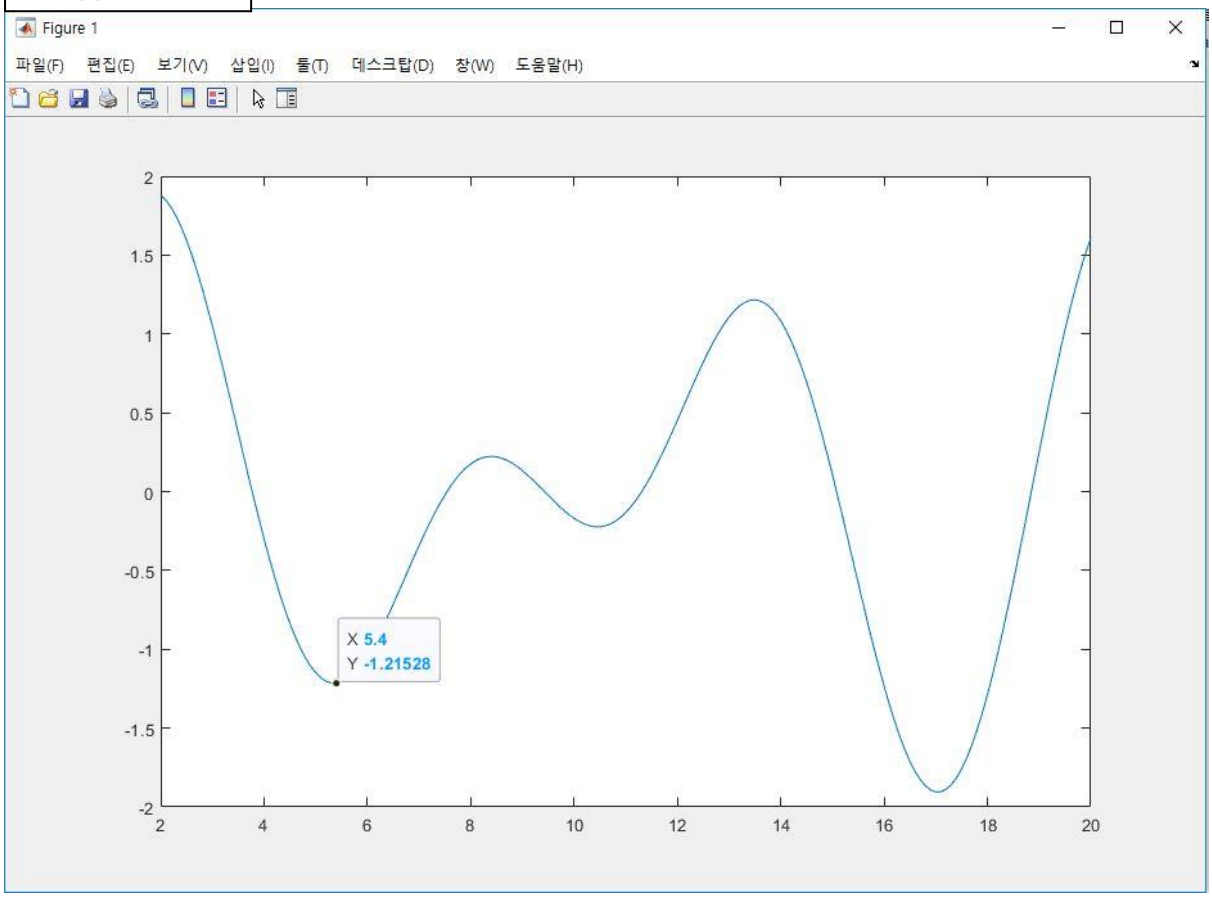
7.11(b)를 위한 fminbnd함수를 이용하여 최소값 찾기

7.11(c)에서 golden-section search를 이용하여 최소값 찾기

```
function golden_section_search(bracket, func, maxit, es)
    xl = bracket(1);
    xu = bracket(2);
    phi = (1+sqrt(5))/2;
    d = (phi - 1)*(xu - xl);

    x1 = xl + d;
    x2 = xu - d;
    iteration = 0;

    while(1)
        if func(x2) > func(x1)
            xopt = x1;
            xl = x2;
            x2 = x1;
            d = (phi - 1)*(xu - xl);
            x1 = xl + d;
        else
            xopt = x2;
            xu = x1;
            x1 = x2;
            d = (phi - 1)*(xu - xl);
            x2 = xu - d;
        end
        iteration = iteration + 1;
        ea = (2 - phi)*abs((xu - xl)/xopt) * 100;
        if iteration >= maxit | ea <= es, break, end
    end

    fprintf("minimum x point by golden section search : %f₩n", xopt);

end
```

```
45      function drowfunc()
46 -        x = 2:0.1:20;
47 -        func = sin(x) + sin((2*x)/3);
48 -        plot(x, func);
49 -    end
```

