**고급소프트웨어실습1**

**4주차 과제**

**20161663 허재성**

**실습 문제 1-1**

의 근을 구하기 위해 Newton-Raphson 방법과 Secant 방법을 이용해 각각 근을 구한다.

Newton\_Raphson 방법으로 근을 구하기 위해 초기값 x0를 3.0으로 설정 후 근을 구해보면 다음과 같다.

|  |
| --- |
| n xn1 |f(xn1)|  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 |

Secant 방법으로 근을 구하기 위해 초기값 x0, x1을 각각 2.0, 4.0으로 설정 후 근을 구해보면 다음과 같다.

|  |
| --- |
| n xn1 |f(xn1)|  0 2.000000000000000e+00 6.931471805599453e-01  1 4.000000000000000e+00 2.613705638880109e+00  2 2.419218645889003e+00 7.077003413730971e-01  3 2.756039712206013e+00 4.421987159786225e-01  4 3.317022504001125e+00 5.354807316593784e-01  5 3.009768959371032e+00 8.222996682146388e-02  6 3.050670709725762e+00 1.145253099627896e-02  7 3.057289041659920e+00 3.315284194784773e-04  8 3.057102843928875e+00 1.261809842834083e-06  9 3.057103549917540e+00 1.379611980212303e-10 |

(ii)

구한 값이 방정식의 실제 근이 맞는지 확인하기 위해 구한 값 xn1을 함수 f1(x)에 대입한 값을 확인한다. 이 때, f1(xn1)의 절댓값이 충분히 작은 양의 수 delta보다 작으면 방정식 f1(x) = 0의 근이라고 생각할 수 있다. delta를 0에 가깝게 할수록 더 정밀한 근을 얻을 수 있다. 또한 n번째 반복에서 구한 근 xn과 n+1번째 x(n+1)을 비교하여 그 차이가 충분히 작다면 더 이상 반복해도 이전에 구한 근과 큰 차이가 없으므로 해당 값이 근이라고 생각할 수 있다. 그 외에도 충분한 횟수를 반복하여 구한 값을 근으로 생각할 수 있다.

(iii)

위 방정식의 실제 근 a는 3.0571035로 알려져 있다. 실제 근 a와 Newton-Rapshon 방법으로 구한 근의 오차를 라 할 때 이론 상으로 수식

의 값이 n이 커질수록 특정 값 c(c는 양의 상수)에 수렴해야 한다.

|  |  |
| --- | --- |
| n |  |
| 0 | 0.632932732 |
| 1 | 0.599753038 |
| 2 | 7660.893167 |

n이 0, 1일 때는 그럭저럭 비슷한 값을 가지지만 n=2일 때의 값과는 차이가 매우 크다.

반복문의 조건 중 n <= Nmax 만을 남겨서 다시 확인해보면 n = 5 이후로 근의 값이 3.057103549994738e+00에서 변하지 않으므로 수식의 값도 일정한 값 20002108.35가 나온다. Nmax(50)번의 반복 중 n=5에 근이 일정하게 나오고 위 수식이 일정 값에 수렴하는 것을 통해 근에 매우 빠르게 수렴하는 것을 알 수 있다.

실제 근 a와 Secant 방법으로 구한 근의 오차를 라 할 때 이론 상으로 수식

의 값이 n이 커질수록 특정 값 c(c는 양의 상수)에 수렴해야 한다.

|  |  |
| --- | --- |
| n |  |
| 0 | 0.861774534 |
| 1 | 0.701633913 |
| 2 | 0.623700383 |
| 3 | 1.817235572 |
| 4 | 0.419894853 |
| 5 | 0.900763464 |
| 6 | 0.658922628 |
| 7 | 0.727906837 |
| 8 | 518.5524267 |
| 9 | 33709.63758 |
| 10 | 33625.36295 |

Secant 방법의 경우 n이 커질수록 미분 계수를 대신 하는 평균 변화율을 구할 때 xn과 xn+1이 같아져 미분 계수를 구할 수 없게 된다.(0으로 나누게 된다.) 다른 조건이 없을 경우 0으로 나누기 전까지 11회 반복한다. n=9일 때와 10일 때 값이 비슷하다. n이 커질 때 수식의 값이 수렴할 것으로 예측 가능하다.

