**고급소프트웨어실습1: 6주차 과제**

**20171672 이정원**

**프로그램 소개**

minpack 함수를 이용하여 비선형 방정식 시스템을 풀이하고 구한 근을 결과에 출력한다.

**프로그램 구동 방법**

main.cpp에서 실습 부분(practice3\_1~8)과 과제 부분(program3\_1~4)을 실행시킬 수 있다. 각 프로그램마다 초기값 입력을 요구할 수 있으며, 프로젝트 폴더 안에 입력 파일과 출력 파일을 확인할 수 있다.

**실습 문제 3-1**

수업 시간에 다룬 FORTRAN 함수를 저장하고 있는 gespp.f와 solve.f 파일의 함수들을 사용하여 선형 방정식의 근을 구하는 프로그램이다. 입력 4\*4 행렬(a)은 다음과 같다.

2, 0, 0, 0

4, 1, 1, 1

0, 5, 1, 2

1, 1, 1, 3

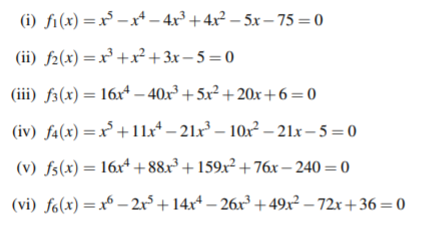
b는 (2, 7, 8, 6)t이다.

형식에 맞게 a와 b에 입력한 후 gespp\_()와 slove\_()를 호출하여 구한 근 (1, 1, 1, 1)t를 x에 저장한다. 출력 결과는 다음과 같다.

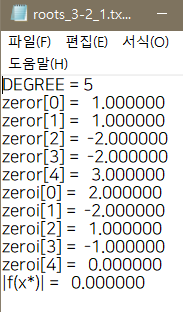
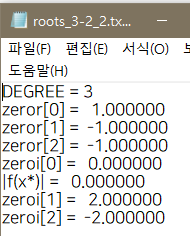
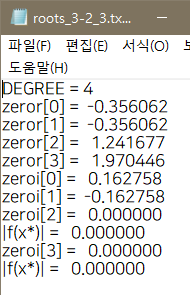
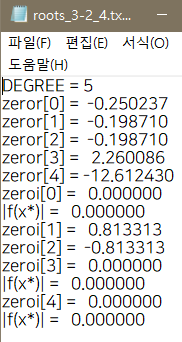
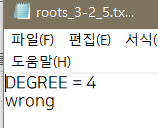
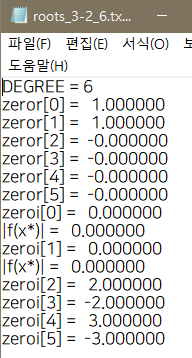


**실습 문제 3-2**

RPOLY 함수를 이용하여 다음 다항식 방정식의 근을 구한다.



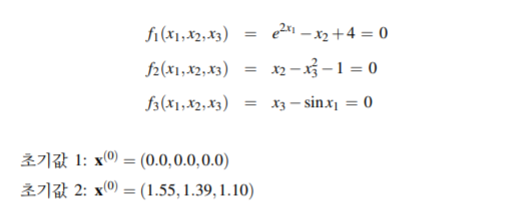
입력 텍스트 파일(polynomial\_3\_1\_i.txt)은 첫 번째 줄에는 DEGREE, 두 번째줄부터는 계수가 저장되어 있다. 이 파일을 읽어 들일 때 순서대로 DEGREE, poly 배열에 저장한다. 이후 rpoly\_()함수를 호출해 roots\_3-2\_i.txt에 결과를 저장한다. 다음은 roots\_3-2\_i.txt에 저장된 결과이다.

