# 고급 SW 실습 I
# 공개 소프트웨어를 사용한 문제 풀이
# (실습 자료)

CSE4152

서강대학교 컴퓨터공학과

# 실습 안내

- 실습 결과물 확인
  - 프로그램 완성 후 담당 조교에게 확인을 받아야 하고 동시에 이를 사이버 캠퍼스 해당 제출함에 제출하여야 한다.
  - 제출할 파일 이름은 snnnnnnL06.cpp로 하여야 한다.
    - 여기서, nnnnnn은 자신의 학번 뒤 6자리.
  - 실습 결과 검사
    - 담당 조교가 결과를 검사하면서 제대로 알고 작성했는지 몇가지 작성 내용에 관한 질문을 할 수 있다.
    - 평가 사항이므로 이에 답을 제대로 못하면 감점할 수 있다.
    - 그러니, 프로그램을 작성할 때 내용을 이해하며 작성하여야 한다(질문이 있으면 주저 말고 조교에게 문의할 것)

(**주의**) 만일 파일의 nnnnnn을 자신의 학번 뒤 6자리로 바꾸지 않고, 그냥 snnnnnnL06.cpp 등으로 제출하면 **0점 처리**한다.

- 숙제가 있을 경우
  - 제출 파일 이름, 마감일 등을 지정해 줄 것이다.
- 제출 마감
  - 실습, 숙제 모두 제출 마감일이 지정되어 있다.
  - Late 제출은 허용하지 않는다. 사이버 캠퍼스가 효과적으로 late 제출을 받지 않을 것이다.
- 실습 시 검사를 못 받은 경우
  - 일단 완성하여 실습 프로그램 제출함에 마감 전 제출한다.
  - 10/18일 이후 조교가 지정한 일시 및 장소에서 검사를 받아야 한다.
  - 마감 전 제출 못한 경우도 조교에게 검사를 받아야 한다.
- 실습 프로그램을 제출했는데 검사를 받지 않은 경우
  - 반드시 검사를 받아야 한다(제출물을 무효화할 수 있다).

# Visual Studio Project 생성

- 생성 내용 및 방법
  - VS 콘솔 프로그램을 위한 프로젝트
  - VS2017 실행(1,2)
    - 파일 → 새로 만들기 → 프로젝트 → Visual C++ → 기타 → 빈 프로젝트 선택
    - 프로젝트 이름(예: swLab19f_04) → 폴더 선택 → 확인
  - Source File 폴더
    - 프로젝트 폴더가 있는 위치에 Source file들을 저장할 폴더를 하나 만든다(예: swLab19f_04_src)
    - 이 폴더에 자신이 작성한 프로그램과 입력 데이터를 저장하면 편리하다.

(1) VS2015, VS2019도 사용 가능할 것이다.
(2) X64에서의 작업은 같은 프로젝트에서 x64로 바꾸어 수행할 수 있다.

## ✦Fortran 함수 컴파일 및 사용

### ◆Fortran 함수 컴파일

- 조교가 제공한 Fortran>file_f 폴더의 파일을 확인하자.
- 필요시 fortran file을 컴파일해야 한다. 이를 위해서는 ~.f를 Fortran>Compiler로 복사한 후, 명령 프롬프트에서 다음과 같이 입력한다

```
g77.exe -c file_name.f
```

- 혹은, ~.f 가 있는 폴더에 g77.exe, f771.exe, as.exe를 복사하여 위의 명령으로 컴파일할 수 있디.
- 컴파일 결과는 object 파일로 ~.o 형태를 갖는다.

### ◆추후 원하는 기능의 함수가 필요할 경우 Netlib에서 검색하여 프로그램을 다운 받아 위 방법으로 object 파일을 만들 수 있다(1).

(1) http://www.netlib.org/

## Fortran 함수 사용

- C/C++ 프로그램에서 fortran 함수를 호출하려면 fortran object 파일을 Visual Studio에 등록하여야 한다.
- 먼저 필요한 ~.o 파일을 적당한 폴더에 저장한다(~.cpp 파일이 있는 폴더가 적당).
- 다음, 두가지 방법으로 ~.o 파일을 Visual studio에 등록할 수 있다.
  1. 솔루션 탐색기의 "리소스 파일"에 ~.cpp 파일을 등록하는 것과 동일하게 필요한 ~.o 파일을 등록(1)
  2. "프로젝트 → 속성 → 구성 속성 → 링커 → 입력 → 추가 종속성"에 컴파일해서 얻은 ~.o 파일을 추가
- 참고. 같은 프로젝트에서 여러가지 서로 다른 작업을 할 경우, ~.o 파일을 프로젝트에 추가/제외가 편리한 1번 방법을 추천한다.

