**고급소프트웨어실습1**

**6주차 과제**

**20161663 허재성**

**실습 결과**

실습은 사용자가 특별한 입력을 할 필요 없이 실행 시키면 콘솔 또는 출력 파일로 결과를 확인할 수 있다.

**실습 문제 3-1**은 미지수가 4개인 선형 방정식을 FORTRAN 함수를 저장하고 있는 gespp.f와 solve.f를 이용하여 가우스-조던 소거법으로 근을 구하였다. 근이 알려진 선형 방정식

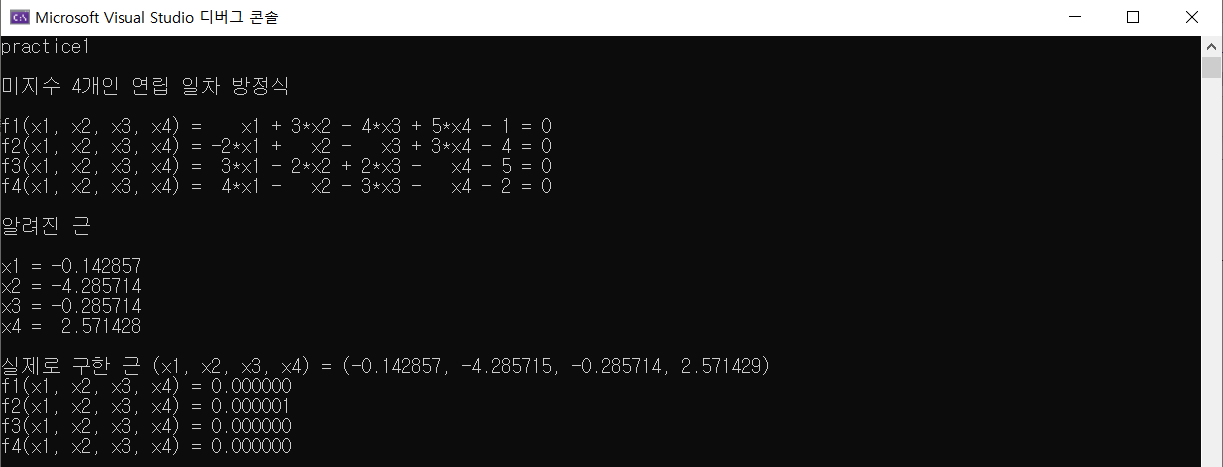
f1(x1, x2, x3, x4) = x1 + 3\*x2 - 4\*x3 + 5\*x4 - 1 = 0

f2(x1, x2, x3, x4) = -2\*x1 + x2 - x3 + 3\*x4 - 4 = 0

f3(x1, x2, x3, x4) = 3\*x1 - 2\*x2 + 2\*x3 - x4 - 5 = 0

f4(x1, x2, x3, x4) = 4\*x1 - x2 - 3\*x3 - x4 - 2 = 0

를 하드 코딩하여 방정식을 만든 후 해를 구해 콘솔에 출력하였다. 이론상 알려진 근은

x1 = -0.142857, x2 = -4.285714, x3 = -0.285714, x4 = 2.571428 이다. 실습 결과는 다음과 같다. 

FORTRAN 소프트웨어를 이용하여 가우스 조던 소거법으로 근을 구한 후 구한 근을 알려진 근과 비교해본 결과 근을 제대로 구했음을 알 수 있다. 또한 원래 방정식에 대입한 결과도 0에 가까운 값이 나오는 것을 확인할 수 있다.

**실습 문제 3-2**는 FORTRAN의 RPOLY 함수를 이용하여 다항 방정식의 근을 구하였다. 이때 복소수 근이 나올 수 있음을 유념하여 n차 방정식의 복소수 근까지 포함한 n개의 근을 구했으며 구한 근과 근을 원래 방정식에 대입했을 때 결과가 0이 되는 것을 roots\_3\_2\_i.txt 파일에 저장하였다.

**실습 문제 3-3**은 FORTRAN의 HYBRJ1 함수를 사용해 비선형 연립방정식의 근을 구하기 위해 노력했다. 하지만 hybrj1\_ 함수를 이용해 구한 근을 원래 방정식에 대입했을 때 0(0에 가까운 값)이 나오지 않았다. wolframAlpha에서 해당 방정식을 입력해 근을 알아본 결과 실근을 구할 수 없음을 확인했다. 실근을 구할 수 없음을 root\_3\_3.txt에 작성하였다.

**실습 문제 3-4**에서도 마찬가지로 FORTRAN의 HYBRJ1 함수를 사용해 비선형 연립방정식의 근을 구하였다. 3-3과 달리 근을 구할 수 있었다. 첫 번째로 주어진 초기값을 이용하면 정수 근을 구할 수 있었고, 두 번째로 주어진 초기값을 이용하면 실수 근을 구할 수 있었다. 대입 후 0이 되는 것도 확인하여 그 결과를 root\_3\_4.txt에 저장하였다.

**실습 문제 3-5**에서는 FORTRAN의 HYBRD1 함수를 사용해 비선형 연립방정식의 근을 구구했다. 구한 근과 근을 대입한 값이 0인 것을 root\_3\_5.txt에 저장하였다.

**실습 문제 3-6**에서는 HYBRJ1 함수를 이용하여 -4 <= x <=4, -5 <= y <= 5의 범위의 근을 찾기 위해 해당 범위의 (x, y) 순서쌍 중 x, y 모두 정수인 경우를 초기값으로 주어 근을 구하였다. 중복된 근을 포함하여 여러 개의 근을 구할 수 있었으며 각 근을 대입했을 때 값이 0이 되는 것도 확인했다. 그 결과를 root\_3\_6.txt에 저장하였다.