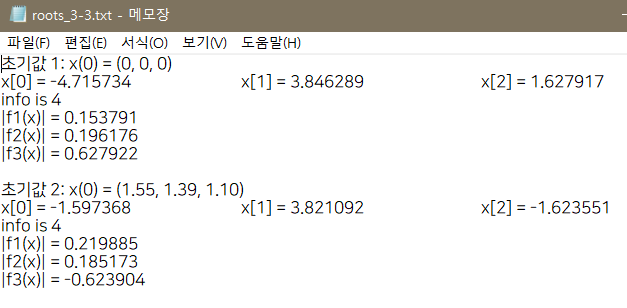
각 근에 대한 추정치 x\*에 대하여 |f(x\*)| 값도 포함시켰다.

**실습 문제 3-3**

HYBRJ1 함수를 이용하여 근을 구하는 프로그램이다. 입력 함수와 초기값은 다음과 같다.



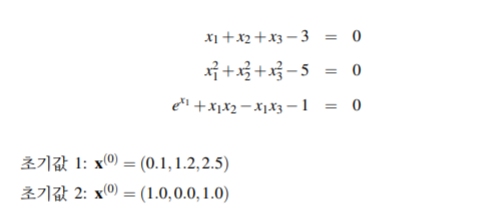
위 함수를 fcn3\_3의 fvec[0], fvec[1], fvec[2]에 저장하고, 각 함수에 대한 미분 함수를 fjac 배열에 저장한다. hybrj1\_( ) 함수를 호출한 후 해당 초기값에 대한 함수의 근은 x 배열에 저장된다. 출력 결과는 다음과 같다.



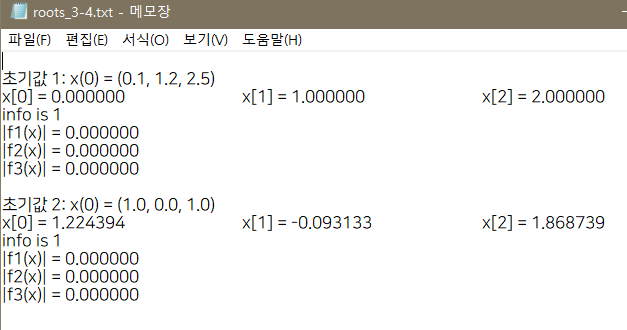
info가 4가 출력되었으므로 두 근 모두 hybrj1\_() 함수에 의해 제대로 구해지지 않은 걸 확인할 수 있었다.

**실습 문제 3-4**

HYBRJ1 함수를 이용하여 근을 구하는 프로그램이다. 입력 함수와 초기값은 다음과 같다.



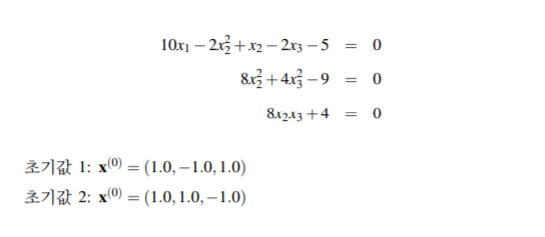
실습 3-3과 같이 진행하였다.



info가 1이 출력되고, 각 근을 넣었을 때 함수값이 0이 되어 제대로 된 근을 구했다는 것을 확인할 수 있었다.

**실습 문제 3-5**

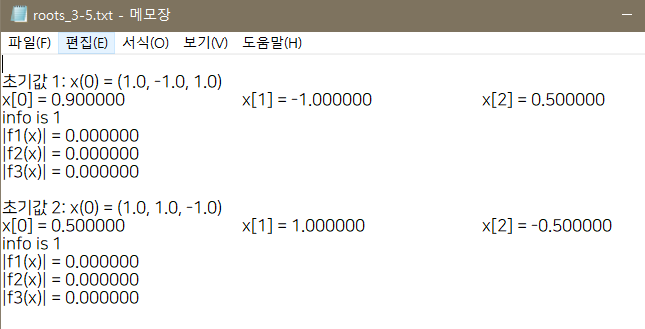
HYBRD1 함수를 사용하여 근을 구하는 프로그램이다. 입력 함수와 초기값은 다음과 같다.