(1) 다른 곳, 즉, "소스 파일" 등에 등록해도 된다.

# 실습에서 사용할 Netlib의 FORTRAN 함수들

## RPOLY

- $n$ 차 다항식 $p_n(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$에 대해 $p_n(x)$=0인 모든 근(1)을 구한다.
- Parameters

```
subroutine rpoly(op, degree, zeror, zeroi, fail)
```

n 개의 근이 존재하는데, 이중 일부는 허근(복소수)일 수도 있다.

- **degree** : $p_n(x)$의 **degree** $n$ (int)
- **op** : 배열 $[a_n, a_{n-1}, \cdots, a_1, a_0]$ (double)
- **zeror** : 근의 실수 파트 $[\alpha_0, \alpha_1, \cdots, \alpha_{n-1}]$ (double)
- **zeroi** : 근의 허수 파트 $[\beta_0, \beta_1, \cdots, \beta_{n-1}]$ (double)
- **fail** : $n+1$개의 근을 찾은 경우 0, 아니면 ≠ 0 (long int)

(1) Zero라고 부른다.

## Nonlinear 연립 방정식

- $n$ 개의 변수를 갖는 nonlinear 연립 방정식은 다음과 같은 형태로 주어진다

$$f_1(x_1, x_2, \cdots, x_n) = 0$$
$$f_2(x_1, x_2, \cdots, x_n) = 0$$

$$f_n(x_1, x_2, \cdots, x_n) = 0$$

- $f_1(\ldots), f_2(\ldots), \cdots, f_n(\ldots)$를 함수 벡터 $\mathbf{f}: \mathbb{R}^n \dashrightarrow \mathbb{R}^n$로 다음과 같이 정의하면,

$$\mathbf{f}(\mathbf{x}) = [f_1(x_1, \cdots, x_n), f_2(x_1, \cdots, x_n), \cdots, f_n(x_1, \cdots, x_n)]^T,$$

- 위 연립 방정식은 다음과 같이 나타낼 수 있다

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}$$

여기서, $\mathbf{x} = [x_1, x_2, \cdots, x_n]^T, \mathbf{0} = [0,0, \cdots, 0]^T$ 이다.

## Jacobian 행렬

- $\mathbf{f}(\mathbf{x})$를 $\mathbf{x}$에 대해 편미분한 것으로 다음과 같은 행렬이다(1)

$$\boldsymbol{F}(\mathbf{x}) = \left[\frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_1}, \frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_2}, \cdots, \frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_n}\right] = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_n(\mathbf{x})}{\partial x_n} \end{bmatrix}$$

- 함수 HYBRJ1에서는 방정식 $\mathbf{f}(\mathbf{x}) = \mathbf{0}$에 대한 근을 구하기 위하여 Jacobian 행렬 값 계산을 필요로 한다.

(1) 보통 Jacobian 행렬을 **J**로 표시하나 여기서는 **F**로 표시하기로 한다.

## Minpack 함수 HYBRJ1, HYBRD1

### 함수 HYBRJ1

- 변형된 Powell hybrid method를 사용하여 비선형 연립 방정식의 근을 구한다.
- Jacobian 행렬 계산이 필요한데, 이를 함수 형태로 작성하여 HYBRJ1의 인수로 제공해야 한다.
- 또한 **f(x)** 계산식도 제공해야 하는데, 이들 모두 C 언어로 작성 가능하다.