(iv)

주어진 초기값 외 다른 초기값을 사용하여 빠르게 수렴하는 지 확인해본다.

실제 근을 약 3.0571035라 했을 때 실제 근에서 먼 초기값을 입력해본다.

이를 위해 Newton-Raphson 방법에서 초기값을 4로, Secant 방법에서 초기값을 4, 5로 놓고 결과를 확인했다.

Newton-Rapshon 방법으로 구한 근

|  |
| --- |
| n xn1 |f(xn1)|  0 4.000000000000000e+00 2.613705638880109e+00  1 3.303011829631971e+00 5.030051002775475e-01  2 3.084624418770029e+00 4.998022416721226e-02  3 3.057535751137745e+00 7.725834922500674e-04  4 3.057103660055339e+00 1.966893221627686e-07  5 3.057103549994745e+00 1.199040866595169e-14 |

Secant로 구한 근

|  |
| --- |
| n xn1 |f(xn1)|  0 4.000000000000000e+00 2.613705638880109e+00  1 5.000000000000000e+00 7.390562087565899e+00  2 3.452839819040576e+00 8.715465113834024e-01  3 3.245995389462433e+00 3.750824618453470e-01  4 3.089722809529695e+00 5.941442037725020e-02  5 3.060309490669859e+00 5.740164412881832e-03  6 3.057163899116693e+00 1.078537581646888e-04  7 3.057103663825804e+00 2.034275217166481e-07  8 3.057103549998788e+00 7.237321852926470e-12 |

Newton-Rapshon의 경우 초기값이 구하려는 근과 멀어져서 반복 횟수가 늘어나 근을 구하는 속도가 조금 떨어졌다. Secant의 경우 오히려 반복 횟수가 1 줄어들어 결론을 내리기 어렵다. 초기값을 근에서 좀 더 먼 값으로 설정해본다.

초기값을 Newton-Raphson 방법에선 10으로 주고 난 다음결과를 확인해본다.

|  |
| --- |
| n xn1 |f(xn1)|  0 1.000000000000000e+01 6.169741490700596e+01  1 6.119659439810946e+00 1.516008745213728e+01  2 4.242461031315675e+00 3.583487944196867e+00  3 3.399130601610467e+00 7.340467468476404e-01  4 3.105988941611133e+00 8.987937457720774e-02  5 3.058434200951909e+00 2.379871778367848e-03  6 3.057104592120599e+00 1.862384363393588e-06  7 3.057103549995379e+00 1.143973804573761e-12 |

초기값을 Secant 방법에서 10, 11로 설정 후 근을 구해봤다.

|  |
| --- |
| n xn1 |f(xn1)|  0 1.000000000000000e+01 6.169741490700596e+01  1 1.100000000000000e+01 7.860210472720163e+01  2 6.350278203076087e+00 1.707642182070596e+01  3 5.059750296642711e+00 7.740754744213042e+00  4 3.989697141103637e+00 2.575179389780408e+00  5 3.456246626293293e+00 8.804710264202393e-01  6 3.179097039689378e+00 2.336726223167556e-01  7 3.078969596169310e+00 3.958039514378764e-02  8 3.058551036204362e+00 2.589010010164872e-03  9 3.057121950595606e+00 3.288407217172384e-05  10 3.057103565682093e+00 2.803487375580005e-08 |

반복 횟수가 증가한 것을 알 수 있다. 예시로 주어진 초기값과 다른 초기값을 사용하면 결과를 도출하는 시간(반복 횟수)이 달라질 수 있다. 근에서 먼 초기값을 잡을수록 근에 수렴하는 데 오래 걸릴 것을 쉽게 예상할 수 있다.