**실습 문제 3-7**에서는 3-6과 다르게 HYBRD1 함수를 이용하여 -4 <= x <=4, -5 <= y <= 5의 범위의 근을 찾기 위해 해당 범위의 (x, y) 순서쌍 중 x, y 모두 정수인 경우를 초기값으로 주어 근을 구하였다. 중복된 근을 포함하여 여러 개의 근을 구할 수 있었으며 각 근을 대입했을 때 값이 0이 되는 것도 확인했다. 그 결과를 root\_3\_7.txt에 저장하였다.

**실습 문제 3-8**에서는 비선형 방정식과 야코비 행렬, 초기값이 주어졌으므로 HYBRJ1을 이용하여 바로 근을 구한다. 이론적으로 알려진 근과 실제 구한 근이 얼마나 비슷한지 확인해볼 수 있으며 실제로 구한 근을 방정식에 대입해 0을 얻어 근이 정확한 것을 알았다. 이를 root\_3\_8.txt에 저장하였다.

숙제 또한 숙제 3-1에서 GPS 위치를 찾기 위한 초기값 입력을 제외하면 사용자가 특별히 입력할 것은 없다.

**숙제 3-1**에서는 4, 5, 6주차 수업 시작 전에 언급되었던 GPS 수신기의 위치를 찾는 문제를 해결하였다. 

총 3가지 경우의 수신기 위치를 구하는 문제가 주어졌으며 test case는 GPS\_signal\_i.txt(i =0 , 1, 2)에 저장되어 있다. 각 test case에 대하여 HYBRJ1을 이용하여 한 번, HYBRD1을 이용하여 한 번, 각각 따로 근을 구하였다. HYBRJ1을 이용한 근과 그 정확도(대입했을 때 0에 수렴하는지)는 GPS\_position\_3-1\_i.txt에 저장되어 있으며, HYBRJ2의 결과는 GPS\_position\_3-2\_i.txt에 저장되어 있다. 각 test case에 대하여 HYBRJ1과 HYBRD1이 근을 비슷하게 찾아내는지 확인하는 게 목적이므로 콘솔에 위치의 초기값 x1, x2, x3는 각 test case마다 같은 값을 주었다. 결과를 확인해 보면 두 경우 모두 근을 잘 구하는 것을 알 수 있다. 즉 GPS 위치를 잘 찾아내는 것을 알 수 있다.

**숙제 3-2**에서는 미지수가 4개인 비선형 연립방정식을 풀었다. 비록 비선형 연립방정식이지만 그 형태가 다항식 형태라 편미분을 구하기 용이해 야코비 행렬을 쉽게 구할 수 있었다. 야코비 행렬을 이용해 HYBRJ1으로 근을 구하고 방정식에 대입해 검증해보면 근을 제대로 구했음을 알 수 있다. 이를 roots\_found\_3-2.txt에 저장하였다.

**숙제 3-3**에서는 미지수가 2개인 비선형 연립방정식을 풀기 위해 노력했다. 미지수가 2개지만 삼각함수와 지수 함수, 무리함수가 섞인 복잡한 함수라 편미분을 구하기 어려우 야코비 행렬을 구하기 어렵다. 그래서 대신 HYBRD1을 이용해 원래 방정식만으로 근을 찾기 위해 노력했다. 하지만 근을 구한 후 원래 방정식에 대입한 결과 0을 얻지 못했다. 위에서 언급한 wolframAlpha에서 대신 근을 찾으려 해도 계산 시간이 초과되어 근을 찾을 수 없었다. 방정식에 찾은 근을 대입한 결과 밑 근을 찾지 못한 결과를 roots\_found\_3-3.txt에 저장하였다.

**숙제 3-4**에서는 실습 3-1에서 특정한 미지수가 4개인 선형 방정식을 푼 것을 일반화하여 최대 미지수의 개수가 32개인 선형 방정식을 풀 수 있는 프로그램을 작성하였다. Linear\_system\_3-4.txt에 저장된 방정식을 형식에 맞게 읽어들여 근을 찾아 solution\_3-4.txt에 저장하였다. 근을 찾기 위해 가우스 조던 소거법을 이용해야 하므로 실습 1과 마찬가지로 gespp.f와 solve.f를 사용하였다.

Solution\_3-4.txt에는 미지수의 개수 n과 n개의 미지수의 값, 그리고 근(n개의 미지수)의 오차에 대한 척도를 저장하였다.

오차의 척도 공식에 의하면 ||AX – b||가 분자에 있으므로 구한 근이 실제 근에 가까울수록 분자가 0에 가까워 진다. 따라서 오차의 척도가 0에 가까울수록 실제 근에 가까운 근을 구한 것을 알 수 있다.

|  |  |
| --- | --- |
| n | 오차의 척도 |
| 9 | 0.257075 |
| 16 | 0.667847 |
| 24 | 0.521255 |
| 32 | 0.244104 |

미지수의 개수가 32개로 많을 때 오차가 작으므로 미지수의 개수가 적을 때 보다 많을 때 상대적으로 적은 오차로 더 빠르게 계산할 수 있으므로 유용하다. 반대로 미지수의 개수가 적을 경우 높은 오차를 감안할 만큼 매력적이지 않을 수 있다.