[CSE 4152] 고급 소프트웨어 실습 I

『GPS 수신기 위치 계산 문제』

수치 컴퓨팅 실험 3: 공개 소프트웨어를 사용한 문제 풀이

담당교수: 컴퓨터공학과 임인성 (AS-905, 02-705-8493, ihm@sogang.ac.kr)

담당조교: 컴퓨터공학과 안재풍 (AS-914, 02-711-5278, ajp5050@sogang.ac.kr)

1 비선형 방정식 시스템의 풀이

이제 첫 주에 제시한 GPS 수신기 위치 계산을 위한 비선형 방정식 시스템을 다시 한번 생각해보자.

$$\begin{aligned}
f_1(x_1,x_2,x_3,x_4) &= (x_1-p_{11})^2+(x_2-p_{12})^2+(x_3-p_{13})^2-\{C(tr_1+x_4-t_1)\}^2 = 0\\
f_2(x_1,x_2,x_3,x_4) &= (x_1-p_{21})^2+(x_2-p_{22})^2+(x_3-p_{23})^2-\{C(tr_2+x_4-t_2)\}^2 = 0\\
f_3(x_1,x_2,x_3,x_4) &= (x_1-p_{31})^2+(x_2-p_{32})^2+(x_3-p_{33})^2-\{C(tr_3+x_4-t_3)\}^2 = 0\\
f_4(x_1,x_2,x_3,x_4) &= (x_1-p_{41})^2+(x_2-p_{42})^2+(x_3-p_{43})^2-\{C(tr_4+x_4-t_4)\}^2 = 0
\end{aligned}$$

이러한 방정식 시스템은 다음과 같이 n개의 변수를 가지는 비선형 방정식 시스템으로 확장할 수가 있다.

$$\begin{aligned}
f_1(x_1,x_2,\cdots,x_n) &= 0\\
f_2(x_1,x_2,\cdots,x_n) &= 0\\
&\vdots\\
f_n(x_1,x_2,\cdots,x_n) &= 0
\end{aligned}$$

이때 함수 $\mathbf{F}:\mathbb{R}^n\rightarrow\mathbb{R}^n$을 다음과 같이 정의하면,

$$\mathbf{F}(x_1,x_2,\cdots,x_n) = (f_1(x_1,x_2,\cdots,x_n),\ f_2(x_1,x_2,\cdots,x_n),\cdots,\ f_n(x_1,x_2,\cdots,x_n))^t$$

그리고, $\mathbf{x}$와 $\mathbf{0}$을 각각 $(x_1,x_2,\cdots,x_n)$과 $(0,0,\cdots,0)^t$ 값을 가지는 벡터로 정의하면, 위의 비선형 방정식 시스템은 다음과 같이 기술할 수 있다.

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

앞에서 $n=1$인 경우의 방정식 $f(x)=0$에 대해 Newton-Raphson 방법을 적용할 경우 다음과 같은 반복식을 통하여 근을 구할 수 있다고 하였다.

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

이 방법의 원리를 n차원 공간으로 확장하면, 기본적인 대학 미적분 개념을 사용하여 비선형 방정식 시스템 $\mathbf{F}(\mathbf{x})=\mathbf{0}$에 대한 반복문이 다음과 같이 됨을 어렵지 않게 보일 수가 있다.

$$\boxed{\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - J(\mathbf{x}^{(k)})^{-1}\mathbf{F}(\mathbf{x}^{(k)})}$$

이 반복문은 $n=1$일 때의 반복문 $x_{k+1}=x_k-f'(x_k)^{-1}f(x_k)$와 기본적으로 동일한 형태임을 알 수 있다. 다만 차이는 한 개의 변수 x_k에 대한 값이 아니라 한 번에 n개의 실수 값으로 구성된 벡터 $\mathbf{x}^{(k)}=(x_1^{(k)},x_2^{(k)},\cdots,x_n^{(k)})$ 값을 동시에 계산하고 있다는 점이다. 또한 $J(\mathbf{x}^{(k)})^{-1}$는 $f'(x_k)^{-1}$에 대응되는 값인데, 여기서 $J(\mathbf{x})$가 바로 미적분 시간에 배운 야코비 행렬 (Jacobian matrix)임을 쉽게 추정할 수 있다.

즉 야코비 행렬은 다음과 같이 정의가 되는데,[1)]

$$\begin{aligned} J(\mathbf{x}) &= \frac{\partial(f_1,f_2,\cdots,f_n)}{\partial(x_1,x_2,\cdots,f_n)}(\mathbf{x}) \\ &= \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \frac{\partial f_1}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}) \\ \frac{\partial f_2}{\partial x_1}(\mathbf{x}) & \frac{\partial f_2}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_2}{\partial x_n}(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(\mathbf{x}) & \frac{\partial f_n}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_n}{\partial x_n}(\mathbf{x}) \end{pmatrix} \end{aligned}$$

$f'(x)$가 함수 $f(x)$의 1차 미분 정보를 제공하듯이, n행 n열 행렬인 $J(\mathbf{x})$에는 함수 $\mathbf{F}(\mathbf{x})$에 대한 1차 미분 정보가 담겨져 있다.

