고급소프트웨어실습I

6주차 보고서

20171646 박태윤

**- 프로그램 구동 방법 및 간략한 소개**

#include "my\_solver.h"

int main(void)

{

//practice3\_1();

//practice3\_2();

//practice3\_3();

//practice3\_4();

//practice3\_5();

//practice3\_6();

//practice3\_7();

//practice3\_8();

//program3\_1();

//program3\_2();

//program3\_3();

//program3\_4();

return 0;

}

main함수를 다음과 같이 작성하였다. practice3\_1() ~ practice3\_8()은 실습 프로그램을 나타내며 program3\_1() ~ program3\_4()는 과제 프로그램을 나타냅니다.



특히 program3\_1(과제 1번)같은 경우에는 x[0], x[1], x[2]를 콘솔창에서 입력을 받습니다. x[3]은 텍스트파일의 b값으로 설정이 되기 때문에 따로 입력을 받지 않습니다.

모든 프로그램들이 콘솔창에 실행 결과가 출력이 되며 이에 대응되는 텍스트 파일에 해당 결과를 작성합니다.

추가적으로, 과제4번에 대응되는 program3\_4는 다양한 텍스트파일을 적용시킬 수 있는데, 코드 17번째 줄 안의 텍스트 파일 이름을 바꿔서 실행시키면 됩니다.

**- 실습1**

practice3\_1()에 해당하는 프로그램입니다. 실행을 하면 다음과 같은 결과를 얻을 수 있습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Ax = b선형방정식에서 행렬 A와 B를 임의로 설정하여 gespp\_, solve\_함수를 이용해 해당 방정식의 근을 구하는 프로그램입니다. gespp\_를 실행하면 LU decomposition을 실행하는데 이에 대한 결과를 <<< Result of gespp\_ >>>밑에 출력을 하였으며 이 행렬을 가지고 solve\_함수를 이용해 근을 구합니다. 실제 근은 x1 = -6, x2 = -7 – 2\*x4, x3 = 0, x4 = x4인데 적절하게 구한 것을 확인할 수 있습니다.

**- 실습2**

practice3\_2()에 해당하는 프로그램입니다. 이를 실행하면 다음과 같은 결과를 얻을 수 있습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

총 6개의 방정식에 대해 근을 구하는 프로그램입니다. 각각의 결과는 roots\_1.txt ~ roots\_6.txt 텍스트 파일에 작성이 됩니다. rpoly함수를 이용하여 근을 구하는데, a + bi로 계산된 근이 실수부는 zeror배열, 허수부는 zeroi배열에 저장이 됩니다. 구해진 근이 적절한지를 판단하기 위해 zeror, zeroi배열에 저장된 근의 정보를 가지고 다시 해당 식에 대입을 하여 얻은 결과를 |f(n)|이라는 줄에 출력을 하였는데, 근이 적절하다면 f(근) = 0이라는 결과가 나와야 하고 실제 결과도 0.00000을 출력하고 있기 때문에 올바른 근을 구했음을 확인할 수 있습니다.

**- 실습3**

세 방정식으로 이루어진 연립방정식의 근을 hybrj1함수를 이용하여 구하는 프로그램입니다. practice3\_3()에 대응이 되며 이를 실행하면 다음과 같은 결과를 얻을 수 있습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

각각 초기값은 First : (0.0, 0.0, 0.0), Second : (1.55, 1.39, 1.10)으로 설정하여 hybrj1함수를 통해 근을 구하였고 구해진 근들이 적절한 지를 판단하기 위해 이를 다시 식에 대입한 결과인 fvec배열 값을 Func줄에 출력을 하였습니다. Func결과가 0.0000…이 아닌 것으로 보아 적절한 근을 구하지 못한 것을 확인할 수 있습니다.