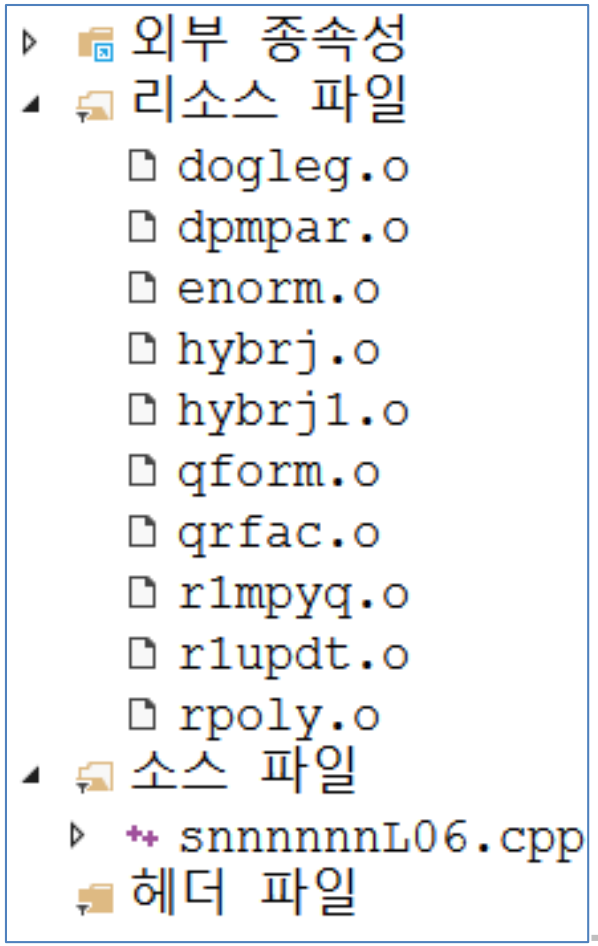
### 함수 HYBRD1

- HYBRJ1와 동일한 방법을 사용하지만 Jacobian 행렬 계산 값을 내부적으로 근사하여 사용한다.
- 따라서, 이 함수의 경우 Jacobian 계산식이 불필요 하다.

## 함수 HYBRJ1

- 이 함수를 사용하기 위해서는 다음과 같은 object file들이 필요하다(이들을 모두 아래 우측에 보인 것과 같이 Visual studio 리소스 파일에 등록해야 한다(1)).

**dogleg.o  dpmpar.o  enorm.o**
**hybrj.o  hybrj1.o  qform.o**
**qrfac.o  r1mpyq.o  r1updt.o**




(1) 혹은, 링커의 '추가 종속성'에 등록(6쪽 참조).

## ✦ 함수 정의

```
subroutine hybrj1(fcn, n, x, fvec, fjac, ldfjac,
                  tol, info, wa, lwa)
```

- **fcn** : $\mathbf{f}(\mathbf{x})$와 Jacobian 행렬 $\mathbf{F}(\mathbf{x})$를 구하는 user supplied 함수 이름으로 C/C++로 작성한다(1). 함수 정의 및 작성 방법은 다음 쪽에서 설명.
- ***n*** : 변수의 개수 (int)
- **x** : ***n*** 개의 변수로 구성된 배열. 함수 호출 시 초기 값을 저장하고, 종료 후에는 구한 근의 근사값 $\bar{\mathbf{x}}$가 저장된다(double)
- **fvec** : $\bar{\mathbf{x}}$에 대한 $f_i(\bar{\mathbf{x}}), i = 1, \ldots, n$, 배열(double).
- **fjac** : $\bar{\mathbf{x}}$에 대한 Jacobian $\mathbf{F}(\bar{\mathbf{x}})$의 QR factorization에서 Q 행렬 (double)
- **ldfjac** : **fjac**의 dimension. 즉, ***n*** (예전 강의 노트 참조) (int)
- **tol** : 근의 추정 값과 실제 근과의 오차 허용 값($\geq 0$). 이 값보다 작거나 같으면 함수 실행을 종료한다 (double)
- **info** : 실행 결과에 대한 flag(예전 강의노트 참조. 1이면 ok, int).
- **wa** : working array(double) of size **lwa** ($\geq n(n+13)/2$), int)

(1) 본래 fortran 함수로 제공하나 C/C++ 함수로 제공해도 된다. Fortran 함수 정의는 예전 강의 자료를 참조하자.