이제 위의 반복문 $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)}-J(\mathbf{x}^{(k)})^{-1}\mathbf{F}(\mathbf{x}^{(k)})$을 통하여 $\mathbf{F}(\mathbf{x})=\mathbf{0}$의 근 $\mathbf{x}$를 구할 수 있는데, 한 가지 문제는 매번 행렬 $J(\mathbf{x}^{(k)})$의 역행렬을 계산해주어야 한다는 사실이다. 하지만 수치 계산 시 어떤 행렬의 역행렬을 계산하는 과정은 일반적으로 비용이 많이 들 뿐만 아니라 종종 그 과정이 수치적으로 불안정하기 때문에 가급적 피하려 한다. 실제로 이 반복문을 계산할 때에도 $J(\mathbf{x}^{(k)})^{-1}$ 행렬을 직접 계산하지 않고, 다음과 같은 선형 방정식 시스템 (system of linear equations)을 풀어,[2)]

$$J(\mathbf{x}^{(k)})\,\mathbf{y} = \mathbf{F}(\mathbf{x}^{(k)})$$

반복문의 $J(\mathbf{x}^{(k)})^{-1}\mathbf{F}(\mathbf{x}^{(k)})$ 부분을 계산하는 방식을 취한다.

이를 위해서는 선형 방정식의 수치적인 풀이 방법에 대해서도 이해를 해야하는데, 본 실습에서는 시간적인 제약으로 인해 위 반복문을 직접 구현하지 않고, 비선형 방정식 시스템을 풀어주는 공개 소프트웨어를 사용하여 문제를 해결하여 보자. 다만 위에서 설명한 바와 같이 비선형 방정식 시스템의 근을 풀기 위해서는 야코비 행렬이 중요한 역할을 한다는 점을 다시 한번 명심하자.

1) 일반적으로 $J(\mathbf{x})$ 대신 $\mathbf{F}'(\mathbf{x})$로도 많이 표기하는데 여기서는 $J(\mathbf{x})$로 표기하자.

2) 여기서 $\mathbf{y}$는 $\mathbf{y}=(y_1,y_2,\cdots,y_n)^t$인 벡터임.

2 Netlib 저장소와 minpack

2.1 Netlib 저장소

C 언어가 존재하지 않았거나 컴파일러 기술 등이 부족하던 80년대 이전에는 상당한 양의 부동 소수점 연산 (floating-point operation)이 필요한 수치 계산을 효율적으로 수행하기 위하여 주로 FORTRAN 언어를 사용하여 수치 계산용 프로그램을 작성하였다. 그 결과 공개 도메인 (public domain)에는 지난 수십년간 개발하고 그 효율성과 안정성을 검증한 수치 계산 용 FORTRAN 코드가 많이 존재한다. 다음은 그 대표적인 사이트 중의 하나인 Netlib 저장소 (Netlib repository)에 대한 설명이다.[3)]

> The Netlib repository contains freely available software, documents, and databases of interest to the numerical, scientific computing, and other communities. The repository is maintained by AT&T Bell Laboratories, the University of Tennessee and Oak Ridge National Laboratory, and by colleagues world-wide. The collection is replicated at several sites around the world, automatically synchronized, to provide reliable and network efficient service to the global community.

이 곳에 공개되어 있는 코드는 `http://www.netlib.org`에서 검색하여 사용할 수 있는데, 2020년 10월 11일 현재까지 총 1,221,587,548의 코드 요청이 있었다고 한다.

2.2 minpack

Netlib 저장소가 제공하는 공개 소프트웨어 중 비선형 방정식 시스템의 근을 구하는데 사용할 수 있는 함수를 제공하는 minpack이라는 코드 패키지가 있는데, 이에 대한 설명은 다음과 같다.[4)]

> MINPACK is a library of FORTRAN subroutines for *the solving of systems of nonlinear equations*, or the least squares minimization of the residual of a set of linear or nonlinear equations.
>
> MINPACK, along with other similar libraries such as LINPACK and EISPACK originated from the Mathematics and Computer Science Division Software (MCS) of Argonne National Laboratory Written by Jorge Moré, Burt Garbow, and Ken Hillstrom MINPACK is free and designed to be highly portable, robust and reliable.

3) 출처: http://www.netlib.org/misc/faq.html#2.1

4) 출처: http://en.wikipedia.org/wiki/MINPACK

> Five algorithmic paths each include a core subroutine and an easy-to-use driver. The algorithms proceed either from an analytic specification of the Jacobian matrix or directly from the problem functions. The paths include facilities for systems of equations with a banded Jacobian matrix, for least squares problems with a large amount of data, and for checking the consistency of the Jacobian matrix with the functions.

본 실습에서는 minpack에서 제공하는 FORTRAN 함수 중에서 `HYBRD1`과 `HYBRJ1` 등 서로 다른 두 개의 함수를 사용하여 몇 가지 예제 비선형 방정식 시스템을 풀어본 후, 앞에서 설명한 GPS 수신기의 위치를 계산하는 문제에 적용하여 보자.

3 C와 FORTRAN 함수의 혼용을 통한 문제 해결

주어진 응용 문제에 적용 가능한 FORTRAN 함수를 찾았을 때, 자신이 작성하고 있는 C 프로그램에서 이 함수를 호출할 수 있는 방법은 다음과 같이 두 가지가 있다.

