4주차 보고서

Created	@October 13, 2021 5:24 PM
Created by	**20190258 김혜린**

프로그램 구동 방법

프로그램은 Newton-Raphson, Secant, Bisection 방법에 대한 비선형 방정식의 풀이 과정을 수행한다.

`function.cpp`에 실험해보고자하는 함수와 도함수가 정의되어 있으며 해당 함수를 `main.cpp` 에 각각 `_f` 와 `_fp` 변수에 할당한 뒤, 원하는 방법이 구현된 함수를 실행해주면된다.

풀이 과정은 "result.txt"에 저장되며 따라서 풀이 방법이 구현된 함수는 파일 포인터 `fp` 를 파라미터로 가진다.

각 방법은 모두 초기값이 필요한데 이는 사용자에게 입력을 받는 방식으로 구동된다. 각 방법은 아래의 함수에 구현되어 있다.

- Newton-Raphson
  - `void program1_1(FILE* fp)` in `program1_1.cpp`
- Secant
  - `void program1_2(FILE* fp)` in `program1_2.cpp`
- Bisection
  - `void program1_3(FILE* fp)` in `program1_3.cpp`

`void program1_1(FILE* fp)`

Newton-Raphson 방법의 경우, 초기값 `x0` 를 사용자로부터 입력받아 프로그램의 종료조건을 만족할 때까지 `x1 = x0 - (_f(x0) / _fp(x0));` 식을 이용해 다음 x를 구한다.

`void program1_2(FILE* fp)`

Secant 방법의 경우, 초기값 `x0`, `x1` 를 사용자로부터 입력받아 프로그램의 종료조건을 만족할 때까지 `temp = x1 - _f(x1)*((x1-x0)/(_f(x1)-_f(x0)));` 식을 이용해 다음 x(`temp`)를 구한다.

`void program1_3(FILE* fp)`

Bisection 방법의 경우, 초기값 `a0`, `b0` 를 사용자로부터 입력받아 프로그램의 종료조건을 만족할 때까지 `x1 = (a0 + b0) / 2;` 식을 이용해 다음 x를 구한다.

이때 다음 구간을 적절히 설정해주어 다시 `a0`와 `b0`을 할당해 다음 x를 구하는 과정을 진행하는데 이에 대한 로직은 아래 **'Bisection 방법을 이용한 비선형 방정식의 풀이 분석 (숙제1)'** 파트에 자세히 설명되어 있다.

프로그램 종료 조건의 경우 아래에 자세히 설명되어 있다.

Newton-Raphson 방법과 Secant 방법을 이용한 비선형 방정식의 풀이 비교(실습1-1)

$$f_1(x) = x^2 - 4x + 4 - lnx = 0$$

비선형 방정식 $f_1(x)$를 Newton-Raphson방법과 Secant방법을 이용하여 풀어보았다.

Newton-Raphson 방법을 이용하든 Secant 방법을 이용하든 프로그램에서는 처음 설정한 초기값에서 방정식의 근을 찾기 위해 각 방법을 반복문을 이용해 반복적으로 수행한다. 이때 프로그램이 무한히 수행되는 것을 막기 위해 적절한 종료 조건을 걸어주었다.

현재 반복문을 통해 구한 x_{n+1} 이 주어진 방정식의 근이라고 판단하려면 충분히 많은 횟수의 근 찾기가 진행됐을 때 $|f(x_{n+1})|$이 0에 가까워야하고 직전에 구한 x_n 과 매우 작은 차이를 보일 경우 즉, 더 이상의 진전이 없다고 판단될 경우 x_{n+1} 충분히 f의 근에 가깝다고 판단할 수 있다.

위의 종료 조건을 정리하면 아래와 같다.

1. $|f(x_{n+1})| < \delta$
2. $n \geq N_{max}$
3. $|x_{n+1} - x_n| < \epsilon$

프로그램에서는 δ, N_{max}, ϵ을 각각 $0.000001, 50, 0.00001$ 설정하여 진행하였다.

Newton-Raphson

초기값 x_0

- $x_0 = 3.0$

결과 분석

초기값을 3.0으로 하고 주어진 비선형 방정식 f_1의 근을 구할 경우, 초기값 제외 총 3번의 반복 후, 위의 종료조건을 만족하고 프로그램이 종료됨을 확인할 수 있다.

i	x_{n+1}	$\lvert f(x_{n+1})\rvert$
0	3.000000000000000e+00	9.861228866810978e-02
1	3.059167373200866e+00	3.692745776079587e-03
2	3.057106054691600e+00	4.476150597731987e-06
3	3.057103549998436e+00	6.609823799408332e-12

프로그램 실행 동안의 x와 $|f(x)|$의 값

마지막 iteration에서 나온 값이자 프로그램이 구한 f_1의 근이라고 할 수 있는 $3.057103549998436e^0$의 $|f|$ 값을 보면 $6.609823799408332e^{-12}$로 프로그램에서 설정한 δ 보다 작은 값으로 매우 작은 0에 가까운 값이라고 분석할 수 있다. 또한 x_3과 x_2의 차이를 비교해보면 대략 0.0000035 로 프로그램의 종료 조건 ϵ보다 작은 차이를 보이는 것을 알 수 있다. 따라서 원하는 근을 정확히 찾고 있다고 판단할 수 있다.

