**[수치해석 과제 보고서]**

**과제 2 : 근 구하기**

**201911291 컴퓨터공학부 허준호**

**과제의 목적 : 사용자 정의 함수 사용법을 익히고, 다양한 근 구하기 알고리즘을 이해하며, 각 알고리즘의 결과를 분석해본다.**

*#문제 1 : 계수가 실수인 이차 방정식의 근을 구하는 프로그램을 작성 하시오.*

* *결과 출력 시 실근, 중근, 허근을 구분하여 출력하고, 그래프도 출력 하시오.*
* 핵심 문제 : 계수가 실수인 이차 방정식을 매개 변수로 받아 근을 구하고, 결과로 나온 근이 실근인지, 중근인지, 허근인지 출력하고, 그래프도 출력하는 일을 총괄적으로 수행하는 함수를 만들어야 한다.
* 해결 방안 : MatLab에서는 복소수인 계수들을 갖는 다항식의 근을 구하는 roots함수를 제공하고 있다. 따라서, 다항식의 계수를 입력 인자로 받아서, 이를 배열로 생성한 다음에 roots 함수를 사용한다면 문제가 해결될 수 있다.

또한, 중근인지, 실근인지, 허근인지를 판단해야 하는 문제도 존재한다. 실근인지 허근인지 판명할 수 있는 기능 MatLab에서 isreal 함수로 제공하고 있다. Isreal 함수는 다항식의 계수들로 이루어져 있는 벡터를 입력 인자로 넘겨주었을 때, 해당 다항식의 근이 실수로 나오는 경우에는 true를 반환하고, 아닌 경우에는 false를 반환한다. 이를 통해, 실근인지 허근인지를 구별할 수 있게 된다. 중근인지, 그저 2개의 실근인지를 판별하기 위한 조건은 간단하다. 그저 두 근이 다른 지 같은 지에 대해서만을 판별하면 되기 때문이다. 이는 방정식의 두 근을 가지고 있는 배열의 원소를 직접적으로 비교해보면 된다.

그래프를 그리는 것에 대해서는 넘겨받은 계수들을 바탕으로 익명함수를 선언한 다음 fplot 함수를 이용하여 그래프를 그려주면 쉽게 그래프를 출력할 수 있다. 위의 해결 방안을 통하여 작성한 소스 코드는 아래와 같다.

**<소스 코드>**

[Solvin\_2.m]

function Solvin\_2(a,b,c)

vec = [a b c];

res = roots(vec)

if isreal(res)

if res(1) == res(2)

fprintf('중근입니다.\n')

else

fprintf('실근입니다.\n')

end

else

fprintf('허근입니다.\n')

end

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

fplot(f,'r')

hold on

fplot(@(x)x\*0,'b')

end

[untitled.m]

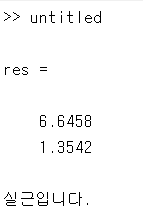
a = 1;

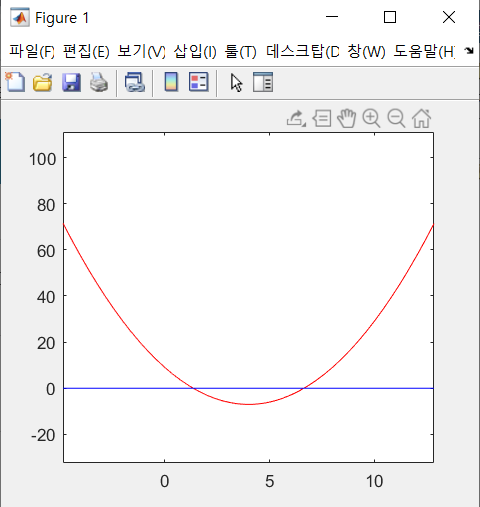
b = -8;

c = 9;

Solvin\_2(a,b,c)