7.11(a)의 결과




7.11(b)와 (c)의 결과

```
명령 창
>> optimizationfunc([4 8], @(x)sin(x) + sin((2*x)/3), 100, 0.01)
    5.3622
   -1.2160

minimum x point by golden section search : 5.362319
>>
```

fminbnd( ) built-in 매트랩 함수를 사용

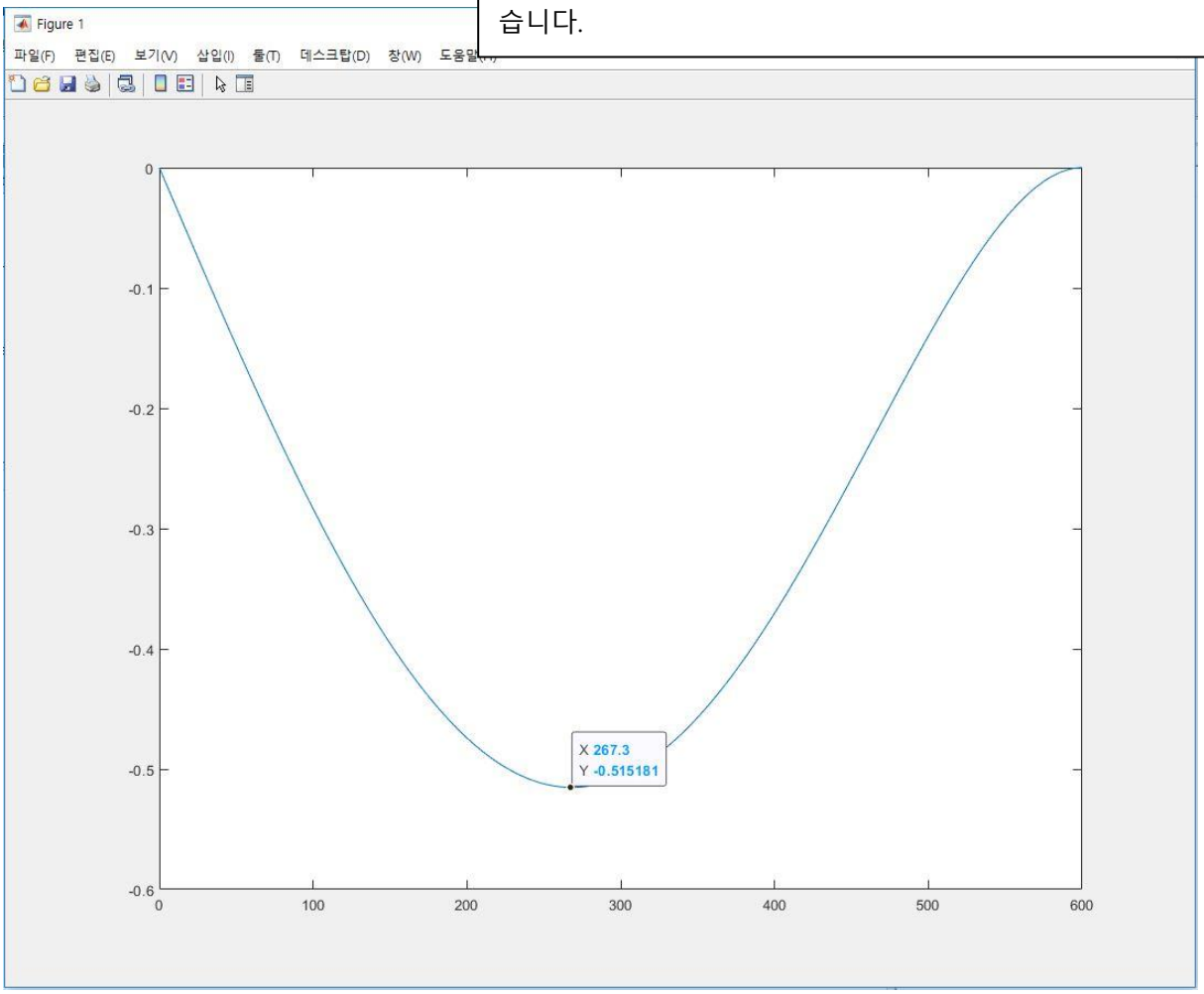
golden-section search method를 사용하여 최소값을 구한 결과