Secant

초기값 x_0, x_1

- $x_0 = 2.0$
- $x_1 = 4.0$

결과분석

초기값을 2.0, 4.0으로 하고 방정식의 근을 구한 경우, 초기값 제외 총 8번의 반복 후, 위의 종료조건을 만족하고 프로그램이 종료되었음을 확인할 수 있다.

i	x_{n+1}	$\|f(x_{n+1})\|$
0	2.000000000000000e+00	6.931471805599453e-01
1	4.000000000000000e+00	2.613705638880109e+00
2	2.419218645889003e+00	7.077003413730973e-01
3	2.756039712206013e+00	4.421987159786218e-01
4	3.317022504001122e+00	5.354807316593719e-01
5	3.009768959371032e+00	8.222996682146211e-02
6	3.050670709725762e+00	1.145253099628074e-02
7	3.057289041659921e+00	3.315284194800316e-04
8	3.057102843928875e+00	1.261809842834083e-06
9	3.057103549917540e+00	1.379611980212303e-10

프로그램 실행 동안의 x와 $|f(x)|$의 값

Newton-Raphson 방법에서 분석한 방법과 마찬가지로 마지막 iteration의 x값과 $|f(x)|$의 값을 보면 $|x_9 - x_8| \approx 0.0000013$ 로 ϵ보다 작은 차이를 보이고 $|f(x_9)|$도 δ보다 작은 값을 가지는 것을 알 수 있다. 따라서 원하는 근을 정확히 찾고 있다고 판단할 수 있다.

두 방법의 수렴 속도 분석

Newton-Raphson 방법의 경우 이론상 근에 2차 수렴을 하고 Secant 방법의 경우 이론상 1.62의 속도로 근에 수렴한다. f_1의 방정식의 근을 찾는 과정에서 Newton-Raphson의 경우 3번의 반복, Secant 방법의 경우 8번의 반복을 수행하였다. 이 결과를 통해 두 방법 모두 빠른 속도로 근에 수렴하고 Newton-Raphson 방법이 Secant 방법보다 더 빠른 속도로 근에 수렴함을 알 수 있다.

이때 Secant 방법에서 Newton-Raphson에서 설정한 초기값 3.0과 유사한 x와 $|f(x)|$를 보이는 iteration이 존재한다. iteration 5번에서 유사함을 확인할 수 있다. 해당 iteration에서부터 4번의 반복문 수행 후 프로그램이 종료됨을 알 수 있다. 따라서 근과 유사한 오차를 보이는 순간부터 Newton-Raphson 방법이 Secant 방법보다 $\frac{4}{3}$만큼 빠르게 오차 감소를 보이는 것을 알 수 있고 이는 이론상 보이는 속도차와 유사하다고 할 수 있다.

임의의 초기값에 대해 두 방법의 수렴속도 분석

위의 방정식 f_1에서 초기값을 변경한 뒤 다시 Newton-Raphson 방법과 Secant 방법을 이용해 방정식을 풀이하면 아래와 같은 결과가 나온다.

초기값 설정

Newton-Raphson

- $x_0 = 2.7$

Secant

- $x_0 = 1.7$
- $x_1 = 3.7$

결과: Newton-Raphson

초기값을 2.7으로 하고 주어진 비선형 방정식 f_1의 근을 구할 경우, 초기값 제외 총 5번의 반복 후, 위의 종료조건을 만족하고 프로그램이 종료됨을 확인할 수 있다.

i	xn1	| f(xn1) |
0	2.700000000000000e+00	5.032517730102832e-01
1	3.188769707599915e+00	2.535382468996263e-01
2	3.065927769928115e+00	1.585178893750805e-02
3	3.057148971254527e+00	8.117451094169503e-05
4	3.057103551210869e+00	2.173347279921245e-09
5	3.057103549994738e+00	0.000000000000000e+00

마지막 iteration에서 $3.057103549994738e+00$의 값을 가짐을 확인 할 수 있고 $|f(x)|$이 0인 것을 통해 해당 프로그램으로 정확한 근을 구해냈음을 알 수 있다.

초기값을 3.0으로 했던 기존의 수행 결과보다 좀 더 많은 반복문이 소요되었지만 종료조건을 만족하면서 매우 정확한 근을 찾아냈음을 알 수 있다.

