**REPORT**



|  |  |
| --- | --- |
| **과목명** | 수치해석 |
| **학과** | 컴퓨터 공학부 |
| **학번** | 201911247 |
| **이름** | 남권표 |
| **제출일자** | 2020.04.19 |

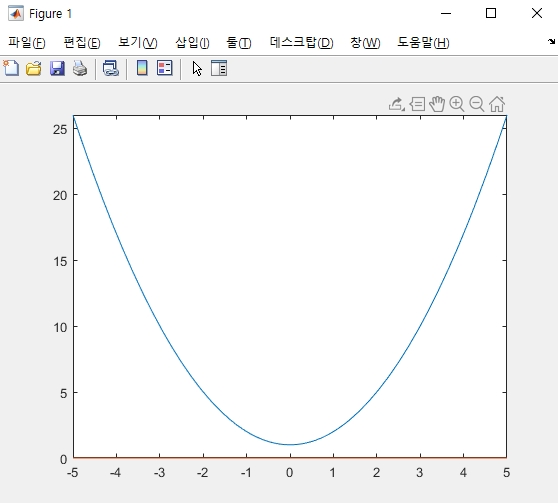
1. **계수가 실수인 이차 방정식의 근을 구하는 프로그램을 작성하시오.**
2. **문제 설명**

우리는 이차 방정식의 근을 ‘함수’를 통하여 구해야 한다. 즉, 함수 안에서의 이차 방정식을 정의는 필수이므로 매개변수를 통해 a, b, c를 입력 받고 ax^2+bx+c라는 기본 형식을 응용하기로 한다.

또한 결과를 출력할 때 실근, 중근, 허근을 구분하여 출력해야 하므로 이는 판별식 b^2-2ac를 통해 if\_else문을 이용하여 출력하기로 한다.

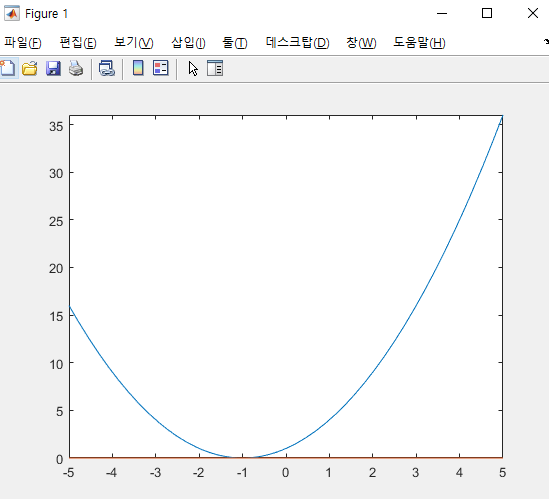
그러면 이차 방정식의 정의하고 근은 어떻게 구할까? 근을 구하는 방식에는 여러 방식이 존재하지만 해당 함수에서는 다항식의 근을 구해주는 matlab 내장함수인 roots()를 사용하기로 한다.

1. **그래프**

**[근이 허근일 때의 그래프]**

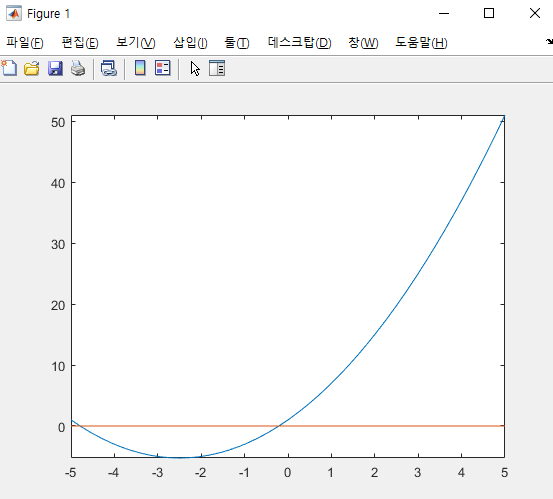
[root] = quadratic\_equation(1, 0, 1)로 호출

* x^2+1 그래프

**[근이 중근일 때의 그래프]**

[root] = quadratic\_equation(1, 2, 1)로 호출

* x^2+2\*x+1 그래프

**[근이 실근일 때의 그래프]**

[root] = quadratic\_equation(1, 5, 1)로 호출

* x^2+5\*x+1 그래프

1. **실행결과 및 분석**

**[1] 소스코드**

**<메인문>(이름: main.m) (case마다 parameter의 값은 바뀜. 그래프 참고)**

[root] = quadratic\_equation(1, 0, 1)

**<함수 quadratic\_equation.m>**

function [root] = quadratic\_equation(a, b, c)

f = @(x) a\*x.^2+b\*x+c;

fplot(f)

hold on

fplot(@(x) x\*0)

if b^2-4\*a\*c < 0

root = roots([a, b, c]);

disp('허근')

elseif b^2-4\*a\*c == 0

root = roots([a, b, c]);

disp('중근')

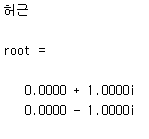
else

root = roots([a, b, c]);

disp('실근')

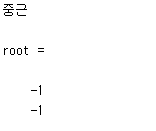
end

**[2] x^2+1의 근(case. 허근)**



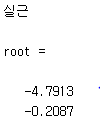
\* 분석: x^2+1의 판별식은 0-4\*1\*1=-4로 음의 값을 가진다. 따라서 해당 식은 허근을 가질 것이라 짐작할 수 있다. 짐작과 같이 근은 허근의 형식을 띈다.