## ✦ 함수 **fcn( )**

◆ C/C++로 다음과 같이 정의하여 **hybrj1**의 인수로 제공

```
void fcn(int *n, double *x, double *fvec,
   double fjac[][MATCOLS], int *ldfjac, int *iflag);
또는
void fcn(int *n, double *x, double *fvec,
   double *fjac, int *ldfjac, int *iflag);
```

- ***n*** : 변수의 개수 (int)
- **x** : ***n*** 개의 변수로 구성된 배열. $\mathbf{f}(\mathbf{x})$ 또는 Jacobian $\mathbf{F}(\mathbf{x})$ 계산에 사용 (double).
- **fvec** : **x**에 대한 $f_i(\mathbf{x}), i = 1, \ldots, n$, 배열(double).
- **fjac** : **x**에 대한 Jacobian $\mathbf{F}(\mathbf{x})$ 행렬 (column wise, double)
- **ldfjac** : **fjac**의 dimension. 즉, ***n*** (예전 강의 노트 참조, int)
- **iflag** : 1이면 **fvec** 계산, 2이면 **fjac** 계산(다른 값은 불변, int)

## 함수 HYBRD1

- 이 함수를 사용하기 위해서는 다음과 같은 함수에 대한 object file ~.o 가 필요하다.

  **dogleg.o  dpmpar.o  enorm.o**

  **fdjac1.o  hybrd.o  hybrd1.o**

  **qform.o  qrfac.o  r1mpyq.o**

  **r1updt.o**

- HYBRJ1의 경우와 마찬가지로 위 object file들을 Visual Studio의 리소스 파일에 등록한다.
- HYBRJ1와 HYBRD1을 같이 사용할 경우 중복되지 않는 **fdjac1.o, hybrd.o, hybrd1.o**을 추가로 등록한다.

## ✦ 함수 정의

```
subroutine hybrd1(fcn,n,x,fvec,tol,info,wa,lwa)
```

- **fcn** : $\mathbf{f}(\mathbf{x})$를 구하는 user supplied 함수 이름으로 C/C++로 작성한다. 함수 정의 및 작성 방법은 다음 쪽에서 설명.
- ***n*** : 변수의 개수 (int)
- **x** : ***n*** 개의 변수로 구성된 배열. 함수 호출 시 초기 값을 저장하고, 종료 후에는 구한 근의 근사값 $\bar{\mathbf{x}}$가 저장된다(double)
- **fvec** : $\bar{\mathbf{x}}$에 대한 $f_i(\bar{\mathbf{x}}), i = 1, \ldots, n$, 배열(double).
- **tol** : 근의 추정 값과 실제 근과의 오차 허용 값($\geq 0$). 이 값보다 작거나 같으면 함수 실행을 종료한다(double).
- **info** : 실행 결과에 대한 flag(파일 hybrd1.f 참조. 1이면 ok, int).
- **wa** : working array(double) of size **lwa** ($\geq n(n+13)/2$), int).

## ✦ 함수 **fcn( )**

◆ C/C++로 다음과 같이 정의하여 **hybrd1**의 인수로 제공

```
void fcn(int *n, double *x, double *fvec,
         int *iflag);
```

- ***n*** : 변수의 개수 (int)
- **x** : ***n*** 개의 변수로 구성된 배열. $\mathbf{f}(\mathbf{x})$ 계산에 사용(double).
- **fvec** : **x**에 대한 $f_i(\mathbf{x}), i = 1, \ldots, n$, 배열(double).
- **iflag** : 이 함수 내에서 critical 오류 가 있을 경우 음수 값으로 설정하고 그 외에는 사용하지 않는다.

# 프로그램 구성 및 입력 형식

## ◆ 프로그램 구성

### ✦ 프로그램은 실습과 숙제 포함 총 7 문제를 해결하도록 구성되어 있다.

- **problem1()** : 다항식의 근을 구하는 함수
- **problem2_()** ~ **problem5_()** : nonlinear 연립 방정식을 구하는 함수 HYBRJ1 또는 HYBRD1에 필요한 fcn( )에 대응되는 함수
- **homework1_j()**, **homework1_d()** : 위와 마찬가지로 HYBRJ1 또는 HYBRD1의 fcn()에 대응되는 함수(숙제 문제)
- **solve_hybrj1()** : 함수 HYBRJ1 호출을 위한 준비 작업 후 HYBRJ1을 호출하여 근을 구하는 함수
- **solve_hybrd1()** : 함수 HYBRD1 호출을 위한 준비 작업 후 HYBRJ1을 호출하여 근을 구하는 함수