- **실습4**

세 방정식으로 이루어진 연립방정식의 근을 hybrj1함수를 이용하여 구하는 프로그램입니다. practice3\_4()에 대응이 되며 이를 실행하면 다음과 같은 결과를 얻을 수 있습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이 역시 초기값을 각각 (0.1, 1.2, 2.5)와 (1.0, 0.0, 1.0)으로 서로 다르게 설정하여 hybrj1를 이용해 방정식의 근을 구하였다. Func값이 0.000000이 출력된 것으로 보아 적절한 근을 구한 것을 확인할 수 있으며 초기값에 따라 다른 근들을 구한 것을 확인할 수 있다.

**- 실습5**

세 방정식으로 이루어진 연립방정식의 근을 쟈코비안 행렬을 필요로 하는 hybrj1함수가 아닌 hybrd1함수를 이용하여 구하는 프로그램입니다. practice3\_5()에 대응이 되며 이를 실행하면 다음과 같은 결과를 얻을 수 있습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이 역시 초기값을 각각 (1.0, -1.0, 1.0)와 (1.0, 1.0, -1.0)으로 서로 다르게 설정하여 hybrd1를 이용해 방정식의 근을 구하였다. Func값이 0.000000이 출력된 것으로 보아 적절한 근을 구한 것을 확인할 수 있으며 초기값에 따라 다른 근들을 구한 것을 확인할 수 있다.

**- 실습6**

두 방정식으로 이루어진 연립방정식의 근을 hybrj1함수를 이용하여 구하는 프로그램입니다. 미지수로 x와 y가 주어지는데, 각각 범위를 [-4,4], [-5,5]로 설정하여 이 범위내의 근을 모두 구합니다. practice3\_6에 대응이 되며 이 프로그램을 실행하면 다음과 같은 근을 얻을 수 있습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

x는 -4에서 시작해서 4까지 0.5씩 증가, y는 -5에서 시작해서 5까지 0.5씩 증가시키면서 hybrj1함수를 호출하였다. 함수를 한 번 실행할 때 마다 근을 구할 수 있는데 이 때 중복되는 근을 두 번 출력하는 문제를 해결하기 위해 thereIsRoot함수를 따로 선언하였다. 구해진 근의 정보를 rootComp라는 배열에 저장을 하였는데, rootComp안에 현재 구해진 근이 존재한다면 출력하지 않고 continue문을 이용하여 넘어갔다. Func값이 0.000000을 출력하는 것을 통해 적절한 근을 구했음을 확인할 수 있다.

**- 실습7**

두 삼차방정식으로 이루어진 연립방정식의 근을 hybrd1함수를 이용하여 구하는 프로그램입니다. 실습6에서와 마찬가지로 x, y를 각각 [-4, 4], [-5, 5]범위로 설정하여 이 안의 근을 모두 구하는 프로그램입니다. practice3\_7에 해당하며 이를 실행시키면 다음과 같은 결과를 얻을 수 있습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이 역시 실습6에서 쓰인 thereIsRoot함수를 이용하여 중복되는 근이 여러 번 출력이 되는 문제를 해결하였으며, Func값이 0.000000을 나타내는 것을 보아 적절한 근을 해당 범위 내에서 구한 것을 확인할 수 있습니다.

**- 실습8**

3개 방정식으로 이루어진 연립방정식의 근을 구하는 프로그램입니다. hybrj1함수를 이용하여 이를 구하였으며, 초기값을 (0.1, 0.1, -0.1)로 설정하여 근이 (0.5, 0.0, -0.52359877)로 수렴하는지를 확인하는 프로그램입니다. practice3\_8에 해당하며 이를 실행시키면 다음과 같은 값을 얻을 수 있습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

근이 실습에서 주어진 (0.5, 0.0, -0.52359877)에 수렴하는 것을 확인할 수 있으며 이 근은 Func값이 0.000000인 것을 보아 적절한 근인 것을 확인할 수 있습니다.