**[3] x^2+2\*x+1의 근(case. 중근)**



\* 분석: x^2+2\*x+1의 판별식은 4-4\*1\*1=0으로 0이라는 값을 가진다. 따라서 해당 식은 중근을 가질 것이라 짐작할 수 있다. 짐작과 같이 근은 중근의 형식을 띈다.

**[4] x^2+5\*x+1의 근(case. 실근)**



\* 분석: x^2+5\*x+1의 판별식은 25-4\*1\*1=21로 양의 값을 가진다. 따라서 해당 식은 실근을 가질 것이라 짐작할 수 있다. 짐작과 같이 근은 허근의 형식을 띈다.

1. **두 점을 지나는 직선을 두 개가 주어지면, 두 직선의 교점이 존재할 경우, 교점을 출력하시오.**
2. **문제 설명**

두 점을 지나는 직선 두 개를 구하기 위해서 가장 먼저 할 일은 각 직선마다 두 개의 점을 입력 받는 것이다. 따라서 input을 활용하여 두 개의 점을 입력 받고 기울기와 y절편을 구하도록 한다. (첫 번째 직선의 기울기와 y절편은 a1과 y\_value1, 두 번째 직선의 기울기와 y절편은 a2와 y\_value2이다.)

이후 두 직선의 교점을 구하기 위해서는 두 직선을 빼서 나오는 방정식의 해를 구해야 하는데 이것이 교점의 x좌표 값이 되기 때문이다. 따라서 방금 구해준 [a1, y\_value1], [a2, y\_value2]를 활용하면 두 직선을 빼 준 방정식은 (a1-a2)\*x+y\_value1-y\_value2의 형태를 띈다는 사실을 알 수 있다.

이 방정식의 근은 roots()함수를 이용하여 구할 수 있고, 이를 통해 구해준 x값을 이용하여 feval()함수를 통해 교점의 y값까지 구할 수 있다.

**(2) 그래프, 실행결과 및 분석**

**[1] 소스코드**

**<메인문>(이름: main2.m)**

disp('직선1')

point1 = input('첫 번째 점을 입력해주세요([x, y]처럼 []를 씌워주세요!) : ');

point2 = input('두 번째 점을 입력해주세요([x, y]처럼 []를 씌워주세요!) : ');

[a1, y\_value1] = funcElement(point1, point2);

disp('직선2')

point1 = input('첫 번째 점을 입력해주세요([x, y]처럼 []를 씌워주세요!) : ');

point2 = input('두 번째 점을 입력해주세요([x, y]처럼 []를 씌워주세요!) : ');

[a2, y\_value2] = funcElement(point1, point2);

if a1 ~= a2

[x, fval] = intersection(a1, y\_value1, a2, y\_value2);

fprintf('교점: (%f, %f)\n', x, fval)

elseif (a1 == a2) && (y\_value1 ~= y\_value2)

draw\_graph(a1, y\_value1, a2, y\_value2);

disp('교점이 존재하지 않습니다.')

else

draw\_graph(a1, y\_value1, a2, y\_value2);

disp('동일한 직선입니다.')

end

**<funcElement.m>(직선의 방정식의 기울기와 y절편을 구해주는 함수)**

function [a, y\_value]=funcElement(point1, point2)

a = (point1(2)-point2(2))/(point1(1)-point2(1));

y\_value = ((point1(2)-point2(2))/(point1(1)-point2(1)))\*((-1)\*point1(1))+point1(2);

**<intersection.m>(두 직선의 그래프를 그려주고, 교점을 구해주는 함수)**

function [x, fval] = intersection(a1, y\_value1, a2, y\_value2)

f1 = @(x) a1\*x+y\_value1;

f2 = @(x) a2\*x+y\_value2;

fplot(f1, 'k')