3.1 f2c를 통한 FORTRAN 프로그램의 자동 변환

첫째, 공개 소프트웨어인 f2c와 같은 프로그램을 통하여 자동으로 FORTRAN 코드를 C 코드로 변환한 후, 자신의 C 프로그램에서 호출하여 사용하는 방법이다. f2c 소프트웨어에 대한 간단한 설명을 보자.[5)]

> f2c is the name of a program to convert Fortran 77 to C code, developed at Bell Laboratories. The standalone f2c program was based on the core of the first complete Fortran 77 compiler to be implemented, the ”f77” program by Feldman and Weinberger. Because the f77 compiler was itself written in C and relied on a C compiler back end to complete its final compilation step, it and its derivatives like f2c were much more portable than compilers generating machine code directly.
>
> The f2c program was released as free software (open-source software) and subsequently became one of the most common means to compile Fortran code on many systems where native Fortran compilers were unavailable or expensive. Several large Fortran libraries, such as LAPACK, were made available as C libraries via conversion with f2c. The f2c program also influenced the development of the GNU g77 compiler, which uses a modified version of the f2c runtime libraries.

5) 출처: http://en.wikipedia.org/wiki/F2c

f2c에 대한 자료는 웹상에서 쉽게 얻을 수 있는데, 본 실습에서는 이 방법이 아닌 다음 방법을 사용하여 문제를 해결하려 하기 때문에 자세한 설명은 생략하도록 한다.

3.2 C 함수에서 FORTRAN 함수의 직접 호출

다음으로 생각할 수 있는 방법은 자신의 C 코드에서 직접 FORTRAN 함수를 호출하는 방법이다. 즉 무료로 구할 수 있는 FORTRAN 컴파일러를 통하여 해당 FORTRAN 함수를 컴파일하여 오브젝트 코드를 생성한 후, 이를 자신이 작성한 C 프로그램 컴파일 시 링크하여 사용하면 된다.

함수 호출 시 두 가지 주의 사항

이러한 방법이 제대로 작동하기 위해서는 C 함수에서 FORTRAN 함수 호출 시 함수 인자를 통하여 주고 받아야 하는 데이터가 올바르게 오가도록 해주는 과정이 필요한데, 이때 무엇보다도 다음과 같은 C 언어와 FORTRAN 언어와의 근본적인 차이에 주의해야 한다.

	C 언어	FORTRAN 언어
매개 변수 전달 방법	call by value	call by reference
행렬 저장 방법	row-major order	column-major order

첫째로, C 코드에서 FORTRAN 함수 호출 시 함수 인자를 통하여 데이터를 전달해야 하는데, 기본적으로 FORTRAN 함수쪽에서는 인자의 주소가 넘어오는 것으로 가정한다 (call-by-reference). 하지만 C 언어 쪽에서는 기본적으로 인자의 주소가 아니라 값이 넘어가기 때문에 (call-by-value), 이에 대해 적절한 처리를 하지 않을 경우 심각한 문제가 발생하게 된다. 따라서 C 함수 쪽에서 인자를 넘길 때, 반드시 포인터를 사용하여 주소를 넘겨주어야 한다.

또한, 수치 계산에서 널리 쓰이는 행렬을 2차원 배열에 담아 인자로 넘길 때, C 함수 쪽에서는 이 행렬 데이터가 행 우선 순서 (row-major order)로 1차원 배열에 저장되어 있다고 가정하나, FORTRAN 함수 쪽에서는 이 데이터가 열 우선 순서 (column-major order) 순서로 저장되어 있다고 가정하기 때문에 이로 인하여 잘못된 계산 결과를 산출하게 된다. 따라서 C 함수 쪽에서 행렬 데이터를 넘길 때에는 우선 그 행렬 자체가 아니라 그에 대한 전치 행렬을 배열에 담아 FORTRAN 함수로 넘기고, 또한 결과 데이터로 행렬 데이터를 리턴 받았다면, 이를 전치 행렬로 해석하여 사용하면 된다.

간단한 함수 호출 예

비선형 방정식 시스템 풀이 문제에 대하여 실습하기 전에, 먼저 간단한 방정식의 풀이 문제를 통하여 C 함수에서 FORTRAN 함수 호출 과정에 대한 연습을 해보자. 지금 아래와 같은 임의의 n차 다항식에 대하여,

$$p_n(x) \quad = \quad a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

방정식 $p_n(x) = 0$의 모든 근을 구하려 한다.[6)]

Netlib 저장소를 검색해보면, 일반 다항식 방정식의 모든 근을 구해주는 코드를 저장하고 있는 파일 `rpoly.f`를 찾을 수가 있다. 이 파일에는 RPOLY라는 FORTRAN 함수가 정의되어 있는데, 이 함수의 헤더 부분은 다음과 같다.

```
      SUBROUTINE RPOLY(OP, DEGREE, ZEROR, ZEROI,          RPO 10
     * FAIL)
C FINDS THE ZEROS OF A REAL POLYNOMIAL
C OP - DOUBLE PRECISION VECTOR OF COEFFICIENTS IN
C      ORDER OF DECREASING POWERS.
C DEGREE - INTEGER DEGREE OF POLYNOMIAL.
C ZEROR, ZEROI - OUTPUT DOUBLE PRECISION VECTORS OF
C      REAL AND IMAGINARY PARTS OF THE
C      ZEROS.
C FAIL - OUTPUT LOGICAL PARAMETER, TRUE ONLY IF
C LEADING COEFFICIENT IS ZERO OR IF RPOLY
C HAS FOUND FEWER THAN DEGREE ZEROS.
C IN THE LATTER CASE DEGREE IS RESET TO
C THE NUMBER OF ZEROS FOUND.
C TO CHANGE THE SIZE OF POLYNOMIALS WHICH CAN BE
C SOLVED, RESET THE DIMENSIONS OF THE ARRAYS IN THE
C COMMON AREA AND IN THE FOLLOWING DECLARATIONS.
C THE SUBROUTINE USES SINGLE PRECISION CALCULATIONS
C FOR SCALING, BOUNDS AND ERROR CALCULATIONS. ALL
C CALCULATIONS FOR THE ITERATIONS ARE DONE IN DOUBLE
C PRECISION.
```

