보고서 : 실습과제 1, 실습과제 2

2019112096 서준원

I. 실습과제 1 - 이분법

I-1) bisection.m 코드 전문

```
%% 0. 이분법을 활용한 방정식의 해 찾기 스크립트
clear, clc;
%% 1. 선언 및 입력
syms x; % x는 함수의 정의역
f(x) = input('계산할 x에 관한 방정식을 입력하시오 : '); % 방정식 f(x) 입력
disp("초기값 x1과 x2의 조건 : f(x1)*f(x2) <= 0"); % 초기값 조건 출력
x1 = input('초기값 x1 입력 : '); % 초기값 x1 입력
x2 = input('초기값 x2 입력 : '); % 초기값 x2 입력
max_err = input('허용할 최대 오차 입력 : '); % 최대오차 입력
err = 1; % 초기 오차 설정
no_iter = 0; % 실행 횟수 카운터 설정
%% 2. 알고리즘 실행
while err > max_err % 오차가 최대 오차보다 작아지면 루프 중단
    no_iter = no_iter + 1; % 카운터 1 증가
    xmid = (x1+x2)/2; % 중간값 계산
if f(x1)*f(xmid)<0 % x1과 중간값이 이분법 사용 가능할때
    x2 = xmid; % x2를 기존 중간값으로 대체
elseif f(xmid)*f(x2)<0 % x1과 중간값이 불가능하면 x2와 중간값 비교
    x1 = xmid; % x1을 기존 중간값으로 대체
elseif (f(xmid)*f(x2) == 0 || (f(xmid)*f(x1) == 0)) % 중간값이 정확히
해일 때
    disp("정확한 해를 찾았습니다!"); % 메시지 출력
    disp("정확한 해 = " + xmid); % 정확한 해 출력
    return; % 스크립트 실행 중지
else  % 이도 저도 아니면 (=오류)
    disp("오류 발생! x1과 x2를 제대로 입력해주세요."); % 메시지 출력
    return; % 스크립트 실행 중지
end
root = xmid; % 결과값 대입
err = abs(f(root)); % 오차 계산
end
%% 3. 결과값 형변환 및 출력
err_output = double(err); % 오차값 실수형으로 형변환
disp("연산을 반복한 횟수 no_iter = " + no_iter); % 반복 횟수 출력
disp("해의 근사값 root"); root
disp("발생한 오차 err"); err_output
```

I-2) 실행 결과 - $f(x)=x^3-3x+1,\ x_1=-3,\ x_2=-1,\ MAX.ERR=0.001$

```
계산할 x에 관한 방정식을 입력하시오 : x^3-3*x+1
초기값 x1과 x2의 조건 : f(x1)*f(x2) <= 0
초기값 x1 입력 : -3
초기값 x2 입력 : -1
허용할 최대 오차 입력 : 0.001
연산을 반복한 횟수 no_iter = 12
해의 근사값 root

root =

  -1.879394531250000

발생한 오차 err

err_output =

     7.056735921651125e-05
```

I-3) 실행 결과 - $f(x)=x^3-3x+1,\ x_1=-3,\ x_2=-20,\ MAX.ERR=0.001$

```
계산할 x에 관한 방정식을 입력하시오 : x^3-3*x+1
초기값 x1과 x2의 조건 : f(x1)*f(x2) <= 0
초기값 x1 입력 : -3
초기값 x2 입력 : -20
허용할 최대 오차 입력 : 0.001
오류 발생! x1과 x2를 제대로 입력해주세요.
```

I-4) 실행 결과 - $f(x)=x^2+2x+1,\ x_1=-2,\ x_2=0,\ MAX.ERR=0.001$

```
계산할 x에 관한 방정식을 입력하시오 : x^2+2*x+1
초기값 x1과 x2의 조건 : f(x1)*f(x2) <= 0
초기값 x1 입력 : -2
초기값 x2 입력 : 0
허용할 최대 오차 입력 : 0.001
정확한 해를 찾았습니다!
정확한 해 = -1
```