# 7.22(a)

Newton_Rapson.m | impedence.m | optimizationfunc.m | deflection.m

```
1   function deflection(bracket, maxit, es)
2 –     L = 600;
3 –     E = 50000;
4 –     I = 30000;
5 –     w = 2.5;
6
7 –     xi = 0 : 0.1 : 600;
8
9 –     func = @(x)w/(120 * E * I * L) *(-x^5 + 2 * L^2 * x^3 - L^4 * x);
10 –    f = w/(120 * E * I * L) *(-xi.^5 + 2 * L^2 * xi.^3 - L^4 * xi);
11
12 –    plot(xi, f);
13
14 –    golden_section_search(bracket, func, maxit, es);
15 –  end
16
17
```

anonymous function을 출력하려면 f(x)형태로 독립 변수를 명시해줘야 한다는 것을 공부하면서 알게 되었는데, 이 때는 왜 오류가 일어나는지 몰라서 별도로 함수를 정의하여 출력하였습니다.



## 7.22(b)

```
18    function golden_section_search(bracket, func, maxit, es)
19 -      xl = bracket(1);
20 -      xu = bracket(2);
21 -      phi = (1+sqrt(5))/2;
22 -      d = (phi - 1)*(xu - xl);
23
24 -      x1 = xl + d;
25 -      x2 = xu - d;
26 -      iteration = 0;
27
28 -      while(1)
29 -          if func(x2) > func(x1)
30 -              xopt = x1;
31 -              xl = x2;
32 -              x2 = x1;
33 -              d = (phi - 1)*(xu - xl);
34 -              x1 = xl + d;
35 -          else
36 -              xopt = x2;
37 -              xu = x1;
38 -              x1 = x2;
39 -              d = (phi - 1)*(xu - xl);
40 -              x2 = xu - d;
41 -          end
42 -          iteration = iteration + 1;
43 -          ea = (2 - phi)*abs((xu - xl)/xopt) * 100;
44 -          if iteration >= maxit | ea <= es, break, end
45 -      end
46
47 -      fprintf("minimum x point by golden section search : %f₩n", xopt);
48
49 -  end
```

명령 창

```
>> deflection([0 600], 100, 0.01)
minimum x point by golden section search : 268.334933
>>
```