결과: Secant

초기값을 1.7, 3.7으로 하고 방정식의 근을 구한 경우, 초기값 제외 총 9번의 반복 후, 위의 종료조건을 만족하고 프로그램이 종료되었음을 확인할 수 있다.

i	xn1	| f(xn1) |
0	1.700000000000000e+00	4.406282510621705e-01
1	3.700000000000000e+00	1.581667180349822e+00
2	2.135770406457892e+00	7.403938244376536e-01
3	2.634528160312509e+00	5.660781134344021e-01
4	4.254209178953081e+00	3.633550134609810e+00
5	2.852848907113028e+00	3.209668531636551e-01
6	2.966589775591207e+00	1.531172751165957e-01
7	3.070347509470592e+00	2.385304056960025e-02
8	3.056362466690744e+00	1.323811448244561e-03
9	3.057097807221920e+00	1.026287507888135e-05
10	3.057103552504711e+00	4.485571691859036e-09

마지막 iteration에서 $3.057103552504711e+00$의 값을 가짐을 확인 할 수 있고 해당 iteration에서 함수값이 $4.485571691859036e-09$을 만족함을 통해 종료 조건인 δ보다 작은 함수값을 가지고 $|x_{10}-x_9|<\epsilon$을 만족함을 통해 프로그램이 원하는 근을 정확하게 찾고 있다고 판단할 수 있다.

두 방법의 수렴 속도

Newton-Raphson과 Secant 방법에서 각각 5번, 9번의 반복 후 프로그램을 종료한 것을 통해 두 방법 모두 이론상 확인한 것처럼 빠른 속도로 근에 수렴함을 알 수 있고 두 방법 중 Newton-Raphson방법이 더 빠른 속도로 근에 가까워짐을 확인할 수 있다.

부동 소수점 연산의 정밀도에 따른 근의 차이(실습 1-4)

$$f_1(x)=lnx-1=0$$

위 비선형 방정식 f_1의 경우 근이 e로 알려져 있다. 위 방정식의 근을 프로그램을 이용해 Newton-Raphson 방법으로 구하려 할 때 동일한 초기값에 대해 double precision 방법과 single precision 방법 모두를 이용해 근을 구해 부동 소수점 연산의 정밀도에 따른 근의 차이를 분석해보았다.

초기값 설정

초기값은 실제 근인 $e=2.71828182845904523536028747135 2...$과 유사한 2.6으로 하여 프로그램을 실행하였다.

- $x_0 = 2.6$

double precision 결과

double precision의 경우 프로그램에서 `double` 타입으로 모든 연산을 처리하였다.

i	x_{n+1}	$\lvert f(x_{n+1})\rvert$
0	2.600000000000000e+00	4.448855497256365e-02
1	2.715670242928665e+00	9.612104402638710e-04
2	2.718280573518483e+00	4.616669394552630e-07

프로그램 실행 동안의 x와 $|f(x)|$의 값

single precision 결과

single precision의 경우 프로그램에서 `float` 타입으로 모든 연산을 처리하였다.

i	x_{n+1}	$\lvert f(x_{n+1})\rvert$
0	2.599999904632568e+00	4.448860883712769e-02
1	2.715670347213745e+00	9.611845016479492e-04
2	2.718280553817749e+00	4.768371582031250e-07

프로그램 실행 동안의 x와 $|f(x)|$의 값

double precision, single precision 연산의 정밀도 차이 분석

프로그램에서 `double` 타입은 `float` 타입보다 두 배 더 많은 부동 소수점 연산 정밀도를 가진다. 프로그램 실행 결과를 비교했을 때 알 수 있는 정보는 아래와 같다.

1. double precision이 초기값을 좀 더 정확하게 계산해 프로그램에 넣는다.
    a. 결과 표에서 iteration 0은 프로그램 실행 시 입력한 초기값으로 2.6을 입력하였다. 하지만 single precision의 경우 정확히 2.6이 아닌 2.6과 유사한 $2.599999904632568e+00$가 초기값이 되어 계산이 진행되었음을 알 수 있다.
2. 프로그램 종료조건을 만족하는 x, 즉 Newton-Raphson 방법이 구해낸 방정식 f_1의 근의 정확도를 비교했을 때 double precision이 좀 더 나은 정확도를 보임을 확인할 수 있다.

a. 실제로 알려진 근 e와 프로그램을 이용해 구한 근 $2.7182805735184 83e+00$과 $2.7182805538177 49e+00$을 비교했을 때 double prescision의 경우 대략 0.000001254940562의 오차를 가지고 single precision의 경우 대략 0.000001274641296 의 오차를 가져 미세하게 double precision이 더 정확한 근을 구해냈음을 알 수 있다.

b. 또한 $|f(x)|$도 double preicision의 경우 같은 e^{-7}에 대해 4.6xxx로 표현되고 single precision의 경우 4.7xxx 로 표현되므로 double precision이 좀 더 0에 가까운 근을 찾아냈다고 분석할 수 있다.

Bisection 방법을 이용한 비선형 방정식의 풀이 분석 (숙제1)

Bisection 방법을 이용해 근을 구할 때 다음 구간 설정 시 $[a_0, \frac{a_0+b_0}{2}]$ 또는 $[\frac{a_0+b_0}{2}, b_0]$ 중 알맞은 구간을 선택해야한다. 이때 구간 선택 로직은 아래와 같다.

