**고급소프트웨어실습1 6주차 보고서**

컴퓨터공학 20172141 김미소

**[프로그램 소개 및 구동 방법]**

FORTRAN으로 쓰여진 수치 계산용 라이브러리를 통해 비선형 방정식의 근을 구하는 프로그램을 구현한다. Minpack 함수와 rpoly 함수 등을 사용하여 비선형 방정식의 근을 찾고 출력하는 프로그램이다.

[구동방법]

- 실습 3-1: A와 b를 임의로 지정하고 ctrl+F5를 눌러 프로그램을 실행시킬 수 있다.

- 실습 3-2: ctrl+F5를 눌러 프로그램을 실행하면 polynomial\_3-2\_i.txt를 읽어 해를 구할 수 있다. 프로젝트 폴더 안에 파일이 존재하는지 확인하고 주의해야한다. 근은 roots\_3-2\_i.txt에 저장된다.

- 실습 3-3 ~ 3-8: ctrl+F5를 눌러 실행하면 근을 확인할 수 있다. 근은 roots\_3-3.txt ~ roots\_3-8.txt에 저장된다.

- 숙제 3-1: ctrl+F5를 눌러 프로그램을 실행하면 GPS\_signal\_i.txt를 읽어 다항식을 풀 수 있다. 프로젝트 폴더 안에 파일이 존재하는지 확인하고 주의해야한다. HYBRJ1으로 구한 근은 GPS\_position\_3-1\_i.txt에 저장되고 HYBRD1으로 구한 근은 GPS\_position\_3-2\_i.txt에 저장된다.

- 숙제 3-2, 3-3: ctrl+F5를 눌러 실행하면 근을 확인할 수 있다. 근은 roots\_found\_3-2.txt, roots\_found\_3-3.txt에 저장된다.

- 숙제 3-4: ctrl+F5를 눌러 프로그램을 실행하면 linear\_system\_3-4.txt를 읽어 다항식의 근을 찾을 수 있다. 프로젝트 폴더 안에 파일이 존재하는지 확인하고 주의해야한다. 찾아낸 근은 solution\_3-4.txt에 저장된다. 첫 줄에는 n에 대한 정보, 그 이후 n줄에는 근이 출력되며 마지막 줄에는 오차를 알 수 있다.

**[실습 3-1]**

임의의 Ax = b 형태의 방정식을

위와 같이 임의로 설정하여 방정식을 푼 결과 아래와 같은 결과를 가졌다.

텍스트이(가) 표시된 사진

자동 생성된 설명

FORTRAN 함수 배열의 order는 C/C++와 다르므로 행렬(배열)을 전치함으로써 FORTRAN 함수에 인자로서 전달하였고 그 결과 근이 모두 1로 잘 도출되는 것을 확인하였다.

**[실습 3-2]**

텍스트이(가) 표시된 사진

자동 생성된 설명

위 식에 대한 정보가 담긴 polynomial\_3-2\_i.txt 파일들을 읽어 rpoly 함수를 통해 다항식의 근을 실수 부분인 zeror 배열, 허수 부분인 zeroi 배열에 각각 저장하였다. 그 결과

텍스트, 명판, 점수판이(가) 표시된 사진

자동 생성된 설명

위와 같은 근들을 찾아 낼 수 있었다. 찾아낸 근들을 다시 다항식에 대입하여 얼마나 정확히 근을 찾게 되었는지 확인해보았는데 모두 0으로 나타나 근이 제대로 구해진 것을 확인할 수 있었다. 근을 찾지 못한 다항식에 대해서는 실패 메시지를 결과로서 출력하였다. 각 다항식에 대한 값은 roots\_3-2\_i.txt에 출력하여 저장하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

**[실습 3-3]**

텍스트이(가) 표시된 사진

자동 생성된 설명

위의 식을 fvec 배열에 저장하고 이 식에 대한 미분 정보를 fjac 배열에 저장하여 HYBRJ1 함수를 통해 각 초기값에 대하여 근을 구하였다. 그 결과로

텍스트이(가) 표시된 사진

자동 생성된 설명

위와 같은 결과를 갖게 되었고 이 결과를 roots\_3-3.txt에 출력하여 저장하였다. Info 값은 4가 출력되었고 info 4는 “Iteration is not making good progress.”라는 메시지로 근이 제대로 구해지지 않았음을 알 수 있다. 그리고 식에 직접 근을 대입해 본 결과 오차가 비교적 크게 생기는 것을 알 수 있었다.