초기값을 100, 100, 101로 설정했을 때 결과는 다음과 같다.

|  |
| --- |
| n xn1 |f(xn1)|  0 1.000000000000000e+02 9.599394829814011e+03  1 5.102099683752227e+01 2.399125893693674e+03  2 2.654571319864124e+01 5.992131681567441e+02  3 1.433027400005807e+01 1.493732825541210e+02  4 8.255909151219363e+00 3.702547010360203e+01  5 5.267741315315264e+00 9.016531626056562e+00  6 3.846841243370685e+00 2.063570222941094e+00  7 3.245870620732976e+00 3.748099933688394e-01  8 3.074227405067638e+00 3.091089849304285e-02  9 3.057272923931396e+00 3.027183997690308e-04  10 3.057103566902662e+00 3.021615224696461e-08  n xn1 |f(xn1)|  0 1.000000000000000e+02 9.599394829814011e+03  1 1.010000000000000e+02 9.796384879483159e+03  2 5.126964612711888e+01 2.423560930607891e+03  3 3.492251190183934e+01 1.080338658265326e+03  4 2.177469458295133e+01 3.879577973524058e+02  5 1.440765350053018e+01 1.512821058307370e+02  6 9.698671849205059e+00 5.699755928837025e+01  7 6.851965037727258e+00 2.161702924937045e+01  8 5.112665362446378e+00 8.056964792994815e+00  9 4.079228073612803e+00 2.917281609227592e+00  10 3.492649584392688e+00 9.773421403790086e-01  11 3.197131149224741e+00 2.708690966164369e-01  12 3.083826314349333e+00 4.850834386527714e-02  13 3.059108688492555e+00 3.587618776291102e-03  14 3.057134601510355e+00 5.549318214970178e-05  15 3.057103586653844e+00 6.551349263084205e-08 |

반복횟수가 확실히 증가한 것을 알 수 있다.

(v)

위의 방정식에서 초기값을 Newton-Raphson 방법에선 0.75, Secant 방법에선 0.5, 1.0으로설정 후 근을 구해보면 다음과 같다.

Newton-Raphson 결과

|  |
| --- |
| n xn1 |f(xn1)|  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 |

Secant 결과

|  |
| --- |
| n xn1 |f(xn1)|  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 |

(ii)에서와 마찬가지로 실제 근인지 확인 가능하다.

위 방정식의 실제 근 a는 0.68197481로 알려져 있다. 위에서와 마찬가지로 Newton-Rapshon 방법에서 다른 조건을 제거하여 Nmax번 반복해 본다.

위에서와 마찬가지로 n=6 이후에는 근이 변하지 않았고, 수식의 값은 1.59101E-18에서 변하지 않았다.

Secant 방법에서도 첫 번째 방정식에서와 마찬가지로 n=11 이후로 계산이 불가했고, n=10, 11에서 같은 근을 가졌다. 수식의 값은 329216.2591이다.

실제 근 약 0.68197481에서 충분히 먼 초기값 10, 10, 11을 입력해 결과를 확인한다.

Newton-Rapshon 결과(초기값 10)

|  |
| --- |
| n xn1 |f(xn1)|  0 1.000000000000000e+01 1.100000000000000e+01  1 1.208207726218718e+01 1.257206639241069e+01  2 1.455910680871516e+01 1.359348844292380e+01  3 8.265907611052445e+00 7.782813458940094e+00  4 1.068639255536724e+01 1.001959758984888e+01  5 8.446151656906341e+00 7.474701810916154e+00  6 1.376297250302063e+02 1.404659197435548e+02  7 2.318786505599139e+02 2.336227763978300e+02  8 2.802267365188743e+02 2.799195631014996e+02  9 3.547638418941937e+02 3.544124437931074e+02  10 2.918335225082849e+02 2.938324929555222e+02  …  50 8.385053889833262e+01 8.575550165161980e+01 |

충분한 횟수 50회 반복 끝에도 근을 찾지 못했다.

Secant 결과(초기값 10, 11)

|  |
| --- |
| n xn1 |f(xn1)|  0 1.000000000000000e+01 1.100000000000000e+01  1 1.100000000000000e+01 1.199999999999999e+01  2 -1.000000000000139e+00 1.008725755419513e-12 |

구하려는 근 x = 0.68197481이 아닌 다른 근 x = -1에 수렴하는 근을 구하고 반복이 종료되었다.

위와 같이 초기값을 적당한 값이 아닌 값으로 잘못 잡을 경우 원하는 근을 구하지 못할 수 있다.

**실습 문제 1-4**

위 방정식의 근은 자연로그의 밑으로 알려진 e = 2.718281828459045235360287471352이다. 이 근을 double-precision 방법과 single-precision 방법을 이용해 각각 구해보면 다음과 같다.