구간 선택 Logic

- 구간 $[a_n, b_n]$에 항상 근이 존재하려면 $f(a_n)f(b_n) < 0$ 을 만족해야한다.
- $\frac{a_n+b_n}{2}$ 에 대해 $f(a_n)f(\frac{a_n+b_n}{2}) < 0$ 또는 $f(\frac{a_n+b_n}{2})f(b_n) < 0$ 를 판단해 부등식을 만족하는 값으로 다음 구간을 정한다.

위 로직을 이용해 프로그램에 구현한 Bisection 방법에 대해 Bisection 방법을 이용한 비선형방정식의 풀이를 분석하고자 한다.

$$f_1(x) = x^2 - 4x + 4 - lnx = 0$$
$$f_2(x) = x + 1 - 2sin\pi x = 0$$
$$f_3(x) = x^4 - 11.0x^3 + 42.35x^2 - 66.55x + 35.184 = 0$$

위 세 비선형 방정식에 대해 Bisection 방법을 이용해 근을 구한 결과는 아래와 같다.

초기값 설정

f_1

- $a_0 = 2.0$
- $b_0 = 4.0$

f_2

- $a_0 = 0.5$
- $b_0 = 1.0$

f_3

$[1.02, 1.48]$

- $a_0 = 1.02$
- $b_0 = 1.48$

$[1.95, 2.37]$

- $a_0 = 1.95$
- $b_0 = 2.37$

$[3.11, 3.73]$

- $a_0 = 3.11$
- $b_0 = 3.73$

$[3.83, 4.61]$

- $a_0 = 3.83$
- $b_0 = 4.61$

결과 분석

f_1

초기 구간을 $[2.0, 4.0]$으로 설정하고 Bisection 방법을 실행하면 결과 표에서 볼 수 있듯이 초기 구간을 이용해 구한 x_0 포함 총 20번의 반복 후 프로그램의 종료 조건을 만족한다.

초기 구간을 $[2.0, 4.0]$으로 설정하고 Bisection 방법을 실행하면 결과 표에서 볼 수 있듯이 초기 구간을 이용해 구한 x_0 포함 총 20번의 반복 후 프로그램의 종료 조건을 만족한다.

i	xn1	| f(xn1) |
0	3.000000000000000e+00	9.861228866810978e-02
1	3.500000000000000e+00	9.972370315046319e-01
2	3.250000000000000e+00	3.838450036583538e-01
3	3.125000000000000e+00	1.261907168116352e-01
4	3.062500000000000e+00	9.674674129154681e-03
5	3.031250000000000e+00	4.549851320365628e-02
6	3.046875000000000e+00	1.816920957907486e-02
7	3.054687500000000e+00	4.311573409498504e-03
8	3.058593750000000e+00	2.665476027821967e-03
9	3.056640625000000e+00	8.270675343551304e-04
10	3.057617187500000e+00	9.181995684575117e-04
11	3.057128906250000e+00	4.531484340919434e-05
12	3.056884765625000e+00	3.909391393928097e-04
13	3.057006835937500e+00	1.728278464081523e-04
14	3.057067871093750e+00	6.376042609557153e-05
15	3.057098388671875e+00	9.223772491129267e-06
16	3.057113647460938e+00	1.804529017213063e-05
17	3.057106018066406e+00	4.410697518775208e-06
18	3.057102203369141e+00	2.406552816580643e-06
19	3.057104110717773e+00	1.002068518385357e-06

결과를 보면 마지막 iteration 값인 $3.057104110717773e+00$ 을 방정식의 근으로 반환하였고 이는 앞의 Newton-Raphson 방법과 Secant 방법으로 구한 근과 유사함을 알 수 있다.

x_{19}와 x_{18}의 차이가 ϵ 보다 작고 $|f(x_{19})|$또한 δ보다 작은 충분히 0에 가까운 값이므로 적절한 초기구간을 사용하여 올바르게 근에 수렴한다고 판단할 수 있다.

f_2

초기 구간을 $[0.5, 1.0]$으로 설정하고 Bisection 방법을 실행하면 결과 표에서 볼 수 있듯이 초기 구간을 이용해 구한 x_0 포함 총 19번의 반복 후 프로그램의 종료 조건을 만족한다.

i	xn1	| f(xn1) |
0	7.500000000000000e-01	3.357864376269049e-01
1	6.250000000000000e-01	2.227590650225735e-01
2	6.875000000000000e-01	2.456077539490931e-02
3	6.562500000000000e-01	1.075925286967101e-01
4	6.718750000000000e-01	4.358222000054424e-02
5	6.796875000000000e-01	1.001963049941446e-02
6	6.835937500000000e-01	7.144338890323620e-03
7	6.816406250000000e-01	1.469329873796887e-03
8	6.826171875000000e-01	2.829599108000957e-03
9	6.821289062500000e-01	6.781563090836329e-04
10	6.818847656250000e-01	3.960816035961656e-04
11	6.820068359375000e-01	1.409136779404463e-04
12	6.819458007812500e-01	1.276148853435188e-04
13	6.819763183593750e-01	6.641666146611769e-06
14	6.819610595703125e-01	6.048854219597999e-05
15	6.819686889648438e-01	2.692392116654396e-05
16	6.819725036621094e-01	1.014124829468166e-05
17	6.819744110107422e-01	1.749821270324858e-06
18	6.819753646850586e-01	2.445914888848932e-06