6) n차 다항식에 대한 방정식은 중근에 대한 차수를 고려하여 모두 n개의 근 (허근 포함)이 존재한다.

```
COMMON /GLOBAL/ P, QP, K, QK, SVK, SR, SI, U,
     ⋮
```

이를 보면, 이 함수는 주어진 n차 다항식 $p_n(x)$에 대하여,

(i) 차수 n을 이름이 DEGREE인 int 타입의 변수에,

(ii) 그리고 $n+1$개의 계수들을 고차항의 계수부터 차례대로 ($\{a_n, a_{n-1}, \cdots, a_1, a_0\}$) 이름이 OP인 double 타입의 배열을 통하여 넘겨 받아,

n개의 근 $\alpha_i + \beta_i \mathbf{i}$ $(i = 0, 1, 2, \cdots, n-1)$을 구한 후,

(i) 실수부 (real part)에 해당하는 값들 ($\{\alpha_0, \alpha_1, \alpha_2, \cdots, \alpha_{n-1}\}$)을 이름이 ZEROR인 double 타입의 배열에 담아,

(ii) 그리고 허수부 (imaginary part)에 해당하는 값들 ($\{\beta_0, \beta_1, \beta_2, \cdots, \beta_{n-1}\}$)을 이름이 ZEROI인 double 타입의 배열에 담아 넘겨주게 된다.

(iii) 또한 이름이 FAIL인 long int 타입의 변수에 함수 수행 결과에 대한 정보를 담아 넘겨주는데, 이에 대한 설명을 발췌하면 다음과 같다.

> FAIL - OUTPUT LOGICAL PARAMETER, TRUE ONLY IF LEADING COEFFICIENT IS ZERO OR IF RPOLY HAS FOUND FEWER THAN DEGREE ZEROS. IN THE LATTER CASE DEGREE IS RESET TO THE NUMBER OF ZEROS FOUND.

다음 코드는 RPOLY 함수를 사용하여 첫 번째 주에 풀었던 4차 다항식 방정식 $p_4(x) = x^4 - 11.0x^3 + 42.35x^2 - 66.55x + 35.1384 = 0$의 근을 구하는 예를 보여주고 있다.

```
#include <stdio.h>
#include <math.h>
#define DEGREE 4
#define NCOEF 5
extern "C"
{
   int rpoly_(double *, int *, double *, double *, long int *);
}
 void main(void) {
   double poly[NCOEF] = { 1.0, -11.0, 42.35, -66.55, 35.1384 };
```

```
    int degree = DEGREE;
    double zeror[DEGREE], zeroi[DEGREE];
    long int fail;
    int i;
    rpoly_(poly, &degree, zeror, zeroi, &fail);

    if (fail) { ··· }
    for (i = 0; i < degree; i++) printf("%10f ", zeror[i]); printf("\n");
    for (i = 0; i < degree; i++) printf("%10f ", zeroi[i]); printf("\n");
}
```

4 minpack 함수를 사용한 비선형 방정식 시스템의 풀이

이제 지금까지 이해한 내용을 바탕으로 minpack에서 제공하는 함수를 사용하여 비선형 방정식 시스템의 근을 구해보자. 이를 위하여 비선형 방정식 시스템 문제를 다시 한번 생각해 보자.

다음과 같이 정의된 함수 $\mathbf{F}:\mathbb{R}^n \to \mathbb{R}^n$에 대해,

$$\mathbf{F}(x_1,x_2,\cdots,x_n) \;=\; (f_1(x_1,x_2,\cdots,x_n),\; f_2(x_1,x_2,\cdots,x_n),\cdots,\; f_n(x_1,x_2,\cdots,x_n))^t$$

$\mathbf{F}(\mathbf{x}) \;=\; \mathbf{0}$의 근을 구하라.

이 방정식 시스템의 근을 구하기 위하여 근에 대한 초기 근사값 $\mathbf{x}^{(0)} = (x_1^{(0)},x_2^{(0)},\cdots,x_n^{(0)})$으로부터 시작하여 앞에서 기술한 식을 반복적으로 계산하여 근에 수렴해가는 $\mathbf{x}^{(k)} = (x_1^{(k)},x_2^{(k)},\cdots,x_n^{(k)})$ $(k = 1,2,3,\cdots)$을 구하는 방식을 취한다고 했는데, 이 과정에서 다음과 같은 야코비 행렬이 중요한 역할을 한다고 하였다.

$$J(\mathbf{x}) \;=\; \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \frac{\partial f_1}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}) \\ \frac{\partial f_2}{\partial x_1}(\mathbf{x}) & \frac{\partial f_2}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_2}{\partial x_n}(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(\mathbf{x}) & \frac{\partial f_n}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_n}{\partial x_n}(\mathbf{x}) \end{pmatrix}$$

4.1 HYBRJ1 함수를 사용한 문제 풀이

minpack에서 제공하는 HYBRJ1 함수는 앞에서 설명한 Newton-Raphson 방법이 아니라 변형된 Powell 방법이라는 기법을 사용하여 비선형 방정식 시스템의 근을 구하는데, 이 방법 역시 근을 구하는 과정에서 야코비 행렬에 저장되어 있는 1차 미분 정보가 중요한 역할을 한다.