HYBRD1 함수는 HYBRJ1 함수와 달리 미분 함수 값을 넣지 않아도 된다. 대신에 wa와 lwa 값을 다음과 같이 설정해 주어야 한다.

double wa[(SOLNUMS \* (3 \* SOLNUMS + 13)) / 2];

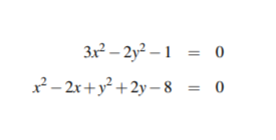
int lwa = (SOLNUMS \* (3 \* SOLNUMS + 13)) / 2;



info가 1이 나오고, 각 근을 넣었을 때 함수값이 0이 되어 제대로 된 근을 구했다는 것을 확인할 수 있었다.

**실습 문제 3-6**

HYBRJ1 함수를 사용하여 (x, y) ∈ [−4,4]×[−5,5]인 모든 근을 구하는 프로그램이다. 입력 함수는 다음과 같다.



모든 구간에 대하여 출력 결과 각 근을 넣었을 때 함수값이 0이 되는 근을 모두 찾아 중복 출력하였다.

근은 총 4개의 조합이 나왔다. 다음과 같다.

x[0], x[1] = -1.192873, 1.278444

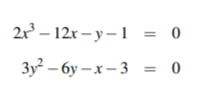
x[0], x[1] = -1.912160, -2.232608

x[0], x[1] = 1.781541, 2.064179

x[0], x[1] = 2.923492, -3.510016

**실습 문제 3-7**

HYBRD1 함수를 사용하여 (x, y) ∈ [−4,4]×[−5,5]인 모든 근을 구하는 프로그램이다. 입력 함수는 다음과 같다.



모든 구간에 대하여 출력 결과 각 근을 넣었을 때 함수값이 0이 되는 근을 모두 찾아 중복 출력하였다.

근은 총 6개의 조합이 나왔다. 다음과 같다.

x[0], x[1] = -2.307891, 2.109371

x[0], x[1] = -2.410820, -0.093798

x[0], x[1] = -0.285560, 2.380150

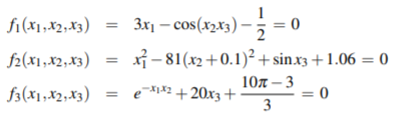
x[0], x[1] = -0.049321, -0.408389

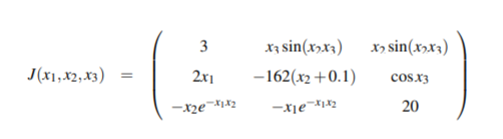
x[0], x[1] = 2.590859, 2.692223

x[0], x[1] = 2.462734, -0.679557

**실습 문제 3-8**

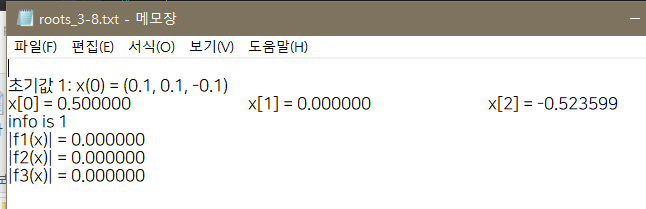
HYBRJ1 함수를 사용하여 비선형 방정식의 근을 구하는 프로그램이다. 입력 함수와 미분 함수, 그리고 초기값은 다음과 같다.







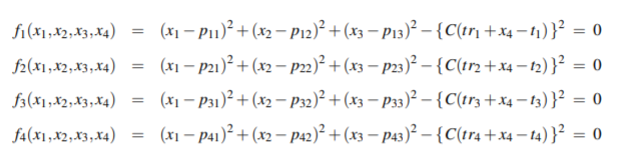
출력 결과는 다음과 같다.



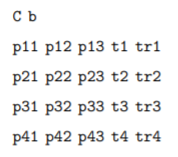
info가 1이 출력되고, 각 근을 넣었을 때 함수값이 0이 되어 제대로 된 근을 구했다는 것을 확인할 수 있었다.