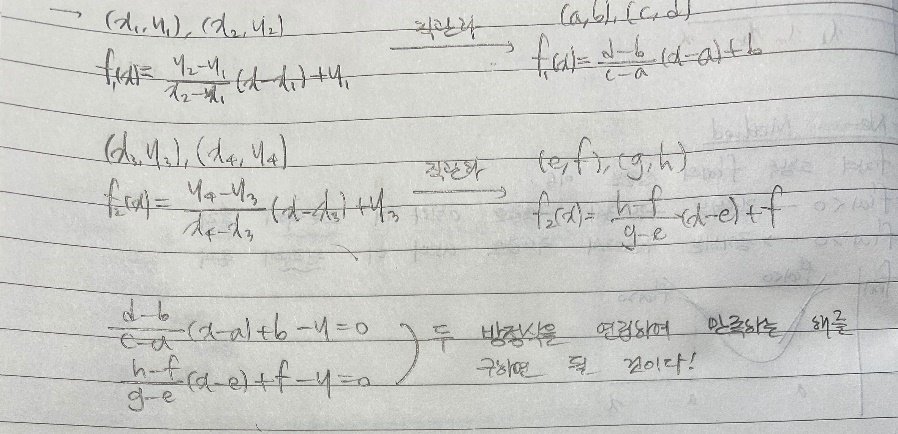
위의 제시된 기능들을 총괄적으로 수행할 함수 Solvin\_2를 정의해 주었다. 함수 내부에서 vec은 넘겨받은 이차식의 계수들을 모아 놓은 배열이고, res은 구하고 싶은 이차방정식의 근을 모아 놓은 배열이다. 그리고, 위에서 언급했던 roots 함수와 isreal 함수를 이용해서 근을 구하고, 근이 어떤 종류인지 분류하고 있다. 함수 마지막 부분에서는 이차식의 계수들을 이용해서 익명함수 f를 선언한 후, fplot 함수를 이용하여 그래프를 그려주었다. 근의 위치를 대략적으로 파악하기 위해 y=0의 그래프도 함께 그려주었다. 위 소스 코드의 실행 결과는 다음과 같다.

(위의 소스 코드에서는 x^2-8\*x+9=0에 대한 근을 구해보았다.)

(그래프 출력)

* 결과 분석 : 구하고자 하는 이차방정식의 계수들을 넘겨준 후 위와 같은 결과를 얻을 수 있다. 넘겨준 인자가 상수가 아니라 변수의 형태로 넘겨주었기 때문에 untitled.m 파일에서 a,b,c의 값을 원하는 값으로 지정해서 근을 구할 수 있다. 따라서, 모든 이차방정식에 대해서 적용될 수 있는 해결 방안이라는 것을 알 수 있다.

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

* 핵심 문제 : 두 점을 지나는 직선의 방정식을 만들어 내야 할 것이며, 두 직선의 교점, 즉 연립방정식의 근이 존재하는 경우 이에 대한 값을 출력할 수 있어야 한다. 이러한 기능을 총괄적으로 수행하는 함수를 만들어야 한다.
* 해결 방안 : 우선 두 점을 지나는 직선을 2개 만들기 위해서는, 각 직선 당 2개의 좌표가 주어져야 한다. 즉, 4개의 x좌표, 4개의 y좌표가 주어져야 한다. 이 주어진 좌표를 통해서 직선의 방정식을 세우게 된다면 아래와 같은 결과를 얻을 수 있다.(직선의 방정식 구하기)

위에서 구한 직선의 방정식을 바탕으로 미지수가 2개이고, 식이 2개인 연립 방정식을 세울 수 있고, 그곳에서 도출해낸 근이 바로 두 직선이 만나는 교점의 좌표가 된다는 것을 알 수 있다. 이러한 분석을 바탕으로 함수를 구현한다면 그에 대한 소스 코드는 아래와 같다.

**<소스 코드>**

[LineSolvin\_2.m]

function LineSolvin\_2(a,b,c,d,e,f,g,h)

syms x y

if (c-a==0)&&(g-e~=0)

eqn1 = x - a == 0;

eqn2 = (h-f)/(g-e)\*(x-e)+f-y == 0;

res = solve([eqn1, eqn2],[x,y]);

if isreal(res.x)&&isreal(res.y)

fprintf('교점의 x좌표 : %12.6f\n',res.x);

fprintf('교점의 y좌표 : %12.6f\n',res.y);

else

fprintf('교점이 존재하지 않습니다.\n');

end

elseif (c-a~=0)&&(g-e==0)

eqn1 = (d-b)/(c-a)\*(x-a)+b-y == 0;

eqn2 = x - g == 0;

res = solve([eqn1, eqn2],[x,y]);

if isreal(res.x)&&isreal(res.y)

fprintf('교점의 x좌표 : %12.6f\n',res.x);

fprintf('교점의 y좌표 : %12.6f\n',res.y);

else

fprintf('교점이 존재하지 않습니다.\n');

end

elseif (c-a==0)&&(g-e==0)&&(a==e)

fprintf('일치하는 직선입니다.\n');

elseif (c-a==0)&&(g-e==0)&&(a~=e)

fprintf('교점이 존재하지 않습니다.\n');

else

eqn1 = (d-b)/(c-a)\*(x-a)+b-y == 0;

eqn2 = (h-f)/(g-e)\*(x-e)+f-y == 0;

res = solve([eqn1, eqn2],[x,y]);

if isreal(res.x)&&isreal(res.y)

fprintf('교점의 x좌표 : %12.6f\n',res.x);