## 문제 선택

- 입력파일에서 문제 번호를 지정한다.
- 문제 번호는 **header.h**에 다음과 같이 지정되어 있다

```
#define p1 1        // problem 1
#define p2 2        // problem 2
#define p3 3        // problem 3
#define p4 4        // problem 4
#define p5 5        // problem 5
#define h1 6        // problem 6 (homework 1)
#define h2 7        // problem 7 (homework 2)
```

## 함수 포인터

```
// HYBRJ1의 fcn()에 대응되는 함수 포인터
void(*fcn_j)(int *, double *, double *, double *,
             int *, int *);
// HYBRD1의 fcn()에 대응되는 함수 포인터
void(*fcn_d)(int *, double *, double *, int *);
// 앞에서 보인 함수들은 이들 포인터에 설정되어 HYBRJ1또
// 는 HYBRD1에 fcn()으로 제공된다
```

## Global variables

### **header.h**에 다음과 같은 global 변수가 지정되어 있다.

- **int n** : 다항식인 경우 계수의 개수, 비선형 연립 방정식인 경우 미지수 개수
- **double *x** :
  다항식인 경우 다항식의 계수 배열(내림차순), 크기는 n-1
  비선형 연립 방정식인 경우 근의 초기값, 실행 후 근의 근사값
  배열 크기: 다항식인 경우 n-1, 비선형 연립 방정식인 경우 n
- **double *fvec** : n x n Jacobian 배열
- x와 fvec는 dynamic allocation을 통하여 메모리를 할당한다. 사용 후 반드시 deallocation하자.
- **int info** : 함수 HYBRJ1과 HYBRD1의 실행 결과를 알리는 flag
- 사전 주어진 코드에서 이들 변수를 사용하므로 이들은 반드시 필요한 곳에서 사용해야 한다.

## 프로그램 실행

- 표준 입출력을 사용하여 명령 프롬프트에서 아래 예로 보인 것과 같이 redirection을 통하여 실행한다.

```
실행파일 < in.txt > out.txt
```

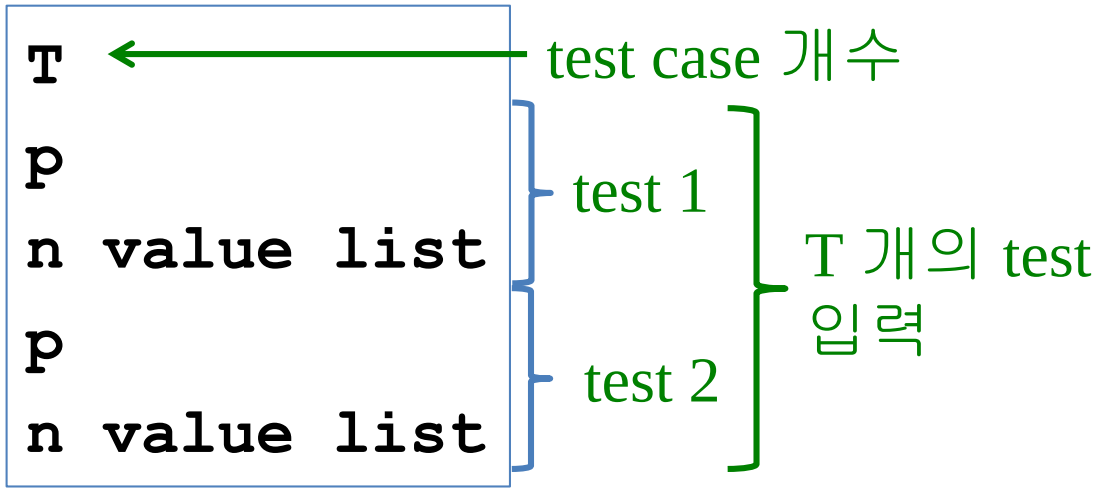
- 여기서, in.txt는 입력 파일, out.txt는 출력 파일이다.
- VS에서 디버깅하려면 프로젝트 속성 → 구성 속성 → 디버깅 → 명령 인수에 아래처럼 입력한 후 디버깅 작업을 한다.

| | |
|---|---|
| 명령 | $(TargetPath) |
| 명령 인수 | < in.txt > out.txt |

- 만일 출력을 보면서 디버깅하려면 > out.txt를 제거한다.