이 함수의 헤더 부분에는 다음과 같이 이 함수에 대한 설명이 주어져 있다.

```
      subroutine hybrj1(fcn,n,x,fvec,fjac,ldfjac,tol,info,wa,lwa)
      integer n,ldfjac,info,lwa
      double precision tol
      double precision x(n),fvec(n),fjac(ldfjac,n),wa(lwa)
      external fcn
c **********
c
c subroutine hybrj1
c
c the purpose of hybrj1 is to find a zero of a system of
c n nonlinear functions in n variables by a modification
c of the powell hybrid method. this is done by using the
c more general nonlinear equation solver hybrj. the user
c must provide a subroutine which calculates the functions
c and the jacobian.
          ⋮
```

여기서 각 인자들에 대해서 간략히 알아보자.[7)]

(a) fcn: 이 함수는 자신이 C 언어로 작성하여 제공하는 함수로서 임의의 $\mathbf{x}$에 대한 다변수 함수값 $\mathbf{F}(\mathbf{x})$와 야코비 행렬값 $J(\mathbf{x})$를 계산해주는 것을 목적으로 한다. 이 함수는 FORTRAN에서는 다음과 같이 정의가 되는데,

```
      subroutine fcn(n, x, fvec, fjac, ldfjac, iflag)
      integer n,ldfjac,iflag
      double precision x(n),fvec(n),fjac(ldfjac,n)
```

이에 대응하여 C에서는 다음과 같은 프로토타입의 함수를 작성하면 된다.

7) 상세한 내용은 실제 함수의 헤더를 자세히 읽어 볼것.

```
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);
```

여기서 iflag는 int 타입의 변수로서 그에 대한 설명을 발췌하면 다음과 같다.

a) If iflag = 1, calculate the functions at x and return this vector in fvec. Do not alter fjac.

b) If iflag = 2, calculate the jacobian at x and return this matrix in fjac. Do not alter fvec.

The value of iflag should not be changed by fcn unless the user wants to terminate execution of hybrj1. In this case, set iflag to a negative integer.

(b) n: 변수의 개수 (즉 방정식의 개수)를 나타내는 int 타입의 양의 정수.

(c) x: n개의 double 타입의 원소를 가지는 배열. 함수 호출 시에는 초기값 $\mathbf{x}^{(0)}$을 저장해 주어야 하며, 함수 리턴 시 이 함수가 구한 근 (정확히 말해서 근에 대한 추정값) $\bar{\mathbf{x}}$를 저장하고 있음.

(d) fvec: 함수 리턴 시 $\bar{\mathbf{x}}$에서의 함수값 $\mathbf{F}(\bar{\mathbf{x}})$를 저장하고 있는 n개의 double 타입의 원소를 가지는 배열.

(e) fjac: 함수 리턴 시 $\bar{\mathbf{x}}$에서의 야코비 행렬 $J(\bar{\mathbf{x}})$와 관련된 정보를 저장하고 있는 double 타입의 배열. 그 크기는 최소한 n행 n열 행렬을 저장할 수 있을 정도로 커야 함.

(f) ldfjac: 야코비 행렬 관련 정보 저장을 위한 공간인 2차원 배열 fjac의 실제 행의 개수로서 n보다 같거나 큰 int 타입의 양의 정수. C 언어와는 달리 FORTRAN에서는 열 우선 순서로 저장이 되기 때문에 실제 fjac 행렬이 double fjac[MATROWS][MATCOLS];와 같이 정의가 되었다면 MATROWS 값을 알아야 주소 계산이 가능하므로, 함수 호출 시 이 값을 ldfjac 변수를 통하여 넘겨주어야 함. 가장 간단한 방법은 double fjac[N][N]; 과 같이 n행 n열 행렬 공간을 잡은 후 n 값을 넘기는 것이다.

(g) tol: 함수 호출시 넘겨주어야 하는 0보다 같거나 큰 double 타입의 값으로서, 현재까지 구한 근 추정값 $\bar{\mathbf{x}}$와 정확한 근과의 상대 오차가 이 값보다 같거나 작다고 판단될 때에 계산을 멈춤.

(h) `info`: 함수 리턴 시 함수 수행 결과에 대한 정보를 저장하는 `int` 타입의 정수 변수로서, 그에 대한 설명을 발췌하면 다음과 같음.

> `info` is an integer output variable. If the user has terminated execution, info is set to the (negative) value of iflag (see description of fcn). Otherwise, `info` is set as follows.
>
> a) `info` = 0: Improper input parameters.
>
> b) `info` = 1: Algorithm estimates that the relative error between x and the solution is at most `tol`.
>
> c) `info` = 2: Number of calls to `fcn` with `iflag` = 1 has reached 100*(n+1).
>
> d) `info` = 3: `tol` is too small. No further improvement in the approximate solution x is possible.
>
> e) `info` = 4: Iteration is not making good progress.
>
> `wa`: 이 함수를 호출하는 함수에서 다음 인자 `lwa`가 저장하고 있는 양의 정수 개 만큼의 원소를 가지는 `double` 타입의 배열에 대한 메모리 영역을 확보한 후, 함수 호출 시 이 배열을 넘김. 이 영역은 근을 구하는 계산 시 임시 데이터를 저장하기 위한 영역으로 쓰임.
>
> `lwa`: 함수 호출 시 `wa` 배열의 길이를 지정하는 `int` 타입의 변수로서, (n*(n+13))/2보다 같거나 큰 양의 정수값을 가져야 함.