fprintf('교점의 y좌표 : %12.6f\n',res.y);

else

fprintf('교점이 존재하지 않습니다.\n');

end

end

end

위에 제시된 소스 코드는 다소 길게 보일 수 있으나, 이는 직선의 방정식을 세우는 경우에 여러 예외 사항이 존재하기 때문에 if문을 통해서 조건을 나눠 놓은 것 뿐이다. 입력 인자로는 4개의 x좌표, 4개의 y좌표를 받는다. 가장 중요한 것은 두 방정식 eqn1과 eqn2를 선언하고, 첫 번째 문제와 마찬가지로 solve 함수를 이용하여 근을 구한 다음 res에 저장한다. 벡터(배열) res에 저장된 값이 허수인 경우에는 좌표평면에서는 교점이 존재하지 않는다는 것으로 간주하고 교점이 존재하지 않는다는 문구를 출력하고, 실수인 경우에는 그에 해당하는 좌표값들을 출력하게 했다. 분모가 0이 되는 경우를 따로 나눠주었다. 이 경우에는 y축과 평행하는 직선이 나온다. 이러한 다양한 상황에 대해 정의한 LineSolvin\_2 함수가 정상적으로 작동하는지 확인하기 위해 untitled.m 파일에서 크게 3가지 경우로 나누어 실행해보았다.

(상황 1 : y=x+1과 y=-2x+9에 대해서 교점 구하기)

[untitled.m]

x1 = -1;

y1 = 0;

x2 = 0;

y2 = 1;

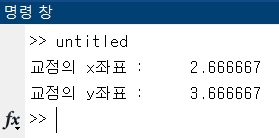
x3 = 3;

y3 = 3;

x4 = 1;

y4 = 7;

LineSolvin\_2(x1,y1,x2,y2,x3,y3,x4,y4);

(실행 결과)

(상황 2 : x=0과 y=-2x+9에 대해서 교점 구하기)

[untitled.m]

x1 = 0;

y1 = 0;

x2 = 0;

y2 = 1;

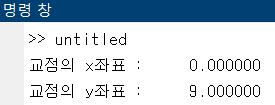
x3 = 3;

y3 = 3;

x4 = 1;

y4 = 7;

LineSolvin\_2(x1,y1,x2,y2,x3,y3,x4,y4);

(실행 결과)

(상황 3 : x=0과 x=1에 대해서 교점 구하기)

[untitled.m]

x1 = 0;

y1 = 0;

x2 = 0;

y2 = 1;

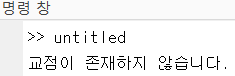
x3 = 1;

y3 = 1;

x4 = 1;

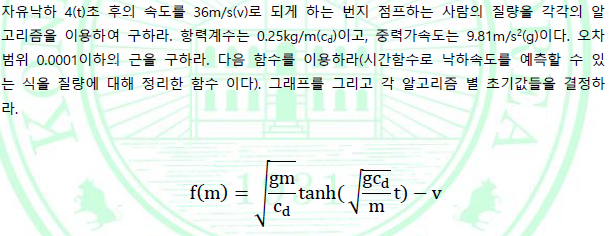
y4 = 5;

LineSolvin\_2(x1,y1,x2,y2,x3,y3,x4,y4);

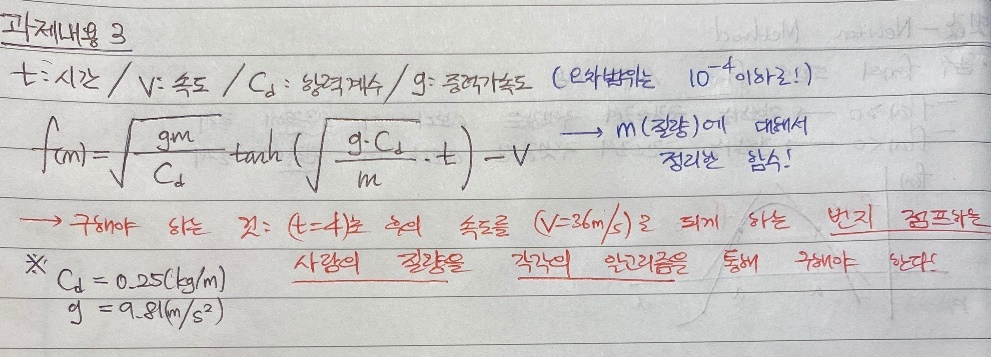
(실행 결과)

* 결과 분석 : 두 직선의 위치관계에서 나올 수 있는 모든 경우에 대해서 LineSolvin\_2 함수에서 if문을 통해 처리해 주었기 때문에, 4개의 좌표가 주어지기만 한다면, 각 두 점을 지나는 직선에 대해서 교점의 유무, 교점이 존재한다면 교점의 좌표를 출력해 준다는 것을 알 수 있다. 즉, 일반화에 성공했다는 것을 확인할 수 있다.