Double-precision 방법

|  |
| --- |
| n xn1 |f(xn1)|  0 2.700000000000000e+00 6.748226989716555e-03  1 2.718220212872235e+00 2.266736454592522e-05  2 2.718281827760716e+00 2.569009449615578e-10 |

절대 오차는 약 6.9833E-10이다.

Single-Precision 방법

|  |
| --- |
| n xn1 |f(xn1)|  0 2.700000047683716e+00 6.748209241777658e-03  1 2.718220233917236e+00 2.265962211822625e-05  2 2.718281745910645e+00 3.036786111465517e-08 |

절대 오차는 약 8.25484E-08이다.

Single-Precision 방법으로 구한 e의 오차가 Double-Precision 방법으로 구한 e의 오차보다 약 118배 더 크다. Single precision 방법을 위해 float 자료형을 사용하였고 double precision 방법을위해 double 자료형을 사용하였다. Float 자료형은 4byte, double 자료형은 8byte로 소수 표현 범위가 double이 float보다 훨씬 넓으며 유효 자릿수도 float은 7자리, double은 16자리로 double이 훨씬 더 정밀한 값을 구할 수 있다. Delta 값을 충분히 작게 하면 실제 e에 가까운 값을 구할 수 있지만 float의 경우 double보다 부정확한 값이 나올 수 밖에 없다.

**숙제 1-1**

(iii)

Bisection 방법으로 f1(x) = 0, f2(x) = 0, f3(x) = 0의 근을 구해본다.

f1(x) = 0을 구하기 위해 초기값 a0 = 2, b0 = 4

|  |
| --- |
| n 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  20 3.057103157043457e+00 7.022431072201130e-07 |

실제 근 3.0571035에 가까워지는 것을 확인할 수 있다.

f2(x) = 0을 구하기 위해 a0 = 0.5, b0 = 1을 초기값으로 놓고 근을 구해본다.

|  |
| --- |
| n 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 |

실제 근 0.68197481에 가까워지는 것을 알 수 있다.

f3(x) = 0의 네 근(1.1, 2.2, 3.3, 4.4)를 구하기 위해 각각 초기값 a0, b0를 (1, 2), (2, 3), (3, 4), (4, 5)를 주었다.

(1, 2)를 준 결과

|  |
| --- |
| 0 1.500000000000000e+00 1.461599999999983e+00  1 1.250000000000000e+00 9.201937500000028e-01  2 1.125000000000000e+00 1.914339843749957e-01  3 1.062500000000000e+00 3.185422119140640e-01  4 1.093750000000000e+00 5.043403472900110e-02  5 1.109375000000000e+00 7.370435628891414e-02  6 1.101562500000000e+00 1.244565505385253e-02  7 1.097656250000000e+00 1.879038651622977e-02  8 1.099609375000000e+00 3.121562588233928e-03  9 1.100585937500000e+00 4.674728576716802e-03  10 1.100097656250000e+00 7.797558845936692e-04  11 1.099853515625000e+00 1.170109841105216e-03  12 1.099975585937500e+00 1.949786366068906e-04  13 1.100036621093750e+00 2.924382049087626e-04  14 1.100006103515625e+00 4.874217994910168e-05  15 1.099990844726562e+00 7.311512930385788e-05  16 1.099998474121094e+00 1.218569993710616e-05  17 1.100002288818359e+00 1.827843369284210e-05  18 1.100000381469727e+00 3.046415308460837e-06  19 1.099999427795410e+00 4.569630213779874e-06 |

실제 근 1.1과 유사한 값을 구할 수 있다.

(2, 3)를 준 결과

|  |
| --- |
| 0 2.500000000000000e+00 6.383999999999972e-01  1 2.250000000000000e+00 1.298062500000086e-01  2 2.125000000000000e+00 2.054964843749971e-01  3 2.187500000000000e+00 3.345974121094031e-02  4 2.218750000000000e+00 4.947273101806360e-02  5 2.203125000000000e+00 8.306866550448433e-03  6 2.195312500000000e+00 1.250448483823874e-02  7 2.199218750000000e+00 2.080424975979156e-03  8 2.201171875000000e+00 3.117866029249683e-03  9 2.200195312500000e+00 5.198757007818244e-04  10 2.199707031250000e+00 7.799866123079369e-04  11 2.199951171875000e+00 1.299833533536798e-04  12 2.200073242187500e+00 1.949642113245886e-04  13 2.200012207031250e+00 3.249493688173288e-05  14 2.199981689453125e+00 4.874308145730311e-05  15 2.199996948242188e+00 8.123790557590382e-06  16 2.200004577636719e+00 1.218564359106722e-05  17 2.200000762939453e+00 2.030944109776556e-06  18 2.199998855590820e+00 3.046418832752806e-06  19 2.199999809265137e+00 5.077362388306028e-07 |