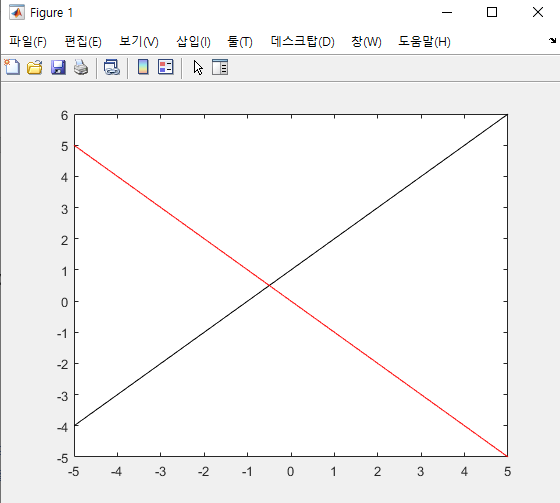
hold on

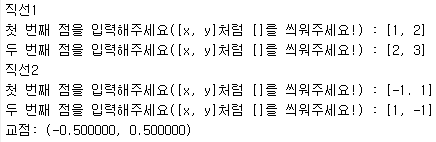
fplot(f2, 'r')

x = roots([a1-a2, y\_value1-y\_value2]);

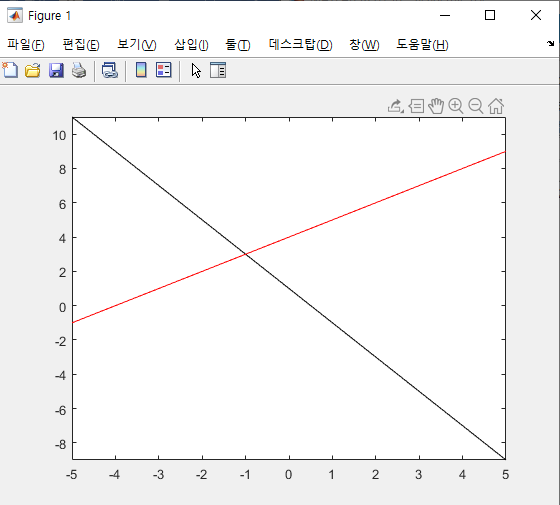
fval = feval(f1, x);

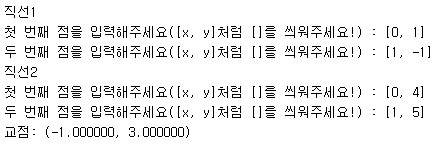
**[2] x+1과 -x의 교점 결과**

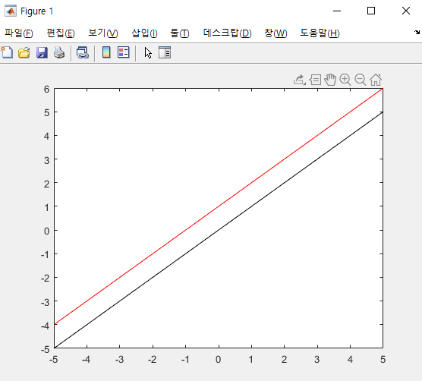
 왼쪽의 그래프는 x+1과 -x의 교점을 보여주는 그래프이다. 첫 번째 직선의 두 점을 입력 받으면 a1=(2-3)/(1-2)=1, y\_value1=1인 것을 구할 수 있고, 두 번째 직선의 경우에는 a2=(-1-1)/(1-(-1))=-1, y\_value2=0인 사실을 알 수 있다. 두 직선의 교점은 (-0.5, 0.5)임으로 아래의 결과가 맞는 결과임을 확인할 수 있다.



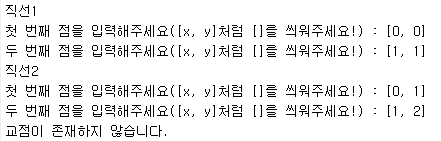
**[3] -2x+1과 x+4의 교점 결과**

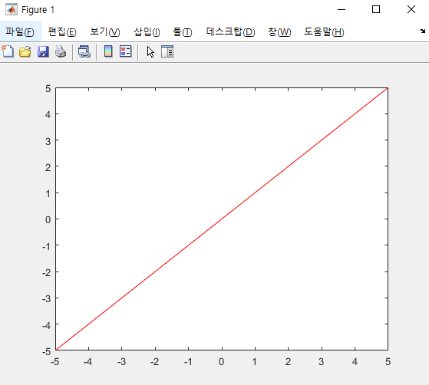
 왼쪽의 그래프는 x+1과 -x의 교점을 보여주는 그래프이다. 첫 번째 직선의 두 점을 입력 받으면 a1=(-1-1)/(1-0)=-2, y\_value1=1인 것을 구할 수 있고, 두 번째 직선의 경우에는 a2=(5-4)/(1-0)=1, y\_value2=4인 사실을 알 수 있다. 두 직선의 교점은 (-1, 3)임으로 아래의 결과가 맞는 결과임을 확인할 수 있다.