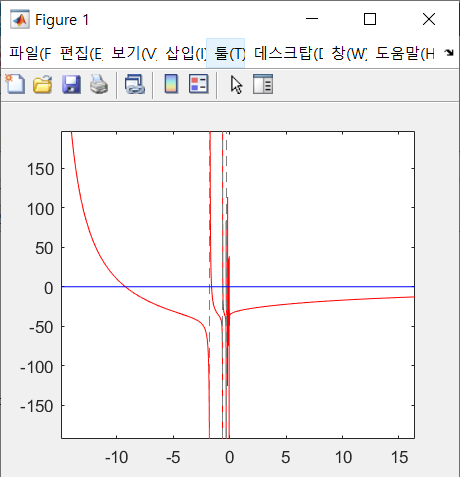
*#문제 3 : 이분법, 가상위치법, 고정점 반복법, 뉴튼 랩슨법, 할선법을 이용한 근 구하기*



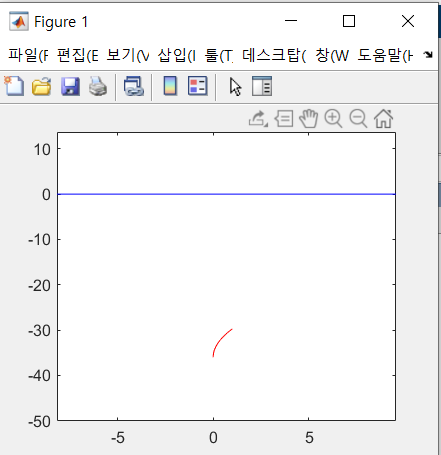
* 핵심 문제 : 우선 이러한 문제는 복잡한 문제 조건을 스스로가 이해할 수 있도록 정리하는 것이 먼저 수반되어야 한다. 위의 긴 조건을 단순화 시킨다면 아래의 정리한 것과 같게 된다.

(문제 조건 단순화) 즉, 위 문제를 단순화 시킨다면, 함수 f(m)에 해당하는 값들을 집어넣은 다음, 다양한 알고리즘을 통해서 근을 구하면 되는 문제이다.

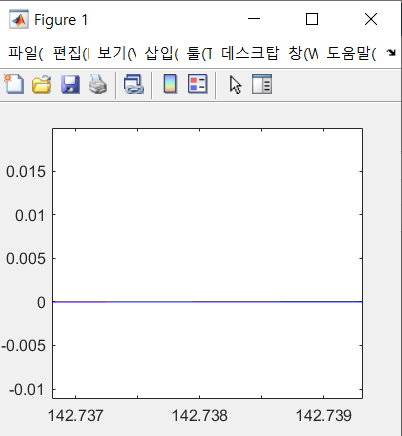
* 해결 방안 : 우선, 각 알고리즘을 적용하기 위해서는 위에서 사용하는 함수에 대해서 그래프를 그려보아야 한다. 위 함수에 대한 그래프를 fplot 함수를 이용하여 그려보게 되면 아래와 같다.

(그래프를 그려본 결과)

굉장히 복잡한 그래프가 그려진다는 것을 알 수 있다. 이를 좀 더 정확히 분석하기 위해서는 그래프를 확대하고, 구간별로 어떤 특성이 있는지를 확인할 필요가 있다는 것을 깨달을 수 있었다. 우선, 위 그래프만 보았을 때, 0 근처에서는 양수의 근이 존재하지 않아 보이는데, 이를 정확하게 확인하기 위해서 [0,1] 범위에서 함수의 그래프를 출력해 보았다. 결과는 아래와 같다.

 ([0,1] 범위에서의 함수의 그래프)

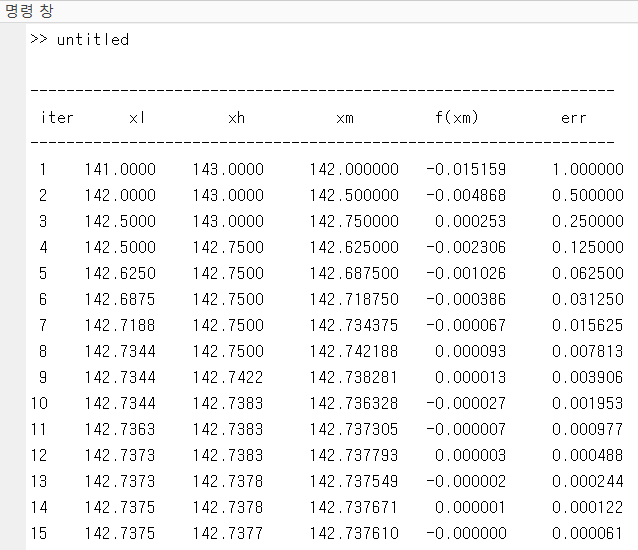
우리가 구하려는 근(질량), 즉 m의 값은 0보다 커야 한다는 조건을 바탕으로 했을 때, 0 부근에서는 근이 존재하지 않고, x축의 양의 방향으로 한참 가야 근이 존재한다는 것을 확인할 수 있다. 그리하여, x축의 양의 방향으로 근이 대략적으로 어디에 있는지 확인해보았다. 결과는 아래와 같다.