II. 실습과제 2 - 할선법

II-1) secant.m 코드 전문

```
%% 0. 할선법을 이용한 방정식의 해 찾기 스크립트
clear, clc;
%% 1. 선언 및 입력
syms x; % x는 함수의 정의역
f(x) = input('계산할 x에 관한 방정식을 입력하시오 : '); % 방정식 f(x) 입력
x1 = input('초기값 x1 입력 : '); % 초기값 x1 입력
x2 = input('초기값 x2 입력 : '); % 초기값 x2 입력
max_err = input('허용할 최대 오차 입력 : '); % 허용할 최대 오차 입력
n=2; % 증감 카운터 선언, 2부터 시작 (점화식)
x(1)=x1; % x벡터의 인덱스 1에 x1 대입
x(2)=x2; % x벡터의 인덱스 2에 x2 대입
err=1; % 초기 오차 설정
%% 2. 알고리즘 실행
while err > max_err % 오차가 최대 오차보다 작아지면 루프 중지
    n = n+1; % 카운터 1 증가
    if f(x(n-1))-f(x(n-2)) == 0 % 초기값의 중간값이 해일 때
        disp("정확한 해를 찾았습니다!"); % 메시지 출력
        answer = ((x(n-1))+(x(n-2)))/2; % 해 계산
        disp("정확한 해 = " + double(answer)); % 형변환된 해 출력
        return; % 스크립트 종료
    end
      k = (f(x(n-1))-f(x(n-2)))/(x(n-1)-x(n-2)); % 기울기 k 선언
    x(n)=x(n-1)-(f(x(n-1))/k); % 점화식 선언
    err = abs(f(x(n))); % 발생한 오차 계산
end
%% 3. 결과값 형변환
root = double(x(n)); % 결과값을 실수형으로 형변환(기존형 : rational)
err_output = double(err); % 오차값을 실수형으로 형변환(기존형 : rational)
%% 4. 결과값 출력
disp("연산을 반복한 횟수 = " + n); % 연산 횟수 출력
disp("해의 근사값"); root % 실수형 근사값 출력
disp("발생한 오차"); err_output % 실수형 오차값 출력
```

II-2) 실행 결과 - $f(x) = x^3 - 3x + 1,\ x_1 = -3,\ x_2 = -1, MAX.ERR = 0.001$

```
계산할 x에 관한 방정식을 입력하시오 : x^3-3*x+1
초기값 x1 입력 : -3
초기값 x2 입력 : -1
허용할 최대 오차 입력 : 0.001
연산을 반복한 횟수 = 11
해의 근사값

root =

  -1.879383738345717

발생한 오차

err_output =

     1.141889356258172e-05
```

II-3) 실행 결과 - $f(x) = x^3 - 3x + 1,\ x_1 = -3,\ x_2 = -20, MAX.ERR = 0.001$

```
계산할 x에 관한 방정식을 입력하시오 : x^3-3*x+1
초기값 x1 입력 : -3
초기값 x2 입력 : -20
허용할 최대 오차 입력 : 0.001
연산을 반복한 횟수 = 9
해의 근사값

root =

  -1.879486710116133

발생한 오차

err_output =

     7.708401708337936e-04
```

II-4) 실행 결과 - $f(x) = x^2 + 2x + 1, x_1 = -2, x_2 = 0, MAX.ERR = 0.001$

```
계산할 x에 관한 방정식을 입력하시오 : x^2+2*x+1
초기값 x1 입력 : -2
초기값 x2 입력 : 0
허용할 최대 오차 입력 : 0.001
정확한 해를 찾았습니다!
정확한 해 = -1
```

Ⅲ. 비교 및 분석

발견했던 가장 큰 차이점으로는, 할선법은 이분법과는 달리 초기값의 제한 조건이 없었다는 점이다. $f(x_1)f(x_2)<0$이 전제 조건이던 이분법과는 달리, 할선법은 할선을 그린 뒤 반복하는 과정이므로, 언젠가는 해에 근사되기 때문에 초기값의 제한 조건이 없었다.