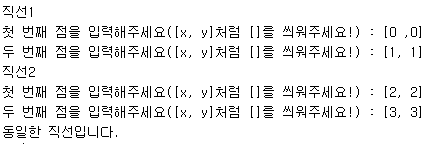
 **[4] x와 x+1의 교점 결과(기울기가 같은 경우)**

아래처럼 기울기를 동일하게 주는 경우에는 교점이 존재하지 않는다. 따라서 ‘교점이 존재하지 않습니다.’라는 문구가 뜨도록 하였다.



**[5] x와 x의 교점 결과(동일한 직선인 경우)**

아래처럼 동일한 직선을 주는 경우에는 ‘동일한 직선입니다’라는 문구가 뜨도록 하였다.

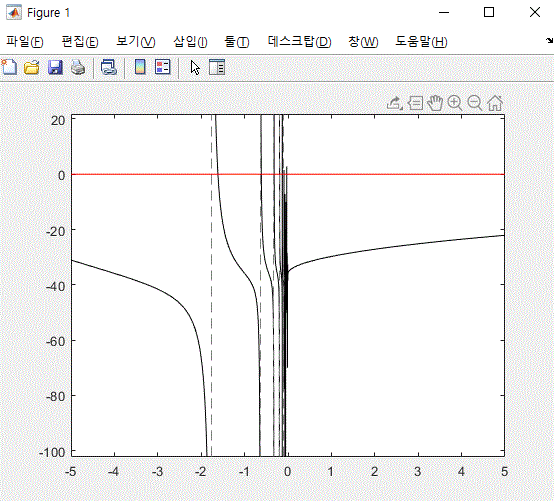


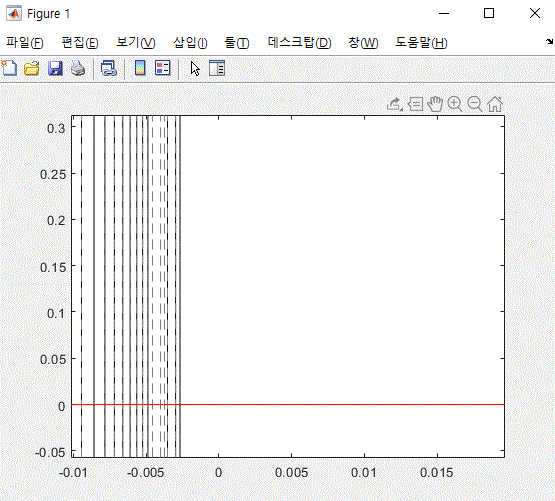
1. **이분법, 가상위치법, 고정점 반복법, 뉴튼-랩슨법, 할선법을 이용한 근 구하기**
2. **문제 설명**

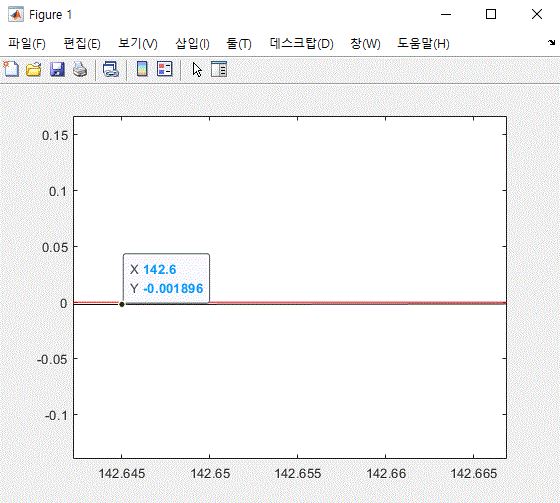
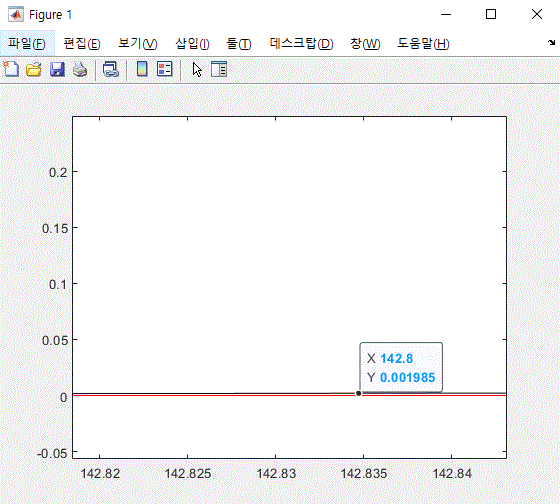
우리는 f(m)의 근을 구하면 된다. 단, 이때 생각해야할 것은 ‘m은 음수가 될 수 없다.’라는 것이다. 왜냐하면 질량이 음의 값을 가질 수는 없기 때문이다. 따라서 구간법과 개간법을 통하여 근을 구할 때는 m값이 양인 구간에서 구간과 초기값을 설정해주면 된다.