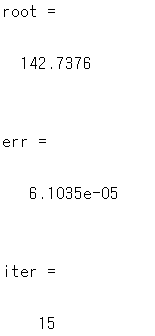
(대략적으로 m = 141~143 사이에서 확인)

대략적으로 [141,143] 사이에서 y=0의 그래프와 함수의 그래프가 만난다는 것을 확인할 수 있었다. 또한, tanh(x) 함수의 특징을 바탕으로 함수 f(m)이 m이 양수인 경우에 점점 값이 증가함에 따라, 함수값이 점점 증가한다는 것을 확인할 수 있다. 즉, 근을 구할 때 구간을 [141,143]으로 잡으면 무난하게 m이 양수인 범위에서 유일한 근을 구할 수 있다는 것을 확인할 수 있다. 이제 이분법, 가상위치법, 고정점 반복법, 뉴튼 랩슨법, 할선법, 이 5가지의 방법으로 각각 근을 구해보도록 하겠다.

1. 이분법

이분법에서 사용된 함수는 수업 시간에 제공되었던 함수 bisection을 이용했다. 구간은 [141,143]으로 설정했고, 허용 오차 범위는 문제에서 주어진 대로 10^-4 이하, 최대 반복 횟수는 혹시나 하는 마음에 충분히 1000회로 설정해서 입력 인자로 넘겨주었다. 근을 구하는 과정과 결과는 다음과 같다.

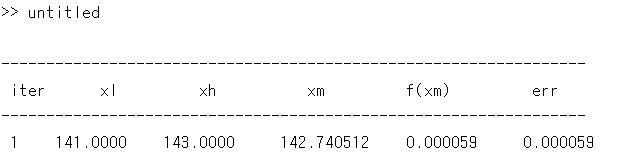
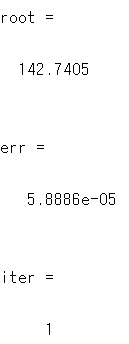
(구하는 과정)

(근을 구한 결과)

* 이분법을 통해 근을 구한 결과, 근은 142.7376, 이 때의 오차는 6.1035e-05, 근을 구하기 위해 반복된 횟수는 15번으로 측정되었다. 그리고, 추가적으로 xm값을 통해 구한 f(xm) 값이 지속적으로 양수와 음수의 범위를 넘나들고 있다는 것을 확인할 수 있다. 항상 수렴하지만 일반적으로 수렴 속도가 느리다는 단점을 가지고 있는 이분법으로 구했을 때에도 반복 횟수가 20회를 넘어가지 않는다는 점에서, 위 방정식의 근을 구하기 위해서 이분법을 쓰는 것은 나쁘지 않은 선택이라는 결론을 내릴 수 있을 것이다.

1. 가상위치법

가상위치법에서 사용된 함수는 이분법과 마찬가지로 수업 시간에 제공되었던 함수 falseposition을 이용했다. 구간은 이분법에서와 마찬가지로 [141, 143]으로 설정했고, 허용 오차 범위는 문제에서 주어진 대로 10^-4 이하, 최대 반복 횟수는 많이 필요하다고 생각되지 않아서 입력 인자로 넘겨 주지 않았다. 즉, 최대 반복 횟수는 100번인 것이다. 근을 구하는 과정과 결과는 다음과 같다.

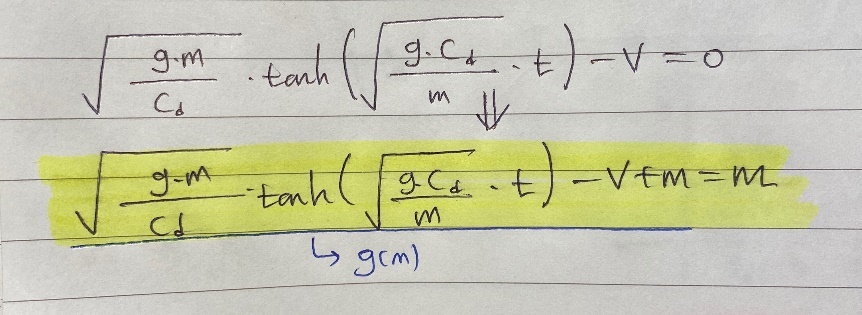
(구하는 과정)(근을 구한 결과)