(3, 4)를 준 결과

|  |
| --- |
| 0 3.500000000000000e+00 5.615999999999630e-01  1 3.250000000000000e+00 1.298062500000370e-01  2 3.375000000000000e+00 2.054964843749687e-01  3 3.312500000000000e+00 3.345974121088346e-02  4 3.281250000000000e+00 4.947273101806360e-02  5 3.296875000000000e+00 8.306866550476855e-03  6 3.304687500000000e+00 1.250448483821032e-02  7 3.300781250000000e+00 2.080424975979156e-03  8 3.298828125000000e+00 3.117866029278105e-03  9 3.299804687500000e+00 5.198757008102461e-04  10 3.300292968750000e+00 7.799866122510934e-04  11 3.300048828125000e+00 1.299833533252581e-04  12 3.299926757812500e+00 1.949642113530103e-04  13 3.299987792968750e+00 3.249493693857630e-05  14 3.300018310546875e+00 4.874308142888140e-05  15 3.300003051757812e+00 8.123790500746964e-06  16 3.299995422363281e+00 1.218564361948893e-05  17 3.299999237060547e+00 2.030944138198265e-06  18 3.300001144409180e+00 3.046418804331097e-06  19 3.300000190734863e+00 5.077361819871840e-07 |

(4, 5)를 준 결과

|  |
| --- |
| 0 4.500000000000000e+00 9.384000000000086e-01  1 4.250000000000000e+00 9.201937500000028e-01  2 4.375000000000000e+00 1.914339843750028e-01  3 4.437500000000000e+00 3.185422119140711e-01  4 4.406250000000000e+00 5.043403472900110e-02  5 4.390625000000000e+00 7.370435628887861e-02  6 4.398437500000000e+00 1.244565505384543e-02  7 4.402343750000000e+00 1.879038651623688e-02  8 4.400390625000000e+00 3.121562588269455e-03  9 4.399414062500000e+00 4.674728576723908e-03  10 4.399902343750000e+00 7.797558846434072e-04  11 4.400146484375000e+00 1.170109841105216e-03  12 4.400024414062500e+00 1.949786366779449e-04  13 4.399963378906250e+00 2.924382049442897e-04  14 4.399993896484375e+00 4.874217989225826e-05  15 4.400009155273438e+00 7.311512934649045e-05  16 4.400001525878906e+00 1.218569992289531e-05  17 4.399997711181641e+00 1.827843379231808e-05  18 4.399999618530273e+00 3.046415251617418e-06  19 4.400000572204590e+00 4.569630114303891e-06 |

Bisection 방법을 사용할 경우 n + 1번 째 반복에서 구한 값 x(n+1)과 이전에 구한 값 xn의 차이가 epsilon(10^-6)보다 작아서 반복문이 종료되었다. Epsilon 값을 더 작게 하면 더 정밀한 근을 구할 수 있다.

(iv)

함수 f1(x), f2(x)에 대해서는 Newton-Raphson 방법과 Secant 방법으로 근을 구해봤고, 해당 방법에 대하여 근에 수렴하는 속도를 확인하였다. f1(x), f2(x)에 대하여 Bisection 방법으로 근을 구하면 다음과 같다.

f1(x) = 0 의 결과는 다음과 같다. 초기값은 a0 = 2, b0 = 4를 주었다.

|  |
| --- |
| n 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  20 3.057103157043457e+00 7.022431072201130e-07 |

Bisection 방법을 사용할 경우 아래 수식은

이론적으로 n이 증가할수록 0.5에 수렴한다.