구간법의 경우에는 y값이 변하는 구간을 찾아 그 구간을 대입해주면 될 것이다. 개간법의 경우 근과 가까운 하나의 초기값을 대입해 주면 될 것이다. 위와 같은 내용을 토대로 이분법, 가위치법, 수정 가위치법, 고정점 반복법, 뉴튼-랩슨법, 할선법을 활용하면 될 것이다.

1. **그래프[고정점 반복법의 g(x)함수는 (3)-고정점 반복법에서 분석]**

왼쪽이 문제에 주어진 함수를 그린 그래프이다. 해당 그래프를 보면 근이 무수히 많은 것을 확인할 수 있다. 그렇다면 함수의 근이 많은 곳을 확대해 보도록 하자.

왼쪽과 같이 확대를 해보면 근이 몰린 곳은 m이 0보다 작을 때이다. 따라서 우리는 0보다 큰 구간에서 y값의 부호가 변하는 부분을 찾아 구간을 정해주거나 근과 가까운 값을 지정하여 구간법이나 개간법을 활용해주면 된다.

위의 그래프 확대 부분은 각각 x=142.8, x=142.6일 때를 확인해본 것이다. 위와 같이 x=142.8일 때 함수는 양의 값을 가지고, x=142.6일 때는 음의 값을 가진다*. 따라서 이분법과 가위치법, 수정 가위치법의 경우에는 구간을 [142.6, 142.8]로 설정해주도록 한다. 또한 고정점 반복법, 뉴튼-랩슨 법, 할선법의 경우에는 142.6을 초기값으로 줄 것이다.(고정점 반복법은 따로 확인할 것임.)*

1. **실행결과 및 분석**
2. **메인 소스 코드(이름: main3.m)**

cd=0.25;

g=9.81;

t=4;

v=36;

f = @(m) sqrt(g\*m/cd)\*tanh(sqrt(g\*cd/m)\*t)-v;

df = @(m) (981\*tanh(4\*(981/(400\*m))^(1/2)))/(50\*((981\*m)/25)^(1/2)) + (981\*(tanh(4\*(981/(400\*m))^(1/2))^2 - 1)\*((981\*m)/25)^(1/2))/(200\*m^2\*(981/(400\*m))^(1/2));

g = @(m) sqrt(g\*m/cd)\*tanh(sqrt(g\*cd/m)\*t)-v+m;

fplot(f, 'k');

hold on;

fplot(@(x) x\*0, 'r');

disp('1. bisection result')

[root err iter]= bisection(f, 142.6, 142.8, 10^-4, 100)

disp('2. falseposition result')

[root err iter]= falseposition(f, 142.6, 142.8, 10^-4, 100)

disp('3. modifiedfalseposition result')

[root err iter]= modifiedfalseposition(f, 142.6, 142.8, 10^-4, 100)

disp('4. fixedpoint result')

[root err iter]= fixedpoint(g, 142.6, 10^-4, 100)

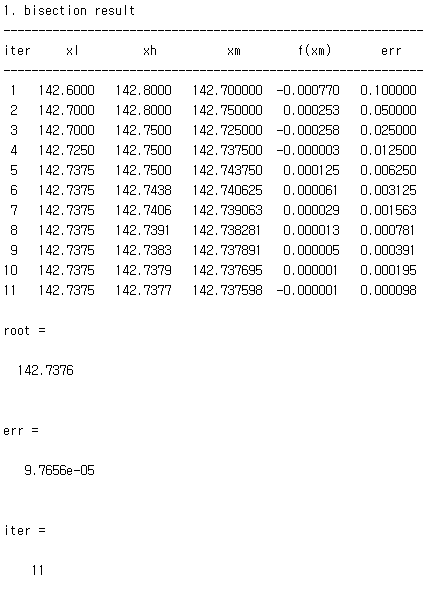
disp('5. newton result')

[root err iter]= newton(f, df, 142.6, 10^-4, 100)

disp('6. secant result')

[root err iter]= secant(f, 142.6, 142.8, 10^-4, 100)