* 가상위치법을 통해 근을 구한 결과, 근은 142.7405, 이 때의 오차는 5.8886e-05, 근을 구하기 위해 반복된 횟수는 단 1회로 측정되었다. 가위치법의 알고리즘 특성상 이분법과 달리 구간의 양 끝 xl, xh의 값에 대해서 f(xl), f(xh)을 이용하여, (xl,f(xl)), (xh, f(xh))를 지나는 직선을 구한 후, 이 직선이 x축과 만나는 점을 xm으로 삼는다. 그런데, 위 과정에서는 첫번째에서 구한 xm이 바로 허용 오차 범위 10^-4보다 작아지게 되어 바로 근이 구해진 것이다. 심지어 오차범위도 이분법에서 구한 것보다 소폭 작게 측정되었다.

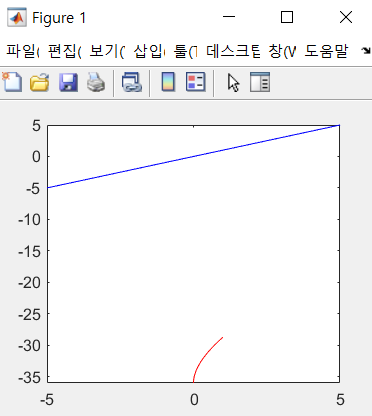
이러한 결과를 바탕으로 보았을 때, 위 방정식의 근을 구하기 위해서 가위치법을 사용하는 것은 탁월한 선택이었다는 결론을 내릴 수 있다. 또한, 가위치법의 수렴 속도가 느릴 때 대안으로 사용할 수 있는 수정 가위치법을 사용할 필요가 없다는 것도 위 결과를 통해 결론을 내릴 수 있을 것이다.

1. 고정점 반복법

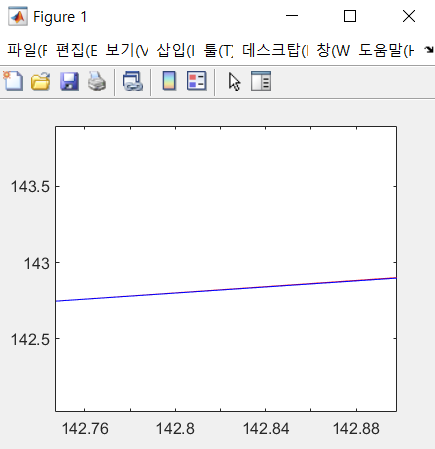
우선 고정점 반복법을 사용하기 위해 f(m)=0으로부터 g(m)=m의 형태로 변환시켜야 한다. 따라서, 문제에서 주어진 식을 g(m)=m의 형태로 변환시킨 결과는 아래와 같다.

(g(m)=m의 형태로 변환)

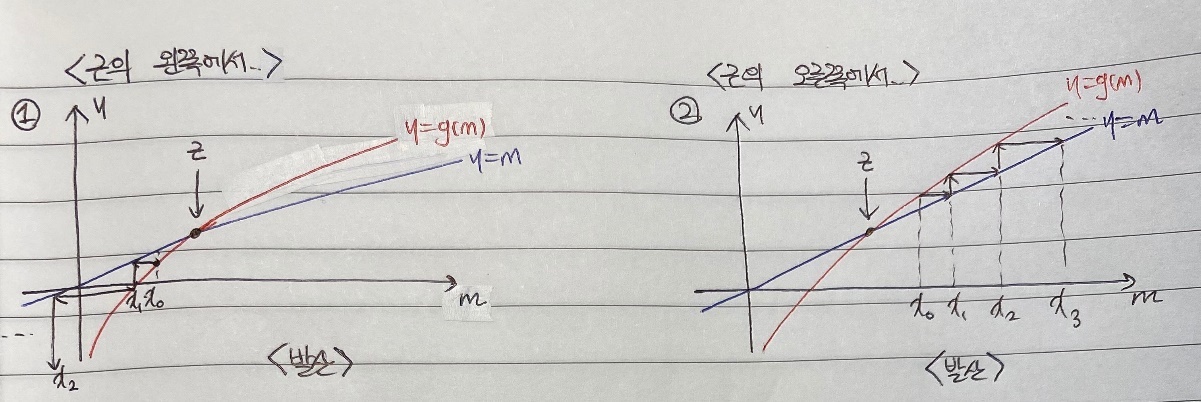
위 형태에서 근을 구하기 위해서는 g(m)과 y=m에 대한 그래프의 관계가 어떻게 되는지 알 필요가 있다. 따라서, g(m)과 y=m에 대해서 그래프를 그려보게 된다면 아래와 같은 결과를 확인할 수 있다.