위의 결과로 이를 확인해본다.

|  |  |
| --- | --- |
| n |  |
| 0 | 7.756030716 |
| 1 | 0.435534036 |
| 2 | 0.351984095 |
| 3 | 0.079481269 |
| 4 | 4.790790327 |
| 5 | 0.395633086 |
| … | … |
| 17 | 0.514931163 |
| 18 | 0.471003574 |
| 19 | 0.561563078 |

수식의 값이 정확히 0.5가 되지는 않지만 0.5에 수렴할 것을 알 수 있다. 근 또한 수렴하는 것을 알 수 있었다.

마찬가지로 f2(x) = 0 의 결과는 다음과 같다. 초기값으로 a0 = 0.5, b0 = 1을 주었다.

|  |
| --- |
| n 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 |

|  |  |
| --- | --- |
| n |  |
| 0 | 0.837554588 |
| 1 | 0.096976014 |
| 2 | 4.655914095 |
| 3 | 0.392609702 |
| 4 | 0.226470597 |
| …. | …. |
| 14 | 0.445152282 |
| 15 | 0.376788865 |
| 16 | 0.172996879 |
| 17 | 1.390225543 |

f1(x) = 0과 달리 0.5에 수렴하는 것을 확인할 수 없다. 근이 수렴하는 것은 확인 가능했다.

f3(x) = 0의 네 근 1.1, 2.2, 3.3, 4.4를 Bisection 방법으로 구해보자.

근 1.1을 구하기 위해 초기값 a0 = 1, b0 = 2로 설정하여 근을 구하면 다음과 같다.

|  |
| --- |
| n xn1 |f(xn1)|  0 1.500000000000000e+00 1.461599999999983e+00  1 1.250000000000000e+00 9.201937500000028e-01  2 1.125000000000000e+00 1.914339843749957e-01  3 1.062500000000000e+00 3.185422119140640e-01  4 1.093750000000000e+00 5.043403472900110e-02  5 1.109375000000000e+00 7.370435628891414e-02  6 1.101562500000000e+00 1.244565505385253e-02  7 1.097656250000000e+00 1.879038651622977e-02  8 1.099609375000000e+00 3.121562588233928e-03  9 1.100585937500000e+00 4.674728576716802e-03  10 1.100097656250000e+00 7.797558845936692e-04  11 1.099853515625000e+00 1.170109841105216e-03  12 1.099975585937500e+00 1.949786366068906e-04  13 1.100036621093750e+00 2.924382049087626e-04  14 1.100006103515625e+00 4.874217994910168e-05  15 1.099990844726562e+00 7.311512930385788e-05  16 1.099998474121094e+00 1.218569993710616e-05  17 1.100002288818359e+00 1.827843369284210e-05  18 1.100000381469727e+00 3.046415308460837e-06  19 1.099999427795410e+00 4.569630213779874e-06 |

|  |  |
| --- | --- |
| n |  |
| 0 | 0.375 |
| 1 | 0.166666667 |
| 2 | 1.5 |
| 3 | 0.166666667 |
| 4 | 1.5 |
| …. | …. |
| 14 | 0.166666667 |
| 15 | 1.49999999 |
| 16 | 0.166666664 |
| 17 | 1.500000027 |

이론과 다르게 0.5에 수렴하는 것을 확인할 수 없었다. 근이 1.1에 수렴하는 것은 알 수 있었다.

근 2.2를 구하기 위해 초기값 a0, b0를 각각 2, 3으로 설정 후 구한 근은 다음과 같다.

|  |
| --- |
| n xn1 |f(xn1)|  0 2.500000000000000e+00 6.383999999999972e-01  1 2.250000000000000e+00 1.298062500000086e-01  2 2.125000000000000e+00 2.054964843749971e-01  3 2.187500000000000e+00 3.345974121094031e-02  4 2.218750000000000e+00 4.947273101806360e-02  5 2.203125000000000e+00 8.306866550448433e-03  6 2.195312500000000e+00 1.250448483823874e-02  7 2.199218750000000e+00 2.080424975979156e-03  8 2.201171875000000e+00 3.117866029249683e-03  9 2.200195312500000e+00 5.198757007818244e-04  10 2.199707031250000e+00 7.799866123079369e-04  11 2.199951171875000e+00 1.299833533536798e-04  12 2.200073242187500e+00 1.949642113245886e-04  13 2.200012207031250e+00 3.249493688173288e-05  14 2.199981689453125e+00 4.874308145730311e-05  15 2.199996948242188e+00 8.123790557590382e-06  16 2.200004577636719e+00 1.218564359106722e-05  17 2.200000762939453e+00 2.030944109776556e-06  18 2.199998855590820e+00 3.046418832752806e-06  19 2.199999809265137e+00 5.077362388306028e-07 |