# 7.33

편집기 - C:₩Users₩JunGi_Kim₩Desktop₩2020학년 3학년 2학기₩수치해석₩과제₩네번째₩charge.m

impedence.m | optimizationfunc.m | deflection.m | charge.m

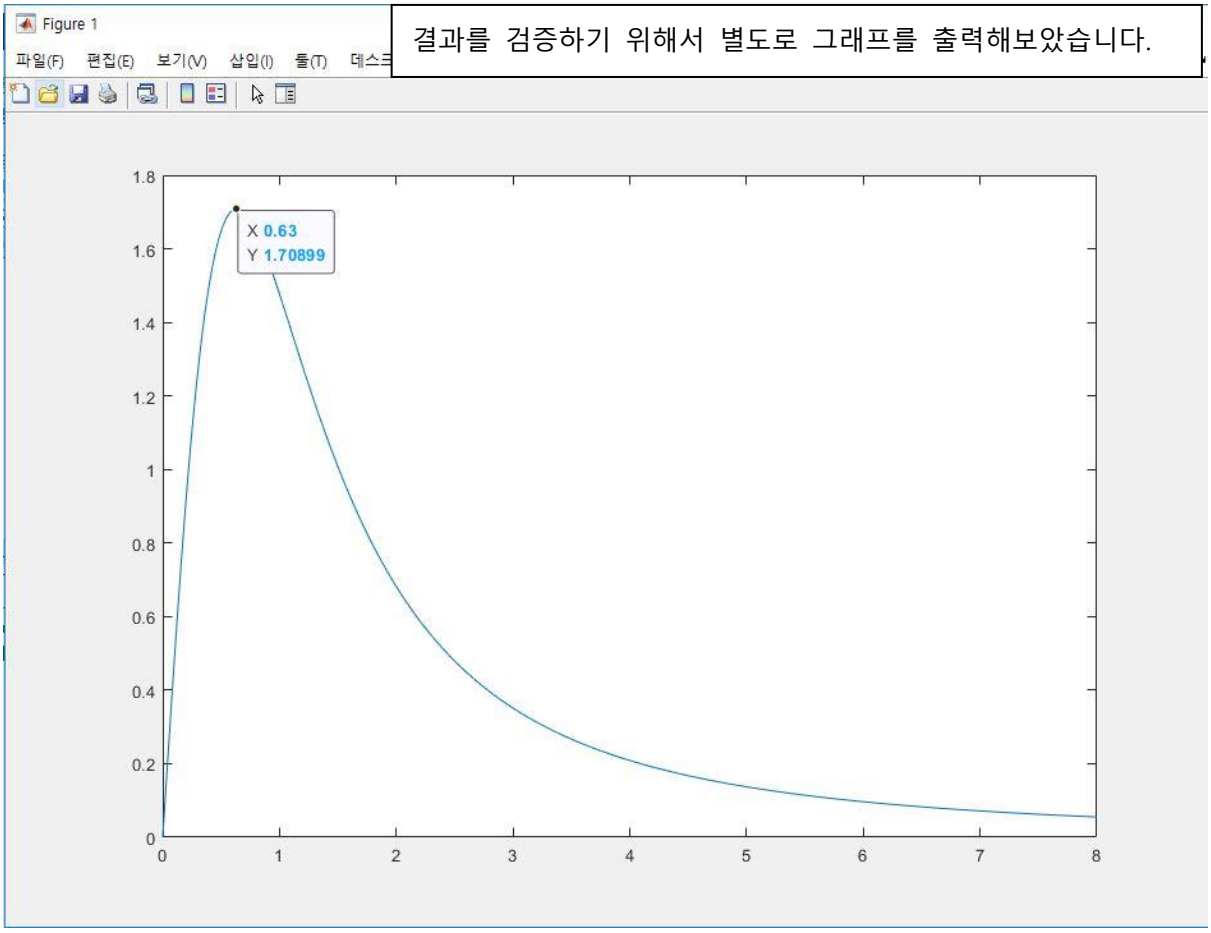
```
1    function charge()
2 -      e0 = 8.85 * 10^-12;
3 -      Q = 2 * 10^-5;
4 -      a = 0.9;
5
6 -      x = 0 : 0.01 : 8;
7 -      func = @(x) -1/(4 * pi * e0) * (Q * Q * x)./((x.^2 + a^2).^1.5);
8
9
10 -     plot(x, -func(x));
11
12 -     [xmin, fval] = fminbnd(func, 0, 8);
13 -     fprintf("F(x) = %f when minimu x value is %f₩n", -fval, xmin);
14       % 원래 함수의 그래프는 최대값을 찾는 문제이므로
15       % 최소값을 찾는 fminbnd bulit-in 함수를 사용하였고
16       % 그 때의 값은 - 를 곱해야함.
17
18   end
```

문제에 주어진 함수에 대해 최소값을 찾는 fminbnd함수를 사용하기 위해 원래 함수값에 마이너스를 곱했습니다.

명령 창

```
>> charge()
F(x) = 1.709110 when minimu x value is 0.636382
>>
```

결과를 검증하기 위해서 별도로 그래프를 출력해보았습니다.



수치 해석 과제 HW#4. 2015111113 김 준 기.

6.3.(b).

$f(x) = x^3 - 6x^2 + 11x - 6.1$

three iterations, $x_0 = 3.5$.

by using Newton-Rapson method,

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

$f'(x) = 3x^2 - 12x + 11$.

• First iteration.

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

$$= 3.5 - \frac{(3.5)^3 - 6\times(3.5)^2 + 11\times 3.5 - 6.1}{3\times(3.5)^2 - 12\times 3.5 + 11}$$

$$= 3.1913$$

• second iteration.

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

$$= 3.1913 - \frac{(3.1913)^3 - 6\times(3.1913)^2 + 11\times 3.1913 - 6.1}{3\times(3.1913)^2 - 12\times 3.1913 + 11}$$

$$= 3.0687.$$

• third iteration.

$$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)}$$

$$= 3.0687 - \frac{(3.0687)^3 - 6\times(3.0687)^2 + 11\times 3.0687 - 6.1}{3\times(3.0687)^2 - 12\times 3.0687 + 11}$$

$$= 3.0413.$$