Ⅲ-1) 수렴 속도 - $f(x)=x^3-3x+1,\ x_1=-3,\ x_2=-1$

두 알고리즘의 수렴 속도는 대체로 할선법이 우세했고, 특히 허용 오차가 작아질수록 수렴 속도 차는 기하급수적으로 벌어졌다.

	이분법	할선법
0.001	12	9
0.000001	23	10
0.00000001	28	11

Ⅲ-2) 오차 - $f(x)=x^3-3x+1,\ x_1=-3,\ x_2=-1$

시행 횟수가 적을 때는 이분법 측이 더 정확한 근사값을 구했지만, 시행 횟수가 늘어날수록 할선법 측의 오차가 급격히 줄어든다.

	이분법	할선법
0.001	7.056735921651125e-05	9.649826921804344e-05
0.000001	6.568037275068593e-08	7.410993129921665e-08
0.00000001	9.083776187660492e-09	6.987789515167469e-13

편의를 위해 부동 소숫점 15자리 형식에서 실수형으로, 유의미한 소수점에서 버린 변환한 데이터는 다음과 같다.

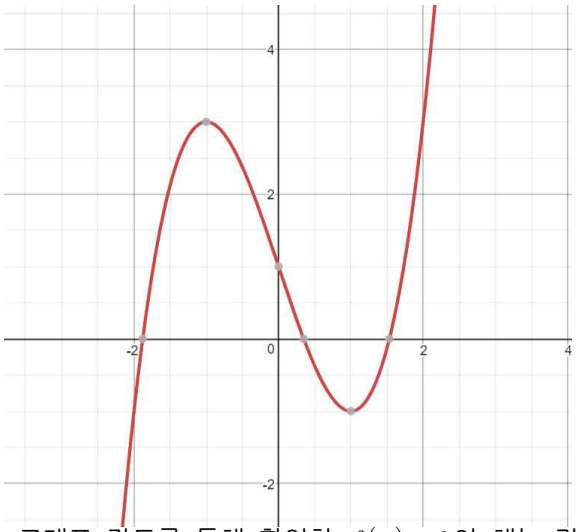
	이분법	할선법
0.001	0.00007	0.00009
0.000001	0.00000006	0.00000007
0.00000001	0.000000009	0.0000000000006

III-3) 해를 찾을 가능성

해를 찾을 가능성은 두 알고리즘이 동일하다. 두 알고리즘 모두 초기값의 설정이 정확한 해 탐색 여부를 좌우한다.

$f(\frac{x_2 - x_1}{2}) = 0$이나 $f(\frac{x_1 - x_2}{2}) = 0$ 둘 중 하나라도 0이라면 그 값이 바로 해이다.

III-4) 다수의 해가 존재하는 경우 - $f(x) = x^3 - 3x + 1, MAX.ERR = 0.001$



그래프 작도를 통해 확인한 $f(x) = 0$의 해는 각 -1.879, 0.347, 1.532였다.

이분법의 경우, 두 초기값의 중간값과 x_1의 사이에 있는 해 쪽으로 근사될 것이다.

할선법의 경우, 첫 할선의 x절편과 가장 가까이 있는 해 쪽으로 근사될 것이다.

	이분법	할선법
(-3, -1)	-1.879394531250000	-1.879383738345717
(-1, 1)	0.347656250000000	0.347294784910048
(-20, 20)	0.347290147344840	-1.879272460937500

IV. 저번 보고서의 보강

저번 보고서에서,

"소수점 아래 5자리 이상 출력되었다. 왜 disp() 함수에서 이러한 결과가 발생하였는지 탐구를 몇 번이고 시도해 보았으나, 아쉽게도 밝혀내지 못하였다."