4.2 `HYBRD1` 함수를 사용한 문제 풀이

`HYBRJ1` 함수 외에 minpack에서는 비선형 방정식 시스템의 근을 구하는 함수로서 `HYBRD1`이라는 함수를 제공한다. 이 함수는 기본적으로 `HYBRJ1` 함수와 동일한 방법을 사용하나 사용자가 야코비 행렬에 대한 정보를 제공할 필요가 없이 다변수 함수값을 계산해주는 함수만 제공하면 된다는 차이가 있다. 이 함수는 내부적으로 미분 정보를 담고 있는 야코비 행렬을 수치적인 방법을 사용하여 근사적으로 추정하여 사용하게 된다. 따라서 사용자 입장에서는 야코비 행렬을 계산해주는 코드를 작성할 필요가 없어 사용하기가 더 용이하겠지만, 근사적으로 계산한 야코비 행렬을 사용하기 때문에 근에 대한 수렴 속도가 느릴 수도 있다는 점을 명심하자.

이 함수의 프로토타입은 다음과 같은데, 앞에서 설명한 내용을 상기하면 각 인자에 대한 의미를 어렵지 않게 이해할 수 있을 것이다. 반드시 그 의미를 이 함수의 헤더 부분을 읽고 확인할 것.

```
      subroutine hybrd1(fcn,n,x,fvec,tol,info,wa,lwa)
      integer n,info,lwa
      double precision tol
      double precision x(n),fvec(n),wa(lwa)
      external fcn
c **********
c
c subroutine hybrd1
c
c the purpose of hybrd1 is to find a zero of a system of
c n nonlinear functions in n variables by a modification
c of the powell hybrid method. this is done by using the
c more general nonlinear equation solver hybrd. the user
c must provide a subroutine which calculates the functions.
c the jacobian is then calculated by a forward-difference
c approximation.
          ⋮
```

이제 다음 주 실습 수업 시간까지 위의 두 FORTRAN 함수를 사용하여 다음과 같은 비선형 방정식 시스템의 근을 구하는 프로그램을 어떻게 작성할 지에 대하여 곰곰히 생각해 보자.

$$\begin{aligned} x_1 + x_2 + x_3 - 3 &= 0 \\ x_1^2 + x_2^2 + x_3^2 - 5 &= 0 \\ e^{x_1} + x_1 x_2 - x_1 x_3 - 1 &= 0 \end{aligned}$$

5 실습 문제

실습 문제 3-1

수업 시간에 다룬 FORTRAN 함수를 저장하고 있는 `gespp.f`와 `solve.f` 파일의 함수들을 사용하여 선형 방정식의 근을 구해보자.

1. 먼저 조교가 제공하는 `FortranBin.zip` 파일을 자신의 PC의 적절한 위치에 푼다.
2. 다음 역시 조교가 제공하는 `gespp.f`와 `solve.f` 파일을 `FortranBin` 디렉토리에 복사한다.
3. 다음과 같은 명령어를 사용하여 Fortran 함수들을 C/C++ 함수에서 호출할 수 있는 형태로 컴파일한다.

```
g77.exe -c gespp.f -o gespp.obj
g77.exe -c solve.f -o solve.obj
```

4. 다음 Visual Studio를 사용하여 C/C++ 스타일의 프로젝트를 생성한 후 이 Fortran 함수들을 호출할 주 함수를 저장하고 있는 C/C++ 파일을 소스파일에 추가한 후, 앞에서 컴파일한 `*.obj` 파일들을 동일한 디렉토리에 복사한다.
5. 다음 "프로젝트 → 속성 → 구성 속성 → 링커 → 입력 → 추가 종속성"에 `gespp.obj`와 `solve.obj`를 입력한다.
6. 다음 컴파일한 후 실행한다(또는 Visual Studio의 버전에 따라 비슷한 방법으로 설정).

참고

- 자신이 적절한 4행 4열 행렬 A와 4차원 벡터 x에 대해 4×4 선형 방정식 $Ax=b$를 만든 후, 이의 해를 올바르게 구하는지 확인하라.
- 원할 경우 조교가 제공하는 main.cpp 파일의 내용을 적절히 변경하여 사용하라. 특히 C와 C++의 함수 이름의 차이의 문제를 해결하기 위하여 다음과 같은 내용을 적절히 추가할 것.

```
extern "C"
{
   void gespp_(  ???  );
   void solve_(  ???  );
}
```

실습 문제 3-2

앞에서 배운 내용을 바탕으로 FORTRAN 함수를 사용하여 관련 문제를 풀어보자. 본 실습에서는 FORTRAN 코드의 컴파일을 위하여 GNU Fortran g77 컴파일러를 사용할 것인데,[8] 이에 대한 설치 및 사용 방법은 조교가 실습 시간에 설명할 예정이다.

`RPOLY` 함수를 사용하여 다음 다항식 방정식의 근을 구하라.