([0,1] 범위에서의 두 그래프 간의 관계)

근을 구하기 전에 확인했던 거와 마찬가지로 m이 양의 실수인 범위에서 0의 근처에서는 근을 갖기 않는다는 것을 확인할 수 있다. 따라서, 마찬가지로 x축의 양의 방향으로 한참 가야 근이 존재한다는 것을 확인할 수 있다. 이번에도 x축의 양의 방향으로 근이 대략적으로 어디에 있는지 확인해보았다. 결과는 아래와 같다.

(대략적으로 m=141~143 사이에서 확인)

고정점 반복법에서 사용된 함수는 수업 시간에 제공되었던 fixedpoint 함수를 이용할 것이다. 그런데, 거의 대부분의 근을 구하는 알고리즘에서 주의해야 할 것이, 사용하려는 알고리즘이 근을 구하는 데에 사용할 수 없을 가능성이 있다는 것을 염두해 두어야 한다. 이번 상황이 그러한 상황이다. 왜냐하면, 이번 상황에서 고정점 반복법을 사용할 경우 근으로 수렴하지 않고, 끊임없이 발산하기 때문이다. 아래 그래프의 개형을 통해 아래 상황이 발산하는 경우를 시각적으로 증명하고 있다.

(발산하는 경우 증명)

* 원래의 f(m)에서 +m이 추가된 것이 g(m)이므로, f(m)=0에서와 비슷하게 y=m과 m이 양의 값의 범위에서는 근이 발생하는 것이 확인된 지점 이외에는 근이 더 이상 발생하지 않을 것이기 때문에 위와 같은 그래프의 개형이 그려질 수 있다. 이러한 분석을 바탕으로 위의 문제에서는 고정점 반복법은 사용할 수 없다는 것으로 결론 지을 수 있다.

1. 뉴튼-랩슨법

뉴튼-랩슨법에서 사용된 함수 또한 수업 시간에 제공되었던 함수 newton을 이용했다. 뉴튼-랩슨법에서는 앞의 방법들과 다르게 f(m)을 미분한 함수 f’(m)을 필요로 한다. 따라서, 본격적으로 근을 구하기 전, f(m)을 미분한 함수 f’(m)을 구하면 다음과 같다.

f’(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))

위 함수는 직접 미분하기 상당히 복잡했기 때문에 아래 소스 코드를 사용하여 미분 함수를 도출했다.

**<소스 코드>**

%각 변수에 상수를 할당

g = 9.81;

cd = 0.25;

t = 4;

v = 36;

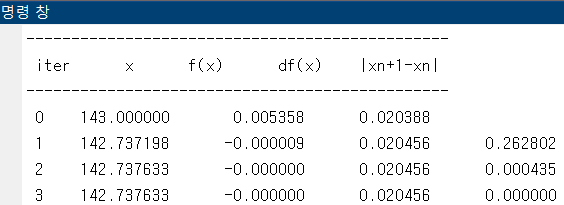
%위에 할당한 상수를 응용하여 아래 과정을 수행

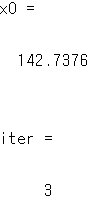
syms m

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

diff(y)

뉴튼-랩슨 방법의 알고리즘의 특성과 위에서 확인했던 f(m)의 그래프 개형의 특성을 고려하여 시작점 x0를 143, 즉 근에 충분히 가까운 근의 오른쪽 값을 주어 실행했다. 허용 오차 범위는 문제에서 제시되었듯이 10^-4로, 혹시 모르니 최대 반복 횟수를 1000회로 주어 함수를 실행했다. 결과는 아래와 같다.

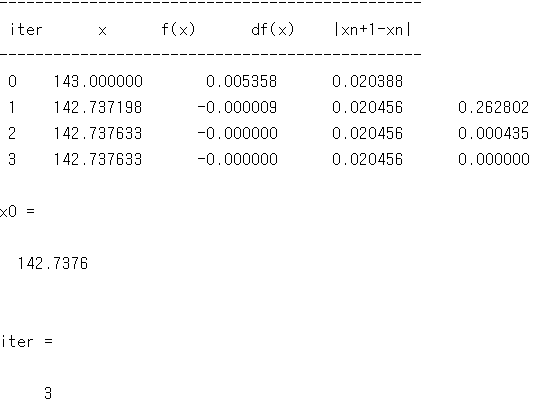
(구하는 과정)

(근을 구한 결과)

* 뉴튼-랩슨법을 통해 근을 구한 결과, 근은 142.7376, 이 때의 오차는 0.000000, 근을 구하기 위해 반복된 횟수는 3번으로 측정되었다. 이는 시각적으로 확인하기에는 다소 힘들 수 있지만, 근에 수렴하지 못하는 경우가 아니며, 수렴 속도가 매우 빨라서 빠른 시간 내에 근을 찾을 수 있다는 뉴튼-랩슨법의 취지에 부합한다는 것을 알 수 있다. 그러나, 가위치법에서 1번만에 근을 구했던 것에 비해 근소하게 성능이 떨어진다는 것을 확인할 수 있다. 대신에, 오차는 가위치법에 비해 소폭 감소하였다.