- 숙제1

hybrj1와 hybrd1함수를 각각 이용하여 GPS수신기의 위치를 찾아주는 프로그램입니다. program3\_1에 해당하며, GPS수신기 식에 대한 정보는 GPS\_signal\_i.txt에 존재하여 이를 읽고 C, b, p11 ~ p43, t1~t4, tr1 ~ tr4를 구해 동작을 진행하였습니다. C는 double C, b는 double b에 해당하며 나머지 변수들은 double타입 배열 arr에 모두 대입을 하였습니다. 프로그램을 실행시키면 다음과 같은 결과를 얻을 수 있습니다.

|  |  |
| --- | --- |
|  |  |

총 4개인 변수 x1, x2, x3, x4에서 b값으로 세팅이 되는 x4를 제외하고 x1, x2, x3를 콘솔창에서 입력받습니다. b값이 0.010000, 0.010012, 0.010008로 모두 다른데, 이에 따라 나머지 x1, x2, x3도 달라지는 것을 확인할 수 있으며 Func값은 0.000000에 거의 수렴하는 값을 얻어 어느 정도 적절한 근을 얻은 것을 확인할 수 있습니다. 출력 결과에서 왼쪽 사진이 hybrj1을, 오른쪽 사진이 hybrd1을 이용하여 근을 구한 결과인데 결과 자체는 두 방법이 모두 같은 것 또한 확인할 수 있습니다. hybrd1함수가 hybrj1에 비해서 쟈코비안 행렬을 구하지 않아도 되기 때문에 구현 면에서는 더 편리한 점이 존재하지만 함수를 동작시키는데 필요한 wa배열의 크기가 hybrd1이 더 크기 때문에 공간적인 면에서는 hybrj1이 이점이 있는 것을 확인할 수 있습니다. hybrd1함수는 또한 쟈코비안 행렬을 함수 내부에서 근사적으로 계산하여 이를 근을 구하는데 사용하기 때문에 계산이 hybrj1함수에 비해 조금 더 느린 단점이 존재합니다.

**- 숙제2**

4개의 방정식으로 이루어진 연립방정식의 근을 구하는 프로그램입니다. 숙제2에서는 쟈코비안 행렬을 필요로 하는 hybrj1함수를 이용하였으며, program3\_2에 해당하고 이를 실행시키면 다음과 같은 결과를 얻을 수 있습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

미지수가 총 x,y,z,w로 4개가 존재하는데, 초기값을 (0.9, -0.9, 1.25, -1.25)로 설정하여 계산을 진행하였고 Func값이 역시 0.000000을 나타내는 것을 보아 적절한 근을 구한 것을 확인할 수 있습니다. 방정식의 우변을 좌변으로 모두 이항하여 방정식 = 0꼴로 만들어 hybrj1함수를 실행시켰습니다.

**- 숙제3**

두 방정식으로 이루어진 연립방정식의 근을 구하는 프로그램입니다. 주어진 식의 미분이 다소 난해하기 때문에 쟈코비안 행렬을 직접 입력해야 하는 hybrj1함수가 아닌 hybrd1함수를 이용하여 근을 구하였습니다. program3\_3에 해당하며 이를 실행시키면 다음과 같은 결과를 얻을 수 있습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

오차가 다소 큰 것을 확인할 수 있다. 와 같은 분수 계산이 제대로 이루어지는 이유가 아닐까 생각이 들어 양변에 분모를 곱한 변형된 식으로 프로그램을 다시 돌렸다.

텍스트이(가) 표시된 사진

자동 생성된 설명

미지수가 x,y로 두개가 존재하고 이 초기값을 (20.0, 0.0)으로 지정하여 계산을 진행하였습니다. 같은 경우 과 같이, 같은 경우 로 적절히 변형하여 계산을 진행하였고, 구한 근에 대한 fvec행렬의 값을 나타내는 Func값은 0.000000을 나타내고 있습니다. 물론 이것만 보고 근이 적절하다고도 할 수 있겠지만, 조금 더 확실한 정보를 얻기 위해 다음과 같은 함수를 선언하여 프로그램 마지막에 동작시켰습니다.