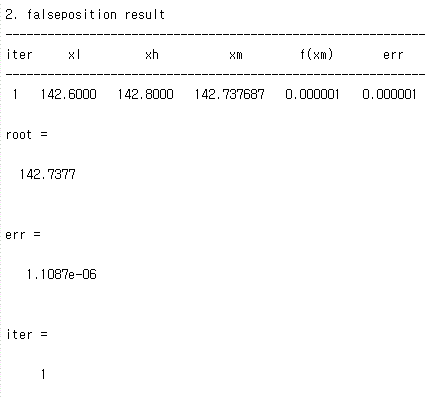
1. **이분법**

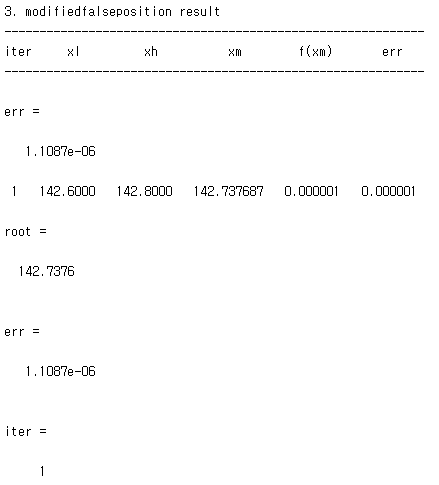


이분법은 구간을 정하여 xl과 xh를 설정하여 xm을 xl과 xh의 중간인 x좌표로 설정하는 방법이다. 수업 때 배운 bisection을 활용하여 bisection(f, 142.6, 142.8, 10^-4, 100)을 통해 호출하였다. 문제의 조건과 같이 허용오차는 10^-4(0.0001)으로 설정하였다.(이후 가위치법, 고정점 반복법 등 모든 함수들은 허용오차 10^-4, 최대 반복횟수 100으로 설정하여 호출하였다.)

결과는 위와 같이 근은 ‘142.7376’, 오차는 9.7656e-05, 총 반복횟수는 11번으로 나왔다.

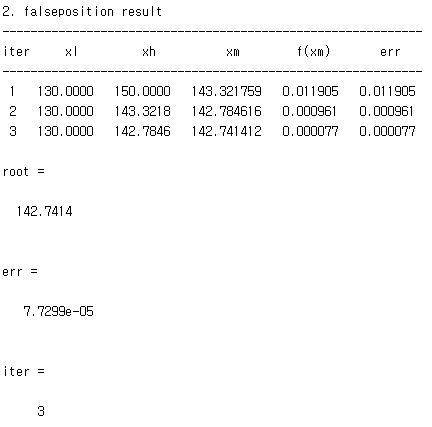
1. **가위치법, 수정 가위치법**

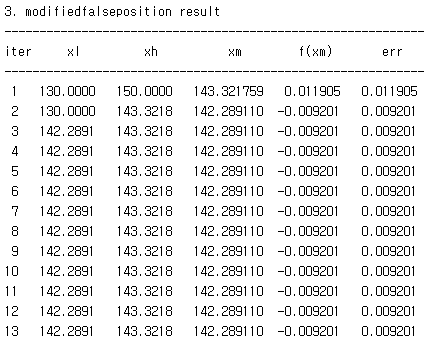




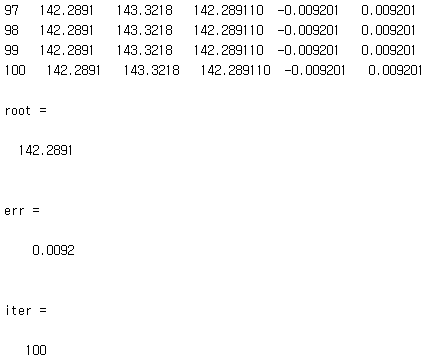
가위치법은 이분법과 동일한 구간법을 사용하며, 수정 가위치법은 가위치법을 보완한 방법이라고 할 수 있겠다. 하지만 이전 과제에서 봤듯 수정가위치법이 항상 가위치법보다 빠른 것은 아니다.

가위치법과 수정가위치법의 결과를 보면 근은 각각 ‘142.7377’, ‘142.7376’으로 0.0001밖에 차이가 나지 않으며 오차와 반복횟수 모두 동일한 것을 확인할 수 있다. **수정가위치법과 가위치법의 반복횟수가 별로 차이 나지 않은 것은 구간이 너무 작기 때문이라는 생각이 들었다.** 따라서 해당 가위치법과 수정 가위치법은 해당 구간 외에 구간을 더 늘려서 실행해보기로 하였다.





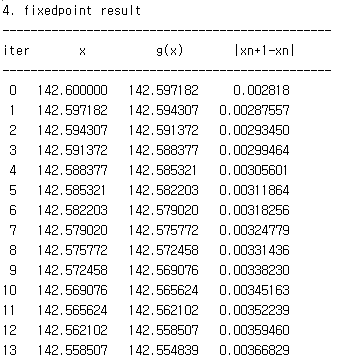
**…**



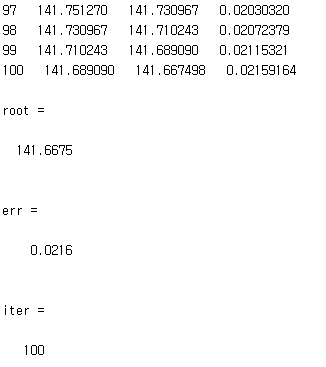
위의 결과는 구간을 늘려 [130, 150]으로 설정하여 실행한 결과이다. 가위치법은 이전과 비슷한 결과를 냈지만 오차가 커진 것을 확인할 수 있었다. 수정 가위치법의 경우 이전과는 다르게 수렴의 시간이 너무 오래 걸려 ‘최대 반복 횟수’로 설정한 100회에 도달하여 함수가 종료된 것을 확인할 수 있었다. 이와 같이 수정가위치법은 일반 가위치법보다 좋지 못한 결과를 낼 수 있다는 사실을 다시 한번 확인할 수 있었다.