테이블이(가) 표시된 사진

자동 생성된 설명

**[실습 3-4]**

텍스트이(가) 표시된 사진

자동 생성된 설명

실습 3-3과 같은 방법으로 위의 식을 fvec 배열에 저장하고 이 식에 대한 미분 정보를 fjac 배열에 저장하여 HYBRJ1 함수를 통해 각 초기값에 대하여 근을 구하였다. 그 결과로

텍스트이(가) 표시된 사진

자동 생성된 설명

위와 같은 결과를 갖게 되었고 이 결과를 roots\_3-4.txt에 출력하여 저장하였다. Info 값은 1이 출력되었고 info 1은 근이 성공적으로 구했음을 의미한다. 그리고 식에 직접 근을 대입해 본 결과 0이 나왔으며 이는 제대로 근을 구했음을 알 수 있다.

테이블이(가) 표시된 사진

자동 생성된 설명

**[실습 3-5]**

텍스트이(가) 표시된 사진

자동 생성된 설명

위의 식을 fvec 배열에 저장하여 HYBRD1 함수를 통해 각 초기값에 대하여 근을 구하였다. HYBRD1 함수는 미분 값을 이용하지 않으므로 fjac 배열이 필요 없다. HYBRD1 함수를 통해 근을 구한 결과

텍스트이(가) 표시된 사진

자동 생성된 설명

위와 같은 결과를 갖게 되었고 이 결과를 roots\_3-5.txt에 출력하여 저장하였다. Info 값은 1이 출력되었고 info 1은 근이 성공적으로 구했음을 의미한다. 그리고 식에 직접 근을 대입해 본 결과 0이 나왔으며 이는 제대로 근을 구했음을 알 수 있다.

**[실습 3-6]**

텍스트, 시계이(가) 표시된 사진

자동 생성된 설명

위의 식을 fvec 배열에 저장하고 이 식에 대한 미분 정보를 fjac 배열에 저장하여 HYBRJ1 함수를 통해 [-4,4] x [-5,5] 구간의 x, y에 대하여 근을 구하였다. 그 결과로 총 4개의 근이 도출되었으며 근은 다음과 같다.

[1] (x, y) = (-1.192873, 1.278444)

[2] (x, y) = (-1.912160, -2.232608)

[3] (x, y) = (1.781541, 2.064179)

[4] (x, y) = (2.923492, -3.510016)

4개의 근 모두 info가 1이 나왔으며 각 근을 식에 대입해본 결과 모두 0이 나와 근을 제대로 구했음을 알 수 있었다. 각 x, y에 대하여 근을 구한 결과는 roots\_3-6.txt에 출력하여 저장하였다.

**[실습 3-7]**

텍스트, 시계이(가) 표시된 사진

자동 생성된 설명

위의 식을 fvec 배열에 저장하고 이 식에 대한 미분 정보를 fjac 배열에 저장하여 HYBRJ1 함수를 통해 [-4,4] x [-5,5] 구간의 x, y에 대하여 근을 구하였다. 그 결과로 총 7개의 근이 도출되었으며 근은 다음과 같다.

[1] (x, y) = (-2.307891, 2.109371)

[2] (x, y) = (-2.410820, -0.093798)

[3] (x, y) = (-0.285560, 2.380150)

[4] (x, y) = (-0.049321, -0.408389)

[5] (x, y) = (2.590859, 2.692223)

[6] (x, y) = (2.462734, -0.679557)

[7] (x, y) = (2.590859, 2.692223)

7개의 근 모두 info가 1이 나왔으며 각 근을 식에 대입해본 결과 모두 0이 나와 근을 제대로 구했음을 알 수 있었다. 각 x, y에 대하여 근을 구한 결과는 roots\_3-7.txt에 출력하여 저장하였다.

**[실습 3-8]**

텍스트이(가) 표시된 사진

자동 생성된 설명

위와 같은 비선형 시스템의 근을 구하기 위해 HYBRJ1 함수를 사용하였다. 위 식을 fvec 배열에 저장하고 위 식의 미분에 대한 정보를 fjac 배열에 저장하여 HYBRJ1 함수를 통해 근을 구하였다.



초기값은 위와 같다.