## 입력 데이터 형식

### 다음과 같은 형식으로 입력 데이터 파일을 만든다




- **p** : header.h에 정의된 문제 번호
- ***n*** : 다항식인 경우 : 다항식 차수
  비선형 연립 방정식인 경우 : 미지수 개수
- **value list**
  다항식인 경우 : **n+1** 개의 계수 값 리스트 (높은 차수부터 순서대로. 계수가 0 이어도 입력해야 함)
  비선형 연립 방정식인 경우 : **n** 개의 미지수에 대한 초기 값 리스트

## 문제를 추가하고자 할 경우

- 현재 7 개의 문제가 사전 정의되어 있다.
- 다른 다항식의 zeros 또는 비선형 연립 방정식의 근을 구하고자 한다면 다음과 같은 방법으로 프로그램을 수정한다.
- 다항식의 zero들을 구하고자 할 경우
  - 입력 파일에 문제 번호 1, 다항식 차수, 계수 리스트만 입력하면 된다.
- 비선형 연립 방정식의 근을 구하고자 할 경우
  - **`header.h`**에 새로운 문제 번호를 정의한다
  - **`fcn()`**에 대응하는 함수를 작성한다(**`progrma1_()`** 등과 같은 형태의 함수)
  - **`main`** 함수의 **`switch`** 문에 해당 문제 번호에 대한 코드를 추가한다.
  - 입력 파일에 문제 번호, 미지수 개수, 초기값 등을 추가.

# 실습 전 준비 및 연습

- 프로그램 복사 및 등록
  - 폴더 소스 코드에서 snnnnnnL06_std.cpp를 snnnnnnL06.cpp로 복사한다. 여기서, nnnnnn은 자신의 학번 뒤 6자리.
  - snnnnnnL06.cpp를 프로젝트 소스 파일 항에 등록한다.
  - header.h를 프로젝트 헤더 파일 항에 등록한다.
  - 폴더 Fortran > file_o에 포함된 ~.o 파일들을 프로젝트 리소스 파일 항에 등록한다(파일 링커_입력_추가종속성.txt 참조)

(1) 빌드시 오류가 생길 때 마다 필요한 것을 등록하면 꼭 필요한 object 파일만을 등록할 수 있다

- 이제 아래 보인 다항식의 근을 구해보자

$$f(x) = x^5 - x^4 - 4x^3 + 4x^2 - 5x - 75 = 0$$

  - 이의 근을 구하기 위한 입력은 다음과 같다(`in_0.txt`)

```
1 ← 테스트 개수
1 ← 문제 번호 1은 다항식의 근을 구하는 함수를 호출한다
5 1.0 -1.0 -4.0 4.0 -5.0 -75.0
```

다항식 차수 — 다항식 계수(높은 차수부터)

- 함수 **program1()**
  - 위 데이터로 프로그램을 실행하면 호출되며 **rpoly()**를 통하여 다항식의 근을 구한다.
  - 일부만 제외하고 이미 프로그래밍 되어 있다.
  - 프로그램에서 TO DO로 표기한 부분을 완성한 후 위 데이터를 입력으로 프로그램을 실행해 보자.

## 프로그램 출력

- 다음과 같은 출력이 생성됨을 확인하자

```
Test 0
*** Roots finding for an n-th degree polynomial using RPOLY
1.0000x^5 + -1.0000x^4 + -4.0000x^3 + 4.0000x^2 + . . .

fail = 0
r( 1) = (1.0000000000) + (2.0000000000) i
r( 2) = (1.0000000000) + (-2.0000000000) i
r( 3) = (-2.0000000000) + (1.0000000000) i
r( 4) = (-2.0000000000) + (-1.0000000000) i
r( 5) = (3.0000000000) + (0.0000000000) i

f(3.000000000000000) = 0.000000000000000
```

- 위에서 fail은 함수 **rpoly()**의 결과 flag이다. 값이 0 이면 제대로 근을 구한 것이다.
- 실근 $r_5$에 대해 $f(r_5) \cong 0$임을 확인할 수 있다.

# 실습 1 RPOLY함수 활용

- 아래 두 다항 방정식에 대한 근을 구해 보자

$$f_1(x) = x^3 + x^2 + 3x - 5 = 0$$
$$f_2(x) = x^6 - 2x^5 + 14x^4 - 26x^3 + 49x^2 - 72x + 36 = 0$$

  - 입력 파일을 만들자.
    - 테스트 개수는 2이다.
    - 두 테스트 모두 문제 번호는 1 번이다.
- 근을 구하여 실습 데모할 때 담당 조교의 확인을 받도록 하자.