또한 같은 구간법인 이분법과 비교했을 때 반복횟수 면에서 좋은 성능을 보여주고 있는 것이 확인 가능하다. 즉, 모든 상황에서 가위치법이 이분법보다 좋은 성능을 보여주는 것은 아니지만 가위치법의 전제 조건에 맞는 상황이라면 이분법 보다 더 좋은 성능을 낼 것임을 짐작할 수 있었다.

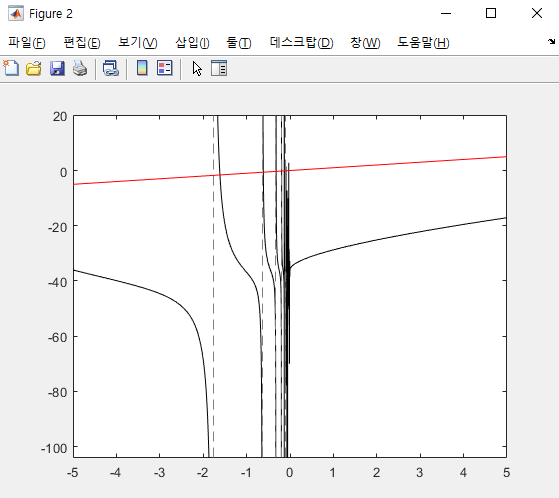
1. **고정점 반복법**

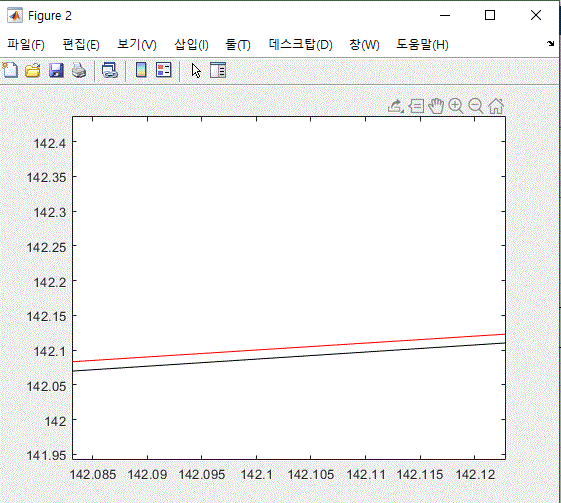
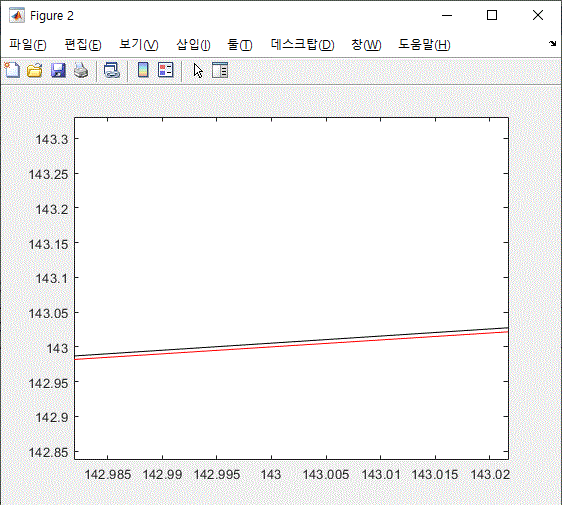


**…**



고정점 반복법은 다른 방식과는 다르게 원래의 함수 f를 쓰는 것이 아니라 g(m)를 활용하여 g(m)=m의 해를 구하는 것이다. 따라서 g(m) = sqrt(g\*m/cd)\*tanh(sqrt(g\*cd/m)\*t)-v+m = m; 이므로 이를 활용하여 고정점 반복법을 실행하도록 하였다. 또한 다른 방식들과는 다르게 g(m)을 활용한다 하였으므로 다시 초기값을 설정하였다.

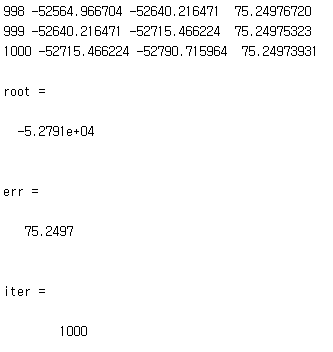
왼쪽의 그래프는 y=m그래프와 g(m)을 그린 것이다. f(m)함수와 비슷하게 g(m)와 y=m의 교점은 단 1개인 것으로 확인하였다.