**숙제 3-1 (program3\_1.cpp)**

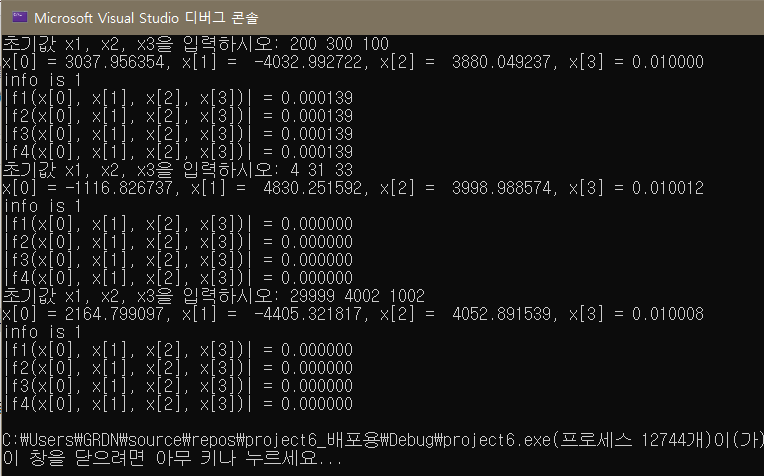
HYBRJ1 함수를 사용하여 GPS 수신기 위치를 찾아 주는 프로그램이다. 입력 함수는 다음과 같다.



입력 텍스트 파일(GPS\_signal\_i.txt)의 형식은 다음과 같다.

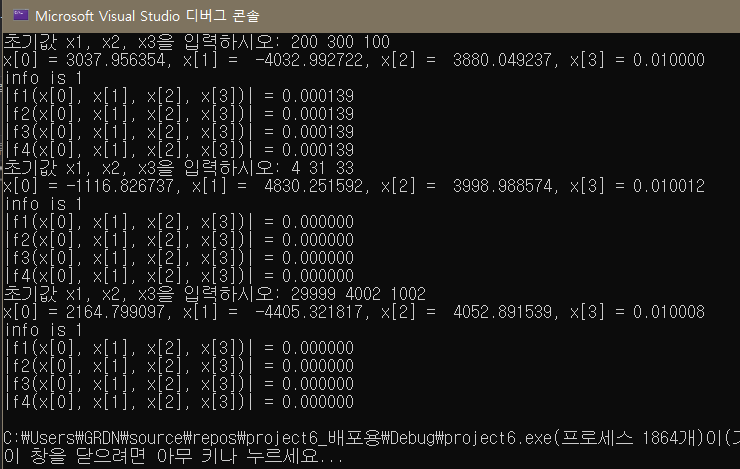


우선 초기값 x1, x2, x3을 입력받고, 입력 파일을 읽어들여 2차원 배열 p[4][5]에 저장한다. HYBRJ1 함수를 쓰기 위해 f3\_1\_a의 fvec에는 f1, f2, f3, f4 함수 정보를 입력하고, fjac에는 각 x에 대한 미분 함수를 입력한다. 이후 hybrj1\_() 함수를 호출하여 결과를 출력한다. 출력 결과는 다음과 같다.



info가 1이 출력되고, 각 근을 넣었을 때 함수값이 0이 되거나 0에 가깝게 되어 제대로 된 근을 구했다는 것을 확인할 수 있었다.

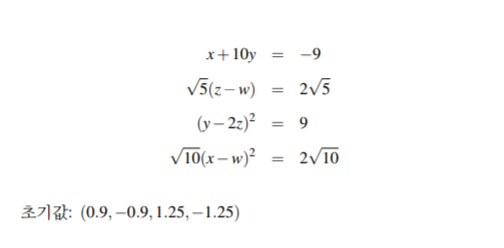
HYBRD1 함수를 이용하여 얻은 결과는 다음과 같다. f3\_1\_b에 fvec에 함수를 지정해 주었다.