# 실습 2 HYBRJ1 함수 활용

- 대상 비선형 연립 방정식

$$f_1(x_1, x_2, x_3) = 3x_1 - \cos(x_2 x_3) - \frac{1}{2} = 0$$
$$f_2(x_1, x_2, x_3) = x_1^2 - 81(x_2 + 0.1)^2 + \sin x_3 + 1.06 = 0$$
$$f_3(x_1, x_2, x_3) = e^{-x_1 x_2} + 20x_3 + \frac{10\pi - 3}{3} = 0$$

- 위 방정식의 근을 함수 HYBRJ1를 사용하여 구해보자
  - 이를 위하여 다음과 같이 $\mathbf{x}$와 $\mathbf{f}(\mathbf{x})$를 정의하자

$$\mathbf{x} = [x_1, x_2, x_3]^T,\ \mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), f_3(\mathbf{x})]^T$$

  - HYBRJ1의 parameters **x**, **fvec**, **fjac** 배열의 원소는 $\mathbf{x}$와 $\mathbf{f}(\mathbf{x})$의 원소 순서에 따르도록 한다.
  - 문제 번호는 2로 하며, **program2_()**와 **solve_hybrj1()**를 작성해야 한다.

## 함수 **problem2_()** 작성

- 이 함수의 위치가 포인터 **fcn_j**에 로드되어 **HYBRJ1**의 parameter **fcn()**으로 제공된다(**main()**의 **case p2** 참고).
- Parameter는 **fcn()**의 그것들과 같다

```
void problem2_(int *n, double *x, double *fvec,
               double *fjac, int *ldfjac, int *iflag)
```

- 입력으로 주어진 **x** 에 대해
  - **iflag=1**이면 **f(x)**를 계산하여 **fvec** 에 저장하고
  - **iflag=2**이면 Jacobian **F(x)**를 계산하여 **fjac** 에 저장한다(1).
  - 여기서, **fjac**는 column-major order로 구성된 $n \times n$ 배열인데, 1차원 배열에 column-major order로 편미분 값을 저장해도 된다.
- 이 함수를 작성하자.

(1) 9 쪽 Jacobian 정의 참조

## ◆ 함수 **solve_hybrj1()** 작성

- ✦ 이 함수에서 해야 할 과정은 다음과 같다
  1. 파일 **header.h**에서 선언한 **n**, **x**, **fvec**, **info**를 제외한 **hybrj1_()**에 필요한 parameter 변수 선언(12 쪽 참조).
  2. 배열의 경우 필요한 만큼 dynamic allocation(**x** 제외).
  3. 함수 **hybrj1_()**의 parameter 변수 초기 값 설정
  4. 함수 **hybrj1_()**호출
  5. 미지수 **x**의 초기 값, **info** flag 값, 구한 근의 근사 값 및 이들 각각의 f(x) 값 출력 (다음 쪽의 출력 format 참조)
  6. 이 함수에서 allocation한 메모리 deallocation
- ✦ 이 함수를 작성하자

## 실행 결과

### 입력

```
1
2
3 0.1 0.1 -0.1
```

### 출력(위 입력에 대해 다음과 같이 출력되도록 작성하자(1))

```
Test 0
*** Roots finding for nonlinear equations using HYBRJ1
  Initial values of X
    x_1 = 0.100000
    x_2 = 0.100000
    x_3 = -0.100000

  info = 1
  root=       0.50000000    0.00000000    -0.52359878
  f(x)=       0.00000000    -0.00000000   0.00000000
```

(1) 포맷이 완전히 일치하지 않아도 된다. 다만, 이쁘게 출력하자. '\t'을 사용하면 편리할 수도 있다.

# 실습 3 HYBRD1 함수 활용

- 대상 비선형 연립 방정식은 실습 2와 동일하다

$$f_1(x_1, x_2, x_3) = 3x_1 - \cos(x_2 x_3) - \frac{1}{2} = 0$$
$$f_2(x_1, x_2, x_3) = x_1^2 - 81(x_2 + 0.1)^2 + \sin x_3 + 1.06 = 0$$
$$f_3(x_1, x_2, x_3) = e^{-x_1 x_2} + 20x_3 + \frac{10\pi - 3}{3} = 0$$

- 이 방정식의 근을 함수 HYBRD1를 사용하여 구해보자
  - 문제 번호는 3으로 한다.
  - **`program3_()`**와 **`solve_hybrd1()`**를 작성해야 한다.