근을 구한 결과,

텍스트이(가) 표시된 사진

자동 생성된 설명

이러한 결과를 얻게 되었고 문제에서 제시한 근에 수렴함을 알 수 있었다. 근을 식에 직접 대입한 결과도 0으로 도출되어 근을 제대로 구했음을 알 수 있었다. 위 결과는 roots\_3-8.txt에 출력하여 저장하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**[숙제 3-1]**

텍스트이(가) 표시된 사진

자동 생성된 설명

위 다항식을 HYBRJ1, HYBRD1 함수를 이용하여 각각 근을 구해야 한다. 각 계수 p와 t, tr에 대한정보는 GPS\_signal\_i.txt에 저장되어 있다. 이를 읽어 들이고 위 식을 fvec에 저장하면

텍스트이(가) 표시된 사진

자동 생성된 설명

이와 같은 코드로 저장할 수 있다. 그리고 이 다항식에 대한 야코비 행렬 fjac은

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트, 전자기기이(가) 표시된 사진

자동 생성된 설명

위와 같다.

이 다항식을 HYBRJ1 함수를 통해 근을 구하면 각 GPS\_signal\_i.txt 파일에 대하여

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명이다.

그리고 HYBRD1 함수를 통해 근을 구하면 각 GPS\_signal\_i.txt 파일에 대하여

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명이다.

두 방법을 통해 구한 근을 직접 식에 넣었을 때 0과 가까운 값을 가졌으므로 제대로 근을 구했음을 알 수 있었다.

위 결과들을 HYBRJ1은 GPS\_position\_3-1\_i.txt에 출력하여 저장하고 HYBRD1은 GPS\_position\_3-2\_i.txt에 출력하여 저장하였다.

위 두 방법을 사용하여 문제를 풀면서 알게 된 것은 HYBRJ1과 HYBRD1은 서로 다른 wa, lwa 크기를 사용한다는 것이다. 이를 알지 못하고 함수를 사용하였을 때 HYBRD1의 결과가 제대로 나오지 않아 애를 먹었다.

**[숙제 3-2]**

테이블이(가) 표시된 사진

자동 생성된 설명

위 식을 minpack 함수 중 적절한 함수를 사용하여 풀면 되는 문제이다. HYBRJ1 함수를 사용하여 문제를 풀었다. 문제를 푼 결과

텍스트이(가) 표시된 사진

자동 생성된 설명

위와 같이 w, x, y, z 값을 얻을 수 있었고 이를 직접 식에 넣어본 결과 0이 나왔으므로 근을 제대로 구했음을 알 수 있다. 이 결과를 roots\_found\_3-2.txt에 출력하여 저장하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**[숙제 3-3]**

테이블이(가) 표시된 사진

자동 생성된 설명

위 식을 minpack 함수 중 적절한 함수를 사용하여 풀면 되는 문제이다. HYBRD1 함수를 사용하여 문제를 풀었다. 문제를 푼 결과

텍스트이(가) 표시된 사진

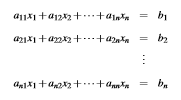
자동 생성된 설명

Info는 4가 나왔고 이는 해를 제대로 구하지 못했음을 의미한다. 제대로 구하지 못한 근을 식에 직접 넣어본 결과 오차가 크게 나타난 것을 확인할 수 있다. 이 결과를 roots\_found\_3-3.txt에 출력하여 저장하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**[숙제 3-4]**



위와 같은 형식의 다항식에서 a와 b에 대한 정보가 담겨있는 txt파일을 읽어 다항식의 근을 찾아야하는 문제이다. General과 Hilbert 파일에 이 정보들이 담겨있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이러한 정보가 General, Hilbert 파일에 담겨있다. 이 파일들을 읽어서 배열에 저장하고 이를 gespp, solve 함수를 이용하여 근을 구해야 한다. 이 파일들을 linear\_system\_3-4.txt로 고쳐 실험을 진행하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

N의 개수는 파일마다 다르기 때문에 배열을 동적 할당하여 사용하는 방식을 택했다. 파일을 읽으면서 a와 b에 대한 정보들을 배열에 저장하였다. mat배열은 나중에 오차에 대한 척도를 편하게 계산하기 위하여 따로 할당한 배열이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Gespp, solve 함수를 이용하여 다항식의 근을 찾고 이 근을 콘솔과 solution\_3-4.txt에 출력하여 저장하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그리고 오차에 대한 척도를 계산하였는데, 오차에 대한 척도를 구하는 방법은



이와 같다.