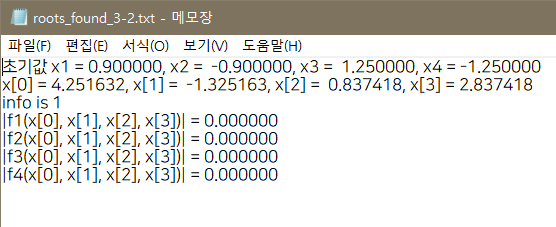
HYBRJ1과 마찬가지로 info가 1이 출력되고, 각 근을 넣었을 때 함수값이 0이 되거나 0에 가깝게 되어 제대로 된 근을 구했다는 것을 확인할 수 있었다. 두 방법 모두 근을 구하는 데는 지장이 없었으나, wa 값과 lwa값을 다르게 설정해 주어야 하는 것과 미분 함수의 유무에 주의해야 한다는 것을 알 수 있었다.

**숙제 3-2 (program3\_2.cpp)**

minpack에서 제공하는 적절한 FORTRAN 함수를 사용하여 다음과 같이 정의되는 비선형 방정식 시스템의 근을 찾아주는 프로그램이다.



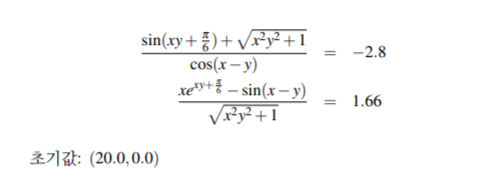
미분하기 쉽다고 판단하여 미분 함수가 필요한 HYBRJ1 함수를 이용하여 근을 구하였다. fvec과 fjac은 f3\_2에 지정해 주었다. roots\_found\_3-2.txt 결과는 다음과 같다.



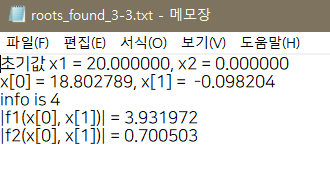
info가 1이 출력되고, 각 근을 넣었을 때 함수값이 0이 되어 제대로 된 근을 구했다는 것을 확인할 수 있었다.

**숙제 3-3 (program3\_3.cpp)**

minpack에서 제공하는 적절한 FORTRAN 함수를 사용하여 다음과 같이 정의되는 비선형 방정식 시스템의 근을 찾아주는 프로그램이다.



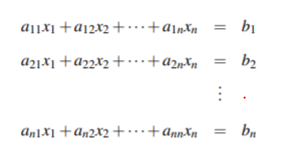
미분 함수를 구하기 어렵다고 판단하여 HYRBD1 함수를 이용하였다. roots\_found\_3-3.txt 결과는 다음과 같다.



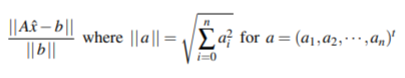
info가 4(Iteration is not making good progress)가 나오는 것을 보아 근을 제대로 구하지 못하는 것을 확인할 수 있었다.

**숙제 3-4 (program3\_4.cpp)**

수업 시간에 다룬 FORTRAN 함수 GESPP()와 SOLVE를 사용하여 선형 방정식의 근을 구하는 프로그램이다. 선형 방정식은 다음과 같다.



linear\_system\_3-4.txt를 읽어들여 n, a, b를 차례대로 저장한다. 이후 gespp\_() 함수와 solve\_() 함수를 호출하여 근을 구한다. solution\_3-4.txt의 첫 줄에는 n값, 다음 n줄에는 x 벡터들의 원소들, 마지막 줄에는 오차에 대한 척도를 저장한다. 오차에 대한 척도는 다음과 같이 정의된다.