(i) $f_1(x) = x^5 - x^4 - 4x^3 + 4x^2 - 5x - 75 = 0$

(ii) $f_2(x) = x^3 + x^2 + 3x - 5 = 0$

(iii) $f_3(x) = 16x^4 - 40x^3 + 5x^2 + 20x + 6 = 0$

(iv) $f_4(x) = x^5 + 11x^4 - 21x^3 - 10x^2 - 21x - 5 = 0$

(v) $f_5(x) = 16x^4 + 88x^3 + 159x^2 + 76x - 240 = 0$

(vi) $f_6(x) = x^6 - 2x^5 + 14x^4 - 26x^3 + 49x^2 - 72x + 36 = 0$

여러분의 코드는 각 다항식에 대하여 이름이 `polynomial_3-1_i.txt`인 파일에 (`i`는 1부터 6까지의 정수) 다음과 같은 형식으로 저장되어 있는 다항식 정보를 읽어들여,

```
5
1.0
-1.0
-4.0
4.0
-5.0
-75.0
```

모든 근을 구한 후 그 결과를 이름이 `roots_i.txt` 파일에 보기 좋게 출력하라. 이 결과 파일에는 자신이 구한 각 근이 얼마나 정확한 것인지 확인하기 위하여, 각 근에 대한 추정치 x^*에 대하여 $|f(x^*)|$ 값 정보도 포함해주어야 한다 (복소수의 연산에 주의할 것).

8) 최신 버전인 gfortran에 대해서는 http://gcc.gnu.org/fortran/과 http://gcc.gnu.org/wiki/GFortran 참조

실습 문제 3-3

다음 비선형 방정식 시스템에 대하여 `HYBRJ1` 함수를 사용하여 근을 구하는 프로그램을 작성하라. 여러분의 프로그램은 아래의 두 초기값 각각에 대하여 적절한 정확도를 가지는 근을 이름이 `roots_3-2.txt` 파일에 보기 좋게 출력하라. 이 결과 파일에는 자신이 구한 각 근이 얼마나 정확한 것인지를 보이는 내용을 포함해야 한다.

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

초기값 1: $\mathbf{x}^{(0)} = (0.0, 0.0, 0.0)$
초기값 2: $\mathbf{x}^{(0)} = (1.55, 1.39, 1.10)$

실습 문제 3-4

다음 비선형 방정식 시스템에 대하여 `HYBRJ1` 함수를 사용하여 근을 구하는 프로그램을 작성하라. 여러분의 프로그램은 아래의 두 초기값 각각에 대하여 적절한 정확도를 가지는 근을 이름이 `roots_3-2.txt` 파일에 보기 좋게 출력하라. 이 결과 파일에는 자신이 구한 각 근이 얼마나 정확한 것인지를 보이는 내용을 포함해야 한다.

$$\begin{aligned} x_1 + x_2 + x_3 - 3 &= 0 \\ x_1^2 + x_2^2 + x_3^2 - 5 &= 0 \\ e^{x_1} + x_1 x_2 - x_1 x_3 - 1 &= 0 \end{aligned}$$

초기값 1: $\mathbf{x}^{(0)} = (0.1, 1.2, 2.5)$
초기값 2: $\mathbf{x}^{(0)} = (1.0, 0.0, 1.0)$

실습 문제 3-5

다음 비선형 방정식 시스템에 대하여 `HYBRD1` 함수를 사용하여 근을 구하는 프로그램을 작성하라. 여러분의 프로그램은 아래의 두 초기값 각각에 대하여 적절한 정확도를 가지는 근을 이름이 `roots_3-3.txt` 파일에 보기 좋게 출력하라. 이 결과 파일에는 자신이 구한 각 근이 얼마나 정확한 것인지를 보이는 내용을 포함해야 한다.

$$\begin{aligned} 10x_1 - 2x_2^2 + x_2 - 2x_3 - 5 &= 0 \\ 8x_2^2 + 4x_3^2 - 9 &= 0 \\ 8x_2x_3 + 4 &= 0 \end{aligned}$$

초기값 1: $\mathbf{x}^{(0)} = (1.0, -1.0, 1.0)$
초기값 2: $\mathbf{x}^{(0)} = (1.0, 1.0, -1.0)$

실습 문제 3-6

다음 비선형 방정식 시스템에 대하여 `HYBRJ1` 함수를 사용하여 $(x,y) \in [-4,4] \times [-5,5]$인 모든 근을 구하는 프로그램을 작성하라. 자신이 구한 모든 근을 이름이 `roots_3-4.txt` 파일에 보기 좋게 출력하라. 이 결과 파일에는 자신이 구한 각 근이 얼마나 정확한 것인지를 보이는 내용을 포함해야 한다.

$$\begin{aligned} 3x^2 - 2y^2 - 1 &= 0 \\ x^2 - 2x + y^2 + 2y - 8 &= 0 \end{aligned}$$

실습 문제 3-7

다음 비선형 방정식 시스템에 대하여 `HYBRD1` 함수를 사용하여 $(x,y) \in [-4,4] \times [-5,5]$인 모든 근을 구하는 프로그램을 작성하라. 자신이 구한 모든 근을 이름이 `roots_3-5.txt` 파일에 보기 좋게 출력하라. 이 결과 파일에는 자신이 구한 각 근이 얼마나 정확한 것인지를 보이는 내용을 포함해야 한다.

$$\begin{aligned} 2x^3 - 12x - y - 1 &= 0 \\ 3y^2 - 6y - x - 3 &= 0 \end{aligned}$$