|  |  |
| --- | --- |
| n |  |
| 0 | 0.166666667 |
| 1 | 1.5 |
| 2 | 0.166666667 |
| 3 | 1.5 |
| 4 | 0.166666667 |
| …. | …. |
| 15 | 1.499999993 |
| 16 | 0.166666666 |
| 17 | 1.500000007 |
| 18 | 0.166666673 |

근이 2.2에 수렴하는 것은 확인할 수 있었지만 수식이 0.5에 수렴하는 것은 확인할 수 없었다.

근 3.3을 구하기 위해 초기값 a0, b0를 3, 4로 설정 후 근을 구했다.

|  |
| --- |
| 0 3.500000000000000e+00 5.615999999999630e-01  1 3.250000000000000e+00 1.298062500000370e-01  2 3.375000000000000e+00 2.054964843749687e-01  3 3.312500000000000e+00 3.345974121088346e-02  4 3.281250000000000e+00 4.947273101806360e-02  5 3.296875000000000e+00 8.306866550476855e-03  6 3.304687500000000e+00 1.250448483821032e-02  7 3.300781250000000e+00 2.080424975979156e-03  8 3.298828125000000e+00 3.117866029278105e-03  9 3.299804687500000e+00 5.198757008102461e-04  10 3.300292968750000e+00 7.799866122510934e-04  11 3.300048828125000e+00 1.299833533252581e-04  12 3.299926757812500e+00 1.949642113530103e-04  13 3.299987792968750e+00 3.249493693857630e-05  14 3.300018310546875e+00 4.874308142888140e-05  15 3.300003051757812e+00 8.123790500746964e-06  16 3.299995422363281e+00 1.218564361948893e-05  17 3.299999237060547e+00 2.030944138198265e-06  18 3.300001144409180e+00 3.046418804331097e-06  19 3.300000190734863e+00 5.077361819871840e-07 |

|  |  |
| --- | --- |
| n |  |
| 0 | 0.25 |
| 1 | 1.5 |
| 2 | 0.166666667 |
| 3 | 1.5 |
| 4 | 0.166666667 |
| …. | …. |
| 15 | 1.500000002 |
| 16 | 0.166666668 |
| 17 | 1.499999988 |
| 18 | 0.166666664 |

마찬가지로 근이 3.3에 수렴하는 것은 확인 가능하지만 수식이 0.5에 수렴하는 것을 확인하지는 못했다.

마지막으로 근 4.4를 구하기 위해 초기값 a0, b0를 4, 5로 설정 후 구했다.

|  |
| --- |
| n xn1 |f(xn1)|  0 4.500000000000000e+00 9.384000000000086e-01  1 4.250000000000000e+00 9.201937500000028e-01  2 4.375000000000000e+00 1.914339843750028e-01  3 4.437500000000000e+00 3.185422119140711e-01  4 4.406250000000000e+00 5.043403472900110e-02  5 4.390625000000000e+00 7.370435628887861e-02  6 4.398437500000000e+00 1.244565505384543e-02  7 4.402343750000000e+00 1.879038651623688e-02  8 4.400390625000000e+00 3.121562588269455e-03  9 4.399414062500000e+00 4.674728576723908e-03  10 4.399902343750000e+00 7.797558846434072e-04  11 4.400146484375000e+00 1.170109841105216e-03  12 4.400024414062500e+00 1.949786366779449e-04  13 4.399963378906250e+00 2.924382049442897e-04  14 4.399993896484375e+00 4.874217989225826e-05  15 4.400009155273438e+00 7.311512934649045e-05  16 4.400001525878906e+00 1.218569992289531e-05  17 4.399997711181641e+00 1.827843379231808e-05  18 4.399999618530273e+00 3.046415251617418e-06  19 4.400000572204590e+00 4.569630114303891e-06 |

|  |  |
| --- | --- |
| n |  |
| 0 | 1.5 |
| 1 | 0.166666667 |
| 2 | 1.5 |
| 3 | 0.166666667 |
| 4 | 1.5 |
| …. | …. |
| 15 | 0.166666666 |
| 16 | 1.500000007 |
| 17 | 0.166666668 |
| 18 | 1.499999984 |

근이 4.4로 수렴하는 것은 확인 가능하지만 수식이 0.5로 수렴하는 것은 확인할 수 없다.