제공받은 선형 방정식들에 대하여 실험을 진행한 결과 다음과 같이 solution\_3-4.txt.에 저장되었다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| General\_2.txt | General\_3.txt | General\_4.txt | General\_5.txt | General\_7.txt |
| 2  0.218763  -0.866386  **2.119797** | 3  2.805151  0.409502  -1.379471  **0.966279** | 4  34.421566 28.507507 24.180370 5.046705 **25.597275** | 5  7.466182  12.709721  1.159172  5.395200  8.344470  **6.629945** | 7  -1.440953  5.036052  -1.522058  -4.341182  4.106568  4.118313  3.882281  **2.434452** |
| General\_9.txt | General\_12.txt | General\_16.txt | General\_24.txt | General\_32.txt |
| 9  -7.237464  5.581297  -3.719851  -10.183822  -7.769780  5.204902  -1.730391  5.897697  -7.470247  **3.760681** | 12  -6.269248  4.251198  -2.303734  1.299261  -4.920690  -2.504002  -8.300145  -1.477556  -4.339134  -5.203034  4.618657  -6.976037  **3.003222** | 16  -2.009309  -2.015326  -3.782659  -4.089380  2.096344  1.051163  1.866920  -2.014789  3.214120  1.545003  -3.086546  0.515974  10.025562  10.336598  -6.599082  7.050585  **2.404150** | 24  5.115118  2.451864  -2.927811  1.426210  -1.157417  6.823476  1.056421  2.810616  5.904284  1.530419  -1.622561  1.796024  -1.728250  -1.862828  7.445944  5.499241  4.109170  4.795875  2.398479  5.162682  -2.039442  -7.085075  5.269317  -4.398377  **4.899178** | 32  0.594597  -15.868876  -3.322563  -1.918715  5.798898  -1.317963  8.759420  -3.167297  4.739877  0.772941  -3.921768  2.646138  6.719144  -1.961146  -5.366654  4.848794  -9.346067  8.027716  4.454370  3.521896  5.798513  0.620762  4.010528  2.538385  -3.399803  3.821739  -10.696643  -2.657655  4.115678  -2.505623  -6.291663  -6.755311  **3.177284** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hilbert\_2.txt | Hilbert\_3.txt | Hilbert\_4.txt | Hilbert\_5.txt | Hilbert\_7.txt |
| 2  1.199997  1.200005  **0.332819** | 3  1.200003  1.199981  1.200021  **0.873005** | 4  1.199931  1.200762  1.198188  1.201169  **1.399125** | 5  1.200397  1.193289  1.227062  1.161077  1.218346  **1.945682** | 7  1.199610  1.212148  1.114923  1.412109  1.013125  1.201968  1.246869  **3.059090** |
| Hilbert\_9.txt | Hilbert\_12.txt | Hilbert\_16.txt | Hilbert\_24.txt | Hilbert\_32.txt |
| 9  1.195502  1.312529  0.501077  2.918451  -0.745466  2.674514  -0.626610  3.060824  0.509489  **4.387221** | 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  **5.894341** | 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  **8.089518** | 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  **12.501135** | 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  **16.655687** |

오차에 대한 척도(굵은 글씨)를 비교했을 때, General 방정식들은 n이 증가할수록 일정한 값에 큰 편차 없이 수렴하는 경향을 보였다. 그러나 Hilbert 방정식은 n이 증가할수록 오차에 대한 척도도 일정하게 증가하는 경향을 보였다. 오차에 대한 척도가 적다는 것은 A의 조건이 적절하다는 가정 하에 찾은 근에 작은 relative error가 있다는 것을 의미한다. 그러나 오차에 대한 척도가 크다는 것은 행렬에 큰 backward error가 존재하고, 근을 찾기 위해 쓴 알고리즘이 불안정하다는 것을 의미한다. General 방정식에서는 n이 클수록 알고리즘이 비교적 안정적으로 작동하고 행렬에 큰 문제가 없는 것을 확인하였다. 그러나 Hilbert 방정식에서는 n이 클수록 알고리즘이 점점 더 불안정적으로 작동하고 행렬에 문제가 생기는 것을 확인할 수 있었다. 그러나 n이 2나 3과 같이 아주 작을 때는 General 방정식보다 Hilbert에 알고리즘이 안정적으로 작동하는 것을 확인하였다.