## ◆ 함수 **problem3_()** 작성

- ✦이 함수의 위치는 포인터 **fcn_d**에 로드되어 **HYBRD1**의 parameter **fcn()**으로 제공된다(**main()**의 **case p3** 참고).
- ✦Parameter는 **fcn()**의 그것들과 같다

```
void problem2_(int *n, double *x, double *fvec,
               int *iflag)
```

- ✦이 함수는 단순히 입력으로 주어진 **x**에 대해 **f(x)**를 계산하여 **fvec**에 저장하면 된다.
- ✦이 함수를 작성하자.

(1) 9 쪽 Jacobian 정의 참조

## ◆ 함수 **solve_hybrd1()** 작성

- ✦ 함수 **solve_hybrj1()** 과 거의 유사하게 다음과 같은 순서로 함수를 작성한다.
  1. 파일 **header.h**에서 선언한 **n**, **x**, **fvec**, **info**를 제외한 **hybrd1_()** 에 필요한 parameter 변수 선언(15 쪽 참조).
  2. 배열의 경우 필요한 만큼 dynamic allocation(**x** 제외).
  3. 함수 **hybrd1_()** 의 parameter 변수 초기 값 설정
  4. 함수 **hybrd1_()** 호출
  5. 미지수 **x**의 초기 값, **info** flag 값, 구한 근의 근사 값 및 이들 각각의 f(x) 값 출력 (다음 쪽의 출력 예 참조)
  6. 이 함수에서 allocation한 메모리 deallocation
- ✦ 이 함수를 작성하자

## 실행 결과

- 입력 (실습 2와 문제 번호외 동일)

```
1
3
3 0.1 0.1 -0.1
```

- 출력(위 입력에 대해 다음과 같이 출력되도록 작성하자(1))

```
Test 0
*** Roots finding for nonlinear equations using HYBRD1
  Initial values of X
    x_1 = 0.100000
    x_2 = 0.100000
    x_3 = -0.100000

  info = 1
  root=       0.50000000    0.00000000    -0.52359878
  f(x)=       0.00000000    -0.00000000   0.00000000
```

(1) 실습 2와 비교했을 때 **HYBRD1**만 제외하고 동일하다.

# 실습 4 HYBRJ1 함수 추가 연습

- 아래 보인 비선형 연립 방정식의 근을 구해보자

$$f_1(x_1, x_2, x_3) = e^{2x_1} - x_2 + 4 = 0$$
$$f_2(x_1, x_2, x_3) = x_2 - x_3 - 1 = 0$$
$$f_3(x_1, x_2, x_3) = x_3 - \sin x_1 = 0$$

- 참고 사항
  - 문제 번호는 4로 한다.
  - **program4_()**를 작성해야 한다.
  - 초기 값으로 $x_1 = 1.55, x_2 = 1.39, x_3 = 1.10$로 시도해보자.
  - 위 초기 값으로 어떤 근의 근사 값을 구하는지 관찰해보자.
  - 위 관찰에 info 값도 도움이 될 것이다.
  - 다음 실습 5도 동시에 진행해보자.

# 실습 5 HYBRD1 함수 추가 연습

- 실습 4와 동일한 아래 비선형 연립 방정식의 근을 HYBRD1을 사용하여 구해보자

$$f_1(x_1, x_2, x_3) = e^{2x_1} - x_2 + 4 = 0$$
$$f_2(x_1, x_2, x_3) = x_2 - x_3 - 1 = 0$$
$$f_3(x_1, x_2, x_3) = x_3 - \sin x_1 = 0$$

- 참고 사항
  - 문제 번호는 5로 한다.
  - **`program5_()`**를 작성해야 한다.
  - 실습 4와 마찬가지로 초기 값을 $x_1 = 1.55, x_2 = 1.39, x_3 = 1.10$로 시도해보자.
  - 실습 4의 결과와 비교해보자.

## 추가 사항

- 실습 4와 5를 수행한 결과를 관찰 분석하여 자신만의 결론을 내린 후 이에 관한 보고서를 작성하여 제출하여야 한다.