왼쪽 그래프는 142일 때 y=m그래프보다 작은 값을 가지는 것을 확인한 것이고, 오른쪽은 143일 때 y=m그래프보다 큰 값을 가지는 것을 확인한 것이다. 따라서 이번에도 동일하게 초기값을 142.6로 설정하고 실행하였다.(142과 143사이에 있는 수이기 때문이다.)

그 결과 최대 반복횟수인 100회를 채우고 함수가 종료된 것을 확인할 수 있었다. 따라서 반복횟수를 더 늘려 1000회로 설정하여 실행해 보았다.

**…**



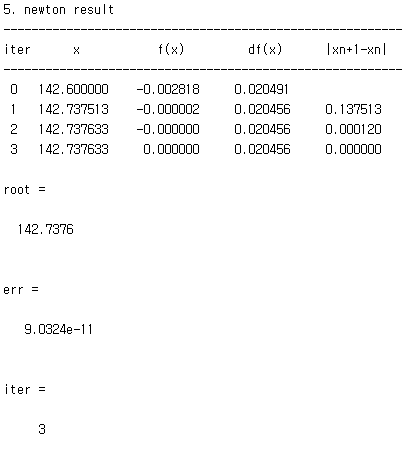
그 결과로 고정점 반복법은 발산한다는 사실을 발견할 수 있었다. 이 사실을 확인하기 위해 고정점 정리를 확인해보기로 하였다. 이를 위해서 g’(x)를 구하여

dg = @(m) (981\*tanh(4\*(981/(400\*m))^(1/2)))/(50\*((981\*m)/25)^(1/2)) + (981\*(tanh(4\*(981/(400\*m))^(1/2))^2 - 1)\*((981\*m)/25)^(1/2))/(200\*m^2\*(981/(400\*m))^(1/2)) + 1;

다음과 같이 정의하였다.

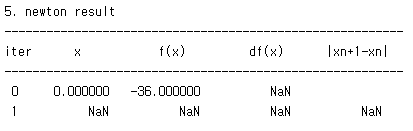
이후 feval(dg, 142.6)을 실행한 결과 ans=1.0205로 1보다 큰 값을 가지는 것을 확인하였다. 따라서 이는 고정점 정리를 만족하지 않음을 확인하였고 고정점 반복법이 발산한다는 결과가 맞는 결과임을 확인하였다. *(파란색 코드는 main3 안에 없고 따로 실행해보았다.)*

1. **뉴튼-랩슨 법**

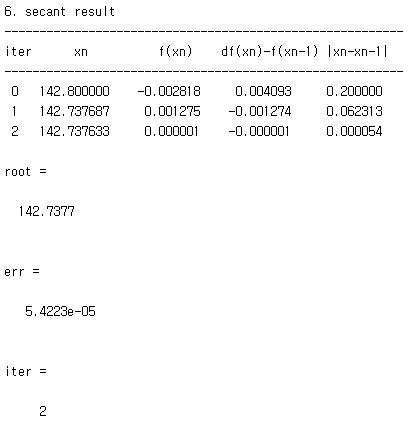


뉴튼-랩슨 법의 큰 특징은 함수의 도함수를 이용한다는 것이다. 함수의 도함수를 구하기위해 syms와 diff()를 활용하여 도함수를 구했고 main3.m에 선언해주었다.

초기값은 위에서 말한대로 142.6으로 선언해주었고, 근은 ‘142.7376’, 오차는 ‘9.0327e-11’, 반복횟수는 3으로 결과가 나왔다.

실험으로 초기값을 0으로 설정하고 뉴튼-랩슨 법을 실행해보았다. 결과는 위와 같이 Nan이 발생하였다. df(x)의 값이 Nan(not a number)이기 때문에 결과가 이렇게 나오는 것이라는 결론을 내렸다. 뉴튼-랩슨 법은 이와 같이 df(x)의 값에 주의하여 초기값을 설정해야 한다는 것을 확인하였다.

1. **할선법**



할선법은 함수 f의 도함수를 알지 못하거나 존재하지 않을 때 사용 가능한 방법이다. 할선법의 특징은 다른 개방법과는 다르게 초기값을 하나만 받는 것이 아니라 두 개를 받는다는 것이다. 따라서 초기값은 142.6, 142.8 두 값을 넣어 주기로 했다. 근은 ‘142.7377’, 오차는 ‘5.4223e-05’, 반복횟수는 2로 결과가 나왔다.

할선법은 가위치법과 비슷한데, 두 포인트를 대상으로 구한 다음 포인트의 값이 동일하기 때문이다. 비슷한 두 개의 방법의 결과를 비교해보면 가위치법은 근 ‘142.7377’, 오차 ‘1.1087e-06’, 반복횟수 1로 할선법에 비해 조금 빠르고 더 적은 오차의 값을 결과로 출력한 것을 확인할 수 있다.