결과를 보면 마지막 iteration 값인 $6.819753646850586e-01$ 을 방정식의 근으로 반환하였고 이는 앞의 Newton-Raphson 방법과 Secant 방법으로 구한 근과 유사함을 알 수 있다.

x_{18}와 x_{17}의 차이가 ϵ 보다 작고 $|f(x_{18})|$또한 δ보다 작은 충분히 0에 가까운 값이므로 적절한 초기구간을 사용하여 올바르게 근에 수렴한다고 판단할 수 있다.

f_3

$[1.02, 1.48]$

초기 구간을 $[1.02, 1.48]$으로 설정하고 Bisection 방법을 실행하면 결과 표에서 볼 수 있듯이 초기 구간을 이용해 구한 x_0 포함 총 19번의 반복 후 프로그램의 종료 조건을 만족한다.

i	xn1	| f(xn1) |
0	1.250000000000000e+00	9.201937500000028e-01
1	1.135000000000000e+00	2.634867243750065e-01
2	1.077500000000000e+00	1.864986219140619e-01
3	1.106250000000000e+00	4.939418792725547e-02
4	1.091875000000000e+00	6.576846241470946e-02
5	1.099062500000000e+00	7.498578681186530e-03
6	1.102656250000000e+00	2.111902525615506e-02
7	1.100859375000000e+00	6.853143165336917e-03
8	1.099960937500000e+00	3.119734348402403e-04
9	1.100410156250000e+00	3.273269151200964e-03
10	1.100185546875000e+00	1.481319153988636e-03
11	1.100073242187500e+00	5.848407115607301e-04
12	1.100017089843750e+00	1.364756048616300e-04
13	1.099989013671875e+00	8.773842294118595e-05
14	1.100003051757813e+00	2.437121394649466e-05
15	1.099996032714844e+00	3.168294875166566e-05
16	1.099999542236328e+00	3.655703466165505e-06
17	1.100001296997070e+00	1.035779622071686e-05
18	1.100000419616699e+00	3.351056619749215e-06

결과를 보면 마지막 iteration 값인 $1.100000419616699e+00$ 을 방정식의 근으로 반환하였고 이는 앞의 Newton-Raphson 방법과 Secant 방법으로 구한 근과 유사함을 알 수 있다.

x_{18}와 x_{17}의 차이가 ϵ 보다 작고 $|f(x_{18})|$또한 δ보다 작은 충분히 0에 가까운 값이므로 적절한 초기구간을 사용하여 올바르게 근에 수렴한다고 판단할 수 있다.

$[1.95, 2.37]$

초기 구간을 $[1.95, 2.37]$으로 설정하고 Bisection 방법을 실행하면 결과 표에서 볼 수 있듯이 초기 구간을 이용해 구한 x_0 포함 총 16번의 반복 후 프로그램의 종료 조건을 만족한다.

i	xn1	| f(xn1) |
0	2.160000000000000e+00	1.082726399999885e-01
1	2.265000000000000e+00	1.673314256250151e-01
2	2.212500000000000e+00	3.308166503905596e-02
3	2.186250000000000e+00	3.682551073973883e-02
4	2.199375000000000e+00	1.664222118954228e-03
5	2.205937500000000e+00	1.576250851211825e-02
6	2.202656250000000e+00	7.062358964738280e-03
7	2.201015625000000e+00	2.702343338434332e-03
8	2.200195312500000e+00	5.198757007818244e-04
9	2.199785156250000e+00	5.719698916593075e-04
10	2.199990234375000e+00	2.599620915333389e-05
11	2.200092773437500e+00	2.469524744768137e-04
12	2.200041503906250e+00	1.104813139605199e-04
13	2.200015869140625e+00	4.224334762881199e-05
14	2.200003051757812e+00	8.123768019174804e-06
15	2.199996643066406e+00	8.936170893036888e-06

결과를 보면 마지막 iteration 값인 $2.199996643066406e+00$ 을 방정식의 근으로 반환하였고 이는 앞의 Newton-Raphson 방법과 Secant 방법으로 구한 근과 유사함을 알 수 있다.

x_{16}와 x_{15}의 차이가 ϵ 보다 작고 $|f(x_{16})|$또한 δ보다 작은 충분히 0에 가까운 값이므로 적절한 초기구간을 사용하여 올바르게 근에 수렴한다고 판단할 수 있다.