Newton-Raphson 방법과 Secant 방법에 대해선 앞에서 설명한 이유로 수렴함을 알 수 있다.

**숙제 1-2**

숙제 1-2를 해결하기 위해 다음과 같이 상수를 계산하였다.

|  |
| --- |
| double l = 89;  double h = 49;  double D = 55;  double B1 = 11.5;  double A = l \* sin(B1 \* M\_PI / 180);  double B = l \* cos(B1 \* M\_PI / 180);  double C = (h + 0.5 \* D) \* sin(B1 \* M\_PI / 180) - 0.5 \* D \* tan(B1 \* M\_PI / 180);  double E = (h + 0.5 \* D) \* cos(B1 \* M\_PI / 180) - 0.5 \* D; |

또한 Newton-Raphson 방법으로 해결하기 위해 함수 f(x)와 f’(x)를 정의했다.

|  |
| --- |
| double \_f\_vehicle(double x) {  return A \* sin(x) \* cos(x) + B \* sin(x) \* sin(x) - C \* cos(x) - E \* sin(x);  }  double \_fp\_vehicle(double x) {  return A \* cos(2 \* x) + 2 \* B \* sin(x) \* cos(x) + C \* sin(x) - E \* cos(x);  } |

Newton-Raphson 방법을 구현한 코드는 다음과 같다.

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Newton-Rapson Method  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void program1\_1(FILE\* fp) {  double x0, x1;  int n;  if (fp == NULL)  return;  // 초기값 입력  printf("Input x0 : ");  scanf("%lf", &x0);  fprintf(fp, " n xn1 |f(xn1)|\n");  fprintf(fp, "%2d %.15e %.15e\n", 0, x0, fabs(\_f(x0)));  for (n = 1; n <= Nmax; n++) {  x1 = x0;  x1 -= \_f(x0) / \_fp(x0);  fprintf(fp, "%2d %.15e %.15e\n", n, x1, fabs(\_f(x1)));  if (fabs(\_f(x1)) < DELTA || fabs(x1 - x0) < EPSILON)  break;  x0 = x1;  }  fprintf(fp, "\n");  } |

사용자가 적당한 초기값을 입력하면 근이 계산된다. 최대 Nmax번 만큼 반복하여 근을 구하고 만약 구한 근을 f(x)에 대입한 값이 충분히 작은 양수 DELTA보다 작거나, 이전 반복문에서 구한 근과 현재 구한 근의 차가 충분히 작은 양수 EPSILON보다 작다면 종료한다.

이론적인 a(여기서는 x)의 값이 33도 이므로, 이를 호도법으로 바꾸면, x=0.575958653158129이다. 근을 구하기 위해 초기값을 0.6으로 설정 후, 근을 구한 결과는 다음과 같다.

|  |
| --- |
| n xn1 |f(xn1)|  0 6.000000000000000e-01 1.304022822541032e+00  1 5.758489459098021e-01 1.966934604737247e-02  2 5.754731101513064e-01 5.123897938830169e-06  3 5.754730121944055e-01 3.517186542012496e-13 |

구한 근은 0.575473012194455이다. 이를 60분법 각으로 바꾸면 약 32.972174822422827도가 된다. EPSILON과 DELTA 값을 기존에 주어진 10^-6보다 훨씬 작은 값(10^-18)으로 설정 후 결과는 다음과 같다.

|  |
| --- |
| n xn1 |f(xn1)|  0 6.000000000000000e-01 1.304022822541032e+00  1 5.758489459098021e-01 1.966934604737247e-02  2 5.754731101513064e-01 5.123897938830169e-06  3 5.754730121944055e-01 3.517186542012496e-13  4 5.754730121943987e-01 0.000000000000000e+00 |

이 때 구한 근은 32.972174822419601651751351238348도로 오히려 이전에 구한 값보다 원래 근에서 더 먼 값이 나왔다.