라고 언급하였다. 이번 과제를 해결하면서, disp() 함수에 대해 알아낸 사실 한가지를 첨언하고자 한다.

disp() 함수에서 출력되는 변수는, char형 즉 문자열로 변환된다. 따라서 format의 설정이 아무런 의미가 없었으리라 추정된다. 그렇다고 하더라도, 불규칙적인 고정 소수점의 자리수에 관해서는 아직까지도 진전이 없는 부분이 아쉬울 따름이다.

V. 함수 형태로의 입출력

V-1) bisect_method.m 코드 전문

```
%% 0. 이분법을 활용한 방정식의 해 찾기 함수
function [root, err, no_iter] = bisect_method(x, x1, x2, max_err)
%% 1. 선언 및 입력
f(x) = f;
err = 1; % 초기 오차 설정
no_iter = 0; % 실행 횟수 카운터 설정
%% 2. 알고리즘 실행
while err > max_err % 오차가 최대 오차보다 작아지면 루프 중단
    no_iter = no_iter + 1; % 카운터 1 증가
    xmid = (x1+x2)/2; % 중간값 계산
if f(x1)*f(xmid)<0 % x1과 중간값이 이분법 사용 가능할때
    x2 = xmid; % x2를 기존 중간값으로 대체
elseif f(xmid)*f(x2)<0 % x1과 중간값이 불가능하면 x2와 중간값 비교
    x1 = xmid; % x1을 기존 중간값으로 대체
elseif (f(xmid)*f(x2) == 0 || (f(xmid)*f(x1) == 0)) % 중간값이 해이면
    root = xmid;
    err = 0;
    return; % 스크립트 실행 중지
else  % 이도 저도 아니면 (=오류)
    root = "err";
    err = "err";
    return; % 스크립트 실행 중지
end
root = xmid; % 결과값 대입
err = abs(f(root)); % 오차 계산
end
%% 3. 결과값 형변환
err = double(err); % 오차값 실수형으로 형변환
end
```

V-2) 실행 결과 - bisect_method.m

```
>> [root, err, no_iter] = bisect_method(x^3-3*x+1, -3, -1, 0.001)

root =
   -1.8794

err =
   7.0567e-05

no_iter =
    12
```

V-3) secant_method.m 코드 전문

```
%% 0. 할선법을 이용한 방정식의 해 찾기 함수
function [root, err, no_iter] = secant_method(f, x1, x2, max_err)
%% 1. 선언 및 입력
syms x; % x는 함수의 정의역
f(x) = f;
n=2; % 증감 카운터 선언, 2부터 시작 (점화식)
x(1)=x1; % x벡터의 인덱스 1에 x1 대입
x(2)=x2; % x벡터의 인덱스 2에 x2 대입
err=1; % 초기 오차 설정
%% 2. 알고리즘 실행
while err > max_err % 오차가 최대 오차보다 작아지면 루프 중지
    n = n+1; % 카운터 1 증가
    if f(x(n-1))-f(x(n-2)) == 0 % 초기값의 중간값이 해일 때
        root = ((x(n-1))+(x(n-2)))/2; % 해 계산
        err = 0;
        return; % 스크립트 종료
    end
    k = (f(x(n-1))-f(x(n-2)))/(x(n-1)-x(n-2)); % 기울기 k 선언
    x(n)=x(n-1)-(f(x(n-1))/k); % 점화식 선언
    err = abs(f(x(n))); % 발생한 오차 계산
end
%% 3. 결과값 형변환
root = double(x(n)); % 결과값을 실수형으로 형변환(기존형 : rational)
err = double(err);
no_iter = n;
end
```

V-4) 실행 결과 - secant_method.m

```
>> [root, err, no_iter] = secant_method(x^3-3*x+1, -3, -1, 0.001)

root =
   -1.8794

err =
   1.1419e-05

no_iter =
    11
```

V-5) 함수로 실행 시 주의사항

*syms x*를 통해 x를 함수의 정의역으로 설정하는 것은, function 내부가 아닌, function을 실행시키기 이전에 해줘야 한다. 그렇지 않으면 x가 정의되지 않았다는 오류가 발생한다.