$[3.11, 3.73]$

초기 구간을 $[3.11, 3.73]$으로 설정하고 Bisection 방법을 실행하면 결과 표에서 볼 수 있듯이 초기 구간을 이용해 구한 x_0 포함 총 18번의 반복 후 프로그램의 종료 조건을 만족한다.

i	xn1	| f(xn1) |
0	3.420000000000000e+00	3.328550399999628e-01
1	3.265000000000000e+00	9.159492562512384e-02
2	3.342500000000000e+00	1.151484155859279e-01
3	3.303750000000000e+00	9.999399411576348e-03
4	3.284375000000000e+00	4.129000711439090e-02
5	3.294062500000000e+00	1.576250851210403e-02
6	3.298906250000000e+00	2.910112113063690e-03
7	3.301328125000000e+00	3.537597931305925e-03
8	3.300117187500000e+00	3.119697382700792e-04
9	3.299511718750000e+00	1.299515944999996e-03
10	3.299814453125000e+00	4.938841097796853e-04
11	3.299965820312500e+00	9.098491450032498e-05
12	3.300041503906250e+00	1.104854825868529e-04
13	3.300003662109375e+00	9.748551370591940e-06
14	3.299984741210937e+00	4.061861475435080e-05
15	3.299994201660156e+00	1.543514000701407e-05
16	3.299998931884765e+00	2.843321375678443e-06
17	3.300001296997070e+00	3.452608204668195e-06

결과를 보면 마지막 iteration 값인 $3.300001296997070e+00$ 을 방정식의 근으로 반환하였고 이는 앞의 Newton-Raphson 방법과 Secant 방법으로 구한 근과 유사함을 알 수 있다.

x_{17}와 x_{16}의 차이가 ϵ 보다 작고 $|f(x_{17})|$또한 δ보다 작은 충분히 0에 가까운 값이므로 적절한 초기구간을 사용하여 올바르게 근에 수렴한다고 판단할 수 있다.

$[3.83, 4.61]$

초기 구간을 $[3.83, 4.61]$으로 설정하고 Bisection 방법을 실행하면 결과 표에서 볼 수 있듯이 초기 구간을 이용해 구한 x_0 포함 총 19번의 반복 후 프로그램의 종료 조건을 만족한다.

i	xn1	| f(xn1) |
0	4.220000000000001e+00	1.043677439999946e+00
1	4.415000000000001e+00	1.228070756250830e-01
2	4.317500000000001e+00	5.719134905858638e-01
3	4.366250000000001e+00	2.546190068334724e-01
4	4.390625000000001e+00	7.370435628904914e-02
5	4.402812500000001e+00	2.256605607454532e-02
6	4.396718750000002e+00	2.606099208150425e-02
7	4.399765625000001e+00	1.870987694751136e-03
8	4.401289062500002e+00	1.031658425427651e-02
9	4.400527343750001e+00	4.215069552465422e-03
10	4.400146484375002e+00	1.170109841105216e-03
11	4.399956054687502e+00	3.509215618962003e-04
12	4.400051269531252e+00	4.094734636694852e-04
13	4.400003662109377e+00	2.924578399898792e-05
14	4.399979858398440e+00	1.608454305070950e-04
15	4.399991760253908e+00	6.580170855130518e-05
16	4.399997711181642e+00	1.827843367863125e-05
17	4.400000686645510e+00	5.483557409036166e-06
18	4.399999198913576e+00	6.397467466001672e-06

결과를 보면 마지막 iteration 값인 $4.399999198913576e+00$ 을 방정식의 근으로 반환하였고 이는 앞의 Newton-Raphson 방법과 Secant 방법으로 구한 근과 유사함을 알 수 있다.

x_{18}와 x_{17}의 차이가 ϵ 보다 작고 $|f(x_{18})|$또한 δ보다 작은 충분히 0에 가까운 값이므로 적절한 초기구간을 사용하여 올바르게 근에 수렴한다고 판단할 수 있다.

수렴 속도 비교

위의 두 비선형 방정식 풀이 방법 Newton-Raphson 방법, Secant 방법 , 그리고 Bisection 방법은 이론상 각각 다음과 같은 수렴 속도를 보인다.

- Newton-Raphson: $\epsilon_{n+1} \approx c\epsilon_n^2, \quad c > 0$
  - 근이 단근(simple root)일 경우 성립
- Secant: $\epsilon_{n+1} \approx c\epsilon_n^{1.62}, \quad c > 0$
- Newton-Raphson: $\epsilon_{n+1} \approx \frac{1}{2}\epsilon_n$

f_1, f_2, f_3의 세 가지 방법에 대한 근에 수렴 속도를 분석하기 위해 위에서 보인 각 함수의 각 방법에 대한 결과를 제외한 나머지 결과를 아래에 정리해였다.

f_2: Newton-Raphson

초기값을 0.75로 하여 Newton-Raphson방법을 수행할 경우 초기값 제외 3번의 반복문 수행 후 프로그램을 종료한다.