void cal\_func(double a, double b) {

double x[2];

double arr[2];

double arr2[2];

x[0] = a;

x[1] = b;

arr[0] = ((sin(x[0] \* x[1] + M\_PI / 6.0) + sqrt(pow(x[0], 2) \* pow(x[1], 2) + 1)) / cos(x[0] - x[1])) + 2.8;

arr[1] = ((x[0] \* exp(x[0] \* x[1] + M\_PI / 6.0) - sin(x[0] - x[1])) / sqrt(pow(x[0], 2) \* pow(x[1], 2) + 1)) - 1.66;

arr2[0] = sin(x[0] \* x[1] + M\_PI / 6.0) + sqrt(pow(x[0], 2) \* pow(x[1], 2) + 1) + 2.8 \* cos(x[0] - x[1]);

arr2[1] = x[0] \* exp(x[0] \* x[1] + M\_PI / 6.0) - sin(x[0] - x[1]) - 1.66 \* sqrt(pow(x[0], 2) \* pow(x[1], 2) + 1);

printf("<<< 기존 식 >>>\n");

printf("%lf\t\t%lf\n", arr[0], arr[1]);

printf("<<< 변형을 한 식 >>>\n");

printf("%lf\t\t%lf\n", arr2[0], arr2[1]);

}

계산을 위해 분수로 되어 있는 부분을 양 변에 곱해서 hybrd1함수를 실행시켰습니다. cal\_func함수는 이를 통해 구한 근을 변형하기 전 그대로의 식에 대입을 시킨 결과와 양변에 분모를 곱하는 변형을 거친 식에 대입한 결과를 출력해주는 함수입니다. 위의 결과에서도 볼 수 있듯이, 변형하기 이전의 식에도 이 근을 대입하면 0.000000을 나타내는 것을 확인할 수 있습니다. 즉 적절한 근을 구했다고 할 수 있습니다.

**- 숙제4**

실습1에서와 같이 GESPP(), SOLVE()함수를 이용하여 특정 텍스트 파일에 적혀져 있는 연립선형방정식의 근과 해당 근에 대한 오차를 구하여 solution\_3-4.txt파일에 적는 프로그램입니다. 예시로 주어진 Hilbert\_4.txt를 이용하여 프로그램을 실행시켰을 때 다음과 같은 결과를 얻을 수 있었습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

코드에서 오차를 구하는 부분은 다음과 같습니다.

for (i = 0; i < n; i++) {

double sum = 0.0;

for (j = 0; j < n; j++) {

sum += back\_up\_a[i + j \* n] \* x[j];

}

sum -= back\_up\_b[i];

sum = pow(sum, 2);

AX\_B += sum;

B += pow(back\_up\_b[i], 2);

func[i] = fabs(sum);

}

back\_up\_a는 a행렬의 LU\_decompotion하기 이전의 값, back\_up\_b는 역시 b행렬의 LU\_decomposition하기 전의 값을 가지고 있습니다. (Ax-b)로 계산된 결과는 func행렬에 저장이 되며 이는 근을 출력할 때 옆에 같이 출력을 해주었습니다. 또한 문제에서 요구하는 를 계산해주기 위해, AX\_B와 B라는 변수를 사용하였습니다. ||Ax-b||를 표현하는 변수가 AX\_B, ||b||를 표현하는 변수가 B입니다. 이 반복문 안에서는 각자 필요한 값들을 제곱한 결과를 모두 더해주었습니다.

printf("\n\n<<< ||Ax-b|| / || b || >>>\n");

printf("%.5e\n", sqrt(AX\_B) / sqrt(B));

이후 이와 같이 루트를 취해준 값으로 출력을 해주었습니다. 결과가 3.08217e-08, 즉 0.0000000308217이라는 매우 작은 값을 나타내는 것을 보아 (0.00000…에 거의 수렴) 꽤 적절한 근을 구했음을 확인할 수 있습니다.

마지막으로 가장 수가 많은 Hilbert\_32.txt를 실행시켰고 이에 대한 결과입니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

+)