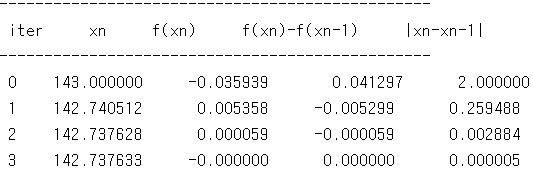
위 과정에서 확인할 수 있는 뉴튼-랩슨법의 또다른 단점이 있다. 바로, 함수를 미분해야 하는 한 단계를 더 거쳐야 하는 점이다. 특히, 위와 같은 경우에는 함수의 미분한 형태가 매우 복잡하기 때문에 Matlab에서 제공하는 diff 함수가 없는 경우에는 사용을 지양해야 하는 알고리즘이라는 것을 확인할 수 있다.

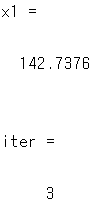
시작점 x0를 141, 즉 근에 충분히 가까운 근의 왼쪽 값을 주어 실행시켜도 위와 비슷한 결과를 얻을 수 있다. 이를 통해 얻어낸 결과는 아래와 같다.

(x0 = 141일 때, 근을 구한 과정과 결과)

1. 할선법

할선법에서 사용된 함수는 수업 시간에 제공되었던 secant 함수를 이용했다. 할선법의 알고리즘의 특성과 그래프의 개형을 고려하여 시작점 x0를 141, 끝점 x1을 143으로 두었다. 그리고, 허용 오차 범위는 문제에서 제시 되었던 10^-4로 잡았고, 혹시 모르니 최대 반복 횟수를 1000회로 주어 함수를 실행했다. 결과는 아래와 같다.

(구하는 과정)

(근을 구한 결과)

* 할선법을 이용하여 근을 구한 결과, 근은 142.7276, 이때의 오차는 0.000005, 근을 구하기 위해 반복된 횟수는 3회로 측정되었다. 할선법은 뉴튼-랩슨법과 달리 도함수 대신에 두 점을 사용한 직선의 기울기를 사용하기 때문에, 두개의 x좌표를 입력 인자로 넘겨주어야 한다. 할선법 또한 뉴튼-랩슨법과 비슷한 시간에 근을 구할 수 있다는 것을 확인할 수 있다. 시각적으로 확인하기는 어렵지만, 근에 수렴한다는 것도 확인할 수 있는 결과이다.

다만 위 문제에서 만큼은 뉴튼-랩슨법에 비해 할선법이 더 효율적이라고 말할 수 있을 것이다. 왜냐하면, 뉴튼-랩슨법에서는 복잡한 f’(m) 식을 도출한 뒤에 실행했는 데에도 할선법과 동일하게 3번 반복되었기 때문이다. 할선법에서는 두 개의 x좌표만을 넘기고 동일한 시간에 근을 구했기 때문에 이번 상황에서는 할선법이 뉴튼-랩슨법에 비해 더 우수했다고 결론 지을 수 있을 것이다.

* 이렇게 5가지 방법으로 한 방정식의 근을 구해보았다. 위의 결과들을 바탕으로 보았을 때, 가위치법이 위 상황에서는 가장 우수했으며, 그 다음으로는 할선법, 뉴튼-랩슨법, 이분법 순으로 성능이 우수했다. 고정점 반복법으로는 근을 구할 수 없었다.

이러한 결론을 바탕으로 총정리를 하자면, 방정식의 근을 구하는 상황마다 우수한 방법은 달라질 수 있다는 것이다. 위 상황에서는 고정점 반복법을 아예 사용할 수 없는 상황이었지만, 어떤 경우에는 고정점 반복법이 가장 효율적인 알고리즘이 될 수도 있는 것이다. 그리고, 이러한 알고리즘의 선택의 바탕에는 그래프 분석이 우선시되어야 한다는 것을 알 수 있다. 왜냐하면, 위에서 시행한 모든 알고리즘에서는 그래프 분석을 통해 구간을 정하고, 시작점을 정하는 등의 행동을 취했기 때문이다.