i	x_{n+1}	$\lvert f(x_{n+1})\rvert$
0	7.500000000000000e-01	3.357864376269049e-01
1	6.883072460969516e-01	2.819126095597957e-02
2	6.820480690939008e-01	3.223569164387818e-04
3	6.819748188604164e-01	4.453119828440322e-08

f_2: Secant

초기값을 $0.5, 1.0$으로 하여 Secant방법을 수행할 경우 초기값 제외 6번의 반복문 수행 후 프로그램을 종료한다.

i	x_{n+1}	$\lvert f(x_{n+1})\rvert$
0	5.000000000000000e-01	5.000000000000000e-01
1	1.000000000000000e+00	2.000000000000000e+00
2	6.000000000000001e-01	3.021130325903070e-01
3	6.524931709804664e-01	1.223535022158933e-01
4	6.882226997487054e-01	2.781052374266890e-02
5	6.816055562499178e-01	1.623411657736940e-03
6	6.819705209666883e-01	1.886410134188665e-05
7	6.819748117285146e-01	1.315406050750312e-08

f_3: Newton-Raphson

$[1.02, 1.48]$

초기값을 구간의 중간값인 1.25로 하여 Newton-Raphson방법을 수행할 경우 초기값 제외 4번의 반복문 수행후 프로그램을 종료한다.

i	x_{n+1}	$\lvert f(x_{n+1})\rvert$
0	1.250000000000000e+00	9.201937500000028e-01
1	1.042046610169491e+00	5.088145931496086e-01
2	1.095070234545689e+00	3.969336566746762e-02
3	1.099959957849027e+00	3.197979590012778e-04
4	1.099999997327961e+00	2.133891285893696e-08

$[1.95, 2.37]$

초기값을 구간의 중간값인 2.16로 하여 Newton-Raphson방법을 수행할 경우 초기값 제외 2번의 반복문 수행후 프로그램을 종료한다.

i	x_{n+1}	$\lvert f(x_{n+1})\rvert$
0	2.160000000000000e+00	1.082726399999885e-01
1	2.199400753425052e+00	1.595628415664407e-03
2	2.199999837218690e+00	4.333238550202623e-07

$[3.11, 3.73]$

초기값을 구간의 중간값인 3.42로 하여 Newton-Raphson방법을 수행할 경우 초기값 제외 3번의 반복문 수행후 프로그램을 종료한다.

i	x_{n+1}	$\lvert f(x_{n+1})\rvert$
0	3.420000000000000e+00	3.328550399999628e-01
1	3.303227113773005e+00	8.603104076250645e-03
2	3.300004664519575e+00	1.241697736276137e-05
3	3.300000000009917e+00	2.639666263348772e-11

$[3.83, 4.61]$

초기값을 구간의 중간값인 4.22로 하여 Newton-Raphson방법을 수행할 경우 초기값 제외 4번의 반복문 수행후 프로그램을 종료한다.

i	x_{n+1}	$\lvert f(x_{n+1})\rvert$
0	4.220000000000000e+00	1.043677439999890e+00
1	4.493744853894648e+00	8.711305502691360e-01
2	4.412015995526431e+00	9.789296460457564e-02
3	4.400234056965293e+00	1.869908166845846e-03
4	4.400000091254408e+00	7.287577048487037e-07

f_3 : Secant

$[1.02, 1.48]$

초기값을 $1.02, 1.48$으로 하여 Secant방법을 수행할 경우 초기값 제외 7번의 반복문 수행 후 프로그램을 종료한다.

i	xn1	| f(xn1) |
0	1.020000000000000e+00	7.274841599999959e-01
1	1.480000000000000e+00	1.454019840000001e+00
2	1.173399999999999e+00	5.170449001636470e-01
3	1.004210879085131e+00	8.929836181502111e-01
4	1.111359856133397e+00	8.901186301301323e-02
5	1.101647459029790e+00	1.312051236190115e-02
6	1.099968326623156e+00	2.529569403080245e-04
7	1.100000087132441e+00	6.958395815104268e-07
8	1.100000000004599e+00	3.671374315672438e-11

$[1.95, 2.37]$

초기값을 $1.95, 2.37$으로 하여 Secant방법을 수행할 경우 초기값 제외 5번의 반복문 수행 후 프로그램을 종료한다.

i	xn1	| f(xn1) |
0	1.950000000000000e+00	7.028437499999640e-01
1	2.370000000000000e+00	4.075976099999892e-01
2	2.215835176564386e+00	4.184115640978092e-02
3	2.198199303220419e+00	4.797365407192444e-03
4	2.200013377196391e+00	3.560988029249756e-05
5	2.200000010904762e+00	2.902846318875163e-08
6	2.199999999999939e+00	1.918465386552271e-13

$[3.11, 3.73]$

초기값을 $3.11, 3.73$으로 하여 Secant방법을 수행할 경우 초기값 제외 5번의 반복문 수행 후 프로그램을 종료한다.