따라서 Ax-b에 대한 크기와 b에 대한 크기를 나누어 오차에 대한 척도를 계산할 수 있었다.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| General\_2 | General\_3 | General\_4 | General\_5 | General\_7 | General\_9 | General\_12 | General\_16 | General\_24 | General\_32 |
| 2  1.200002  1.200001  0.116134 | 3  1.199997  1.200000  1.200001  0.407469 | 4  1.200000  1.200000  1.200000  1.199998  0.479213 | 5  1.200005  1.200009  1.200017  1.200011  1.200002  0.968443 | 7  1.199998  1.199998  1.200000  1.200002  1.200000  1.199998  1.199999  1.045460 | 9  1.199998  1.199999  1.200000  1.199999  1.200000  1.200000  1.200002  1.200000  1.200000  1.230565 | 12  1.200001  1.200001  1.199999  1.199999  1.200001  1.199998  1.199999  1.199999  1.199999  1.200001  1.200001  1.199998  0.947641 | 16  1.199997  1.200026  1.200008  1.199975  1.200016  1.200004  1.200013  1.199994  1.199995  1.200004  1.200018  1.200005  1.200001  1.199981  1.200006  1.199998  0.985508 | 24  1.199999  1.200000  1.200004  1.200001  1.199998  1.200005  1.199993  1.199998  1.200004  1.200004  1.199999  1.200006  1.200003  1.199996  1.200000  1.199994  1.200001  1.200000  1.200003  1.199995  1.200001  1.199999  1.199995  1.199996  0.818300 | 32  1.200009  1.200001  1.200001  1.200003  1.200002  1.200000  1.200002  1.199998  1.200006  1.199999  1.199996  1.199997  1.199998  1.199998  1.199999  1.200000  1.199999  1.200007  1.199999  1.200008  1.200004  1.200011  1.200010  1.200000  1.199997  1.199996  1.200007  1.200004  1.199996  1.199998  1.200002  1.199997  0.897570 |
| Hilbert\_2 | Hilbert \_3 | Hilbert \_4 | Hilbert \_5 | Hilbert \_7 | Hilbert \_9 | Hilbert \_12 | Hilbert \_16 | Hilbert \_24 | Hilbert \_32 |
| 2  1.199997  1.200005  0.499230 | 3  1.200003  1.199981  1.200021  0.678095 | 4  1.199931  1.200762  1.198188  1.201169  0.809543 | 5  1.200397  1.193289  1.227062  1.161077  1.218346  0.915974 | 7  1.199610  1.212148  1.114923  1.412109  1.013125  1.201968  1.246869  1.086490 | 9  1.195502  1.312529  0.501077  2.918451  -0.745466  2.674514  -0.626610  3.060824  0.509489  1.197915 | 12  1.196568  1.291344  0.646764  2.321204  0.552233  1.507283  -0.094211  1.925192  1.589094  1.075964  1.841395  0.543737  1.367641 | 16  1.201568  1.173602  1.260496  1.244426  1.403662  0.444684  0.752322  1.805505  2.525773  1.637263  0.707327  -0.007822  0.139721  1.551384  2.224458  1.141567  1.541855 | 24  1.201115  1.174724  1.305372  1.129437  1.187748  0.741319  1.714322  1.052305  2.136646  0.683046  1.038849  1.082641  0.575605  0.948405  2.153757  0.693043  1.010437  2.332265  0.732618  1.441887  1.274890  0.910785  0.904779  1.373910  1.788032 | 32  1.200483  1.158611  1.594239  0.166005  1.710798  1.801922  1.220548  1.386428  0.958678  -0.213381  1.563813  2.239414  0.293457  0.199223  1.542796  2.730713  2.645681  2.106283  0.203795  -0.351376  -0.778013  0.290373  1.644817  2.956786  3.693313  1.225411  0.679805  0.399413  -0.031147  0.623919  1.888847  1.645662  1.936407 |

빨간색으로 표시된 수치가 오차에 대한 척도이다. 이 데이터들은 각각 solution\_3-4.txt에 저장되었다. 오차들을 살펴보면 General은 오차가 커졌다가 다시 작아지는 모습을 확인할 수 있었고 Hilbert는 n이 커질수록 오차도 점점 커지는 모습을 볼 수 있었다. 이는 General이 Hilbert에 비하여 알고리즘이 안정적으로 수행되었음을 의미한다. General은 n이 커짐에도 불구하고 근들이 모두 1.2에 가까운 값을 갖는 숫자로 나타난 반면 Hilbert는 n이 커질수록 도출되는 근들도 1.2와는 먼 값들을 근으로서 도출하는 것을 확인할 수 있었다.