i	xn1	| f(xn1) |
0	3.110000000000000e+00	4.483124100000353e-01
1	3.730000000000000e+00	1.159285589999904e+00
2	3.282900000000020e+00	4.515546893944844e-02
3	3.299662140424397e+00	8.992439850104006e-04
4	3.300002731018557e+00	7.269980322632819e-06
5	3.299999999580309e+00	1.117207659717678e-09

$[3.83, 4.61]$

초기값을 $3.83, 4.61$으로 하여 Secant방법을 수행할 경우 초기값 제외 8번의 반복문 수행 후 프로그램을 종료한다.

i	xn1	| f(xn1) |
0	3.830000000000000e+00	1.344314789999963e+00
1	4.610000000000000e+00	2.327098409999969e+00
2	4.115602703667348e+00	1.339934858587121e+00
3	4.296255597389961e+00	6.925022448218598e-01
4	4.489484254360017e+00	8.259934490623735e-01
5	4.384376544492151e+00	1.215451563029362e-01
6	4.397859196230239e+00	1.703552336663705e-02
7	4.400056926930033e+00	4.546615977645274e-04
8	4.399999796387763e+00	1.626046632452471e-06
9	4.399999999980666e+00	1.542659333608754e-10

이론상과 마찬가지로 반복문의 수행 횟수를 기반으로 오차 감소 속도를 예측했을 때 Newton-Raphson, Secant, Bisection 순서로 반복문의 횟수가 증가하므로 상대적으로 Newton-Raphson과 Secant 방법이 빠른 근 수렴속도를 가짐을 알 수 있고 모든 함수에서 Newton-Raphson, Secant, Bisection 순서로 빠른 오차 감소 속도를 가짐을 알 수 있다.

각도 α구하기(숙제2)

$$f(\alpha) = Asin\alpha cos\alpha + Bsin^2\alpha - Ccos\alpha - Esin\alpha = 0$$

$$\begin{gathered} A = lsin\beta_1, \\ B = lcos\beta_1, \\ C = (h + 0.5D)sin\beta_1 - 0.5Dtan\beta_1, \\ E = (h + 0.5D)cos\beta_1 - 0.5D \end{gathered}$$

$$l = 89, h = 49, D = 55$$

위와 같은 비선형 방정식을 만족하는 α에 대해 Newton-Raphson방법으로 α를 추정하면 아래와 같은 과정을 가진다.

초기값 설정

- α_0 : 33.0

과정

Newton-Raphson 방법은 주어진 x에 대해 x의 f에 대한 접선의 방정식이 x축과 만나는 점을 다음 근으로 추정해 실제 근과의 오차를 줄여나간다. 위의 과정을 식으로 표현하면 아래와 같다.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

따라서 방정식의 근에 매우 가까이 가게 될수록 x_n과 x_{n+1}의 차이는 0에 수렴하게된다.

프로그램에서는 초기값 x_0을 입력받아 위의 식에 대입해 x_1을 구한 뒤, x_1이 프로그램의 종료 조건을 만족하는지 확인한 후, 아닐 경우 해당 식을 반복한다. 해당 식은 아래와 같이 구현되어 있다.

```
for(n; n<=Nmax; n++) {
    x1 = x0 - (_f(x0) / _fp(x0));
    if (fabs(_f(x1)) < DELTA  && fabs(x1 - x0) < EPSILON) {
      break;
    }
    x0 = x1;
  }
```

위 프로그램에서는 프로그램의 반복문이 무한히 수행되는 것을 막기 위해 최대 `Nmax` 만큼의 수행 횟수 제한을 두었다.

Newton-Raphson 방법을 사용하기 위해서는 함수 $f'(\alpha)$의 식을 알아야 하는데 이는 아래와 같다.

$$f'(\alpha) = A(cos^2\alpha - sin^2\alpha) + 2Bsin\alpha cos\alpha + Csin\alpha - Ecos\alpha = 0$$

위의 상수들을 변수로 설정한 뒤, $f(\alpha)$와 $f'(\alpha)$의 계산 결과를 반환하도록 `_f`와 `_fp` 의 프로그램해 처음 설정한 초기값을 넣어 방정식의 해를 구하면 아래와 같은 결과가 나온다.

결과

초기값을 33.0으로 하고 주어진 비선형 방정식 $f(\alpha)$의 근을 구할 경우, 초기값 제외 총 4번의 반복 후, 위의 종료조건을 만족하고 프로그램이 종료됨을 확인할 수 있다.

i	xn1	| f(xn1) |
0	3.300000000000000e+01	3.434111718478194e+01
1	3.242490719716647e+01	3.195993587951808e+00
2	3.248531333657944e+01	1.762793644939915e-01
3	3.248230378537768e+01	3.869300062593339e-04
4	3.248229715028958e+01	1.893502243888179e-09

프로그램 수행 결과 $3.248229715028958e+01$을 방정식의 근으로 반환하였고 이는 x_4와 x_3의 차이가 ϵ보다 작고 $|f(x_4)|$의 값이 δ보다 작으므로 프로그램이 정확한 근을 구했다고 판단할 수 있다.