실습 문제 3-8

다음과 같은 비선형 방정식 시스템을 고려하자.

$$\mathbf{F}(x_1,x_2,x_3) = (f_1(x_1,x_2,x_3),\ f_2(x_1,x_2,x_3),\ f_3(x_1,x_2,x_3))^t = \mathbf{0},$$

$$\begin{aligned} f_1(x_1,x_2,x_3) &= 3x_1 - \cos(x_2x_3) - \frac{1}{2} = 0 \\ f_2(x_1,x_2,x_3) &= x_1^2 - 81(x_2+0.1)^2 + \sin x_3 + 1.06 = 0 \\ f_3(x_1,x_2,x_3) &= e^{-x_1x_2} + 20x_3 + \frac{10\pi - 3}{3} = 0 \end{aligned}$$

이 비선형 함수 $\mathbf{F}(x_1,x_2,x_3)$에 대한 야코비 행렬은 다음과 같다.

$$J(x_1,x_2,x_3) = \begin{pmatrix} 3 & x_3\sin(x_2x_3) & x_2\sin(x_2x_3) \\ 2x_1 & -162(x_2+0.1) & \cos x_3 \\ -x_2e^{-x_1x_2} & -x_1e^{-x_1x_2} & 20 \end{pmatrix}$$

이제 `HYBRJ1` 함수를 사용하여 이 비선형 방정식 시스템의 근을 구하려 한다. 자신의 프로그램을 작성한 후 다음과 같은 초기값에 대하여

$$\mathbf{x}^{(0)} = (0.1, 0.1, -0.1)$$

자신이 구한 결과가 다음과 같은 근으로 수렴을 하는지 확인하라.

$$\mathbf{x}^* = (0.5, 0.0, -0.52359877)$$

위의 초기값에 대한 수행 결과를 이름이 `roots_3-8.txt` 파일에 이해하기 쉽게 출력하라. 이 결과 파일에는 자신이 구한 근이 얼마나 정확한 것인지를 보이는 정보를 포함해야 한다.

6 숙제

제출 마감: ?월 ?일 오후 ?시 정각
제출물 및 방법: 조교가 실습 시간에 공지

이번 주의 숙제는 다음과 같다.

숙제 3-1

(i) minpack에서 제공하는 FORTRAN 함수인 HYBRJ1을 사용하여 다음과 같이 정의되는 비선형 방정식 시스템을 풀어 GPS 수신기 위치를 찾아주는 프로그램 (프로그램 3-1)을 작성하라.

$$
\begin{aligned}
f_1(x_1,x_2,x_3,x_4) &= (x_1-p_{11})^2+(x_2-p_{12})^2+(x_3-p_{13})^2-\{C(tr_1+x_4-t_1)\}^2 = 0\\
f_2(x_1,x_2,x_3,x_4) &= (x_1-p_{21})^2+(x_2-p_{22})^2+(x_3-p_{23})^2-\{C(tr_2+x_4-t_2)\}^2 = 0\\
f_3(x_1,x_2,x_3,x_4) &= (x_1-p_{31})^2+(x_2-p_{32})^2+(x_3-p_{33})^2-\{C(tr_3+x_4-t_3)\}^2 = 0\\
f_4(x_1,x_2,x_3,x_4) &= (x_1-p_{41})^2+(x_2-p_{42})^2+(x_3-p_{43})^2-\{C(tr_4+x_4-t_4)\}^2 = 0
\end{aligned}
$$

여러분의 프로그램은 이름이 GPS_signal_i.txt인 파일에 (i는 0보다 같거나 큰 정수) 다음과 같은 형식으로 저장되어 있는 다항식 정보를 읽어들인 후,

```
C b
p11 p12 p13 t1 tr1
p21 p22 p23 t2 tr2
p31 p32 p33 t3 tr3
p41 p42 p43 t4 tr4
```

콘솔 윈도우에서 다음과 같은 초기값 정보를 읽어들여,

```
x10 x20 x30 x40
```

GPS 수신기의 위치를 HYBRJ1 함수를 사용하여 구한 후 그 결과를 이름이 GPS_position_3-1_i.txt 파일에 보기 좋게 출력하라. 이 결과 파일에는 자신이 구한 위치가 실제 정확한 위치인지를 보이는 내용을 포함해야 한다.

(ii) minpack에서 제공하는 FORTRAN 함수인 HYBRD1을 사용하여 위 문제를 반복하라. 결과는 이름이 GPS_position_3-2_i.txt인 파일에 저장하라.

(iii) 위 두 방법을 사용하여 문제를 풀면서 알게 된 내용을 보고서 형식의 파일에 기술하여 제출하라.

숙제 3-2

(i) minpack에서 제공하는 적절한 FORTRAN 함수를 사용하여 다음과 같이 정의되는 비선형 방정식 시스템의 근을 찾아주는 프로그램 (프로그램 3-2)를 작성하라.

$$\begin{aligned} x+10y &= -9 \\ \sqrt{5}(z-w) &= 2\sqrt{5} \\ (y-2z)^2 &= 9 \\ \sqrt{10}(x-w)^2 &= 2\sqrt{10} \end{aligned}$$

초기값: $(0.9, -0.9, 1.25, -1.25)$

(ii) 자신이 구한 결과는 이름이 `roots_found_3-2.txt`인 파일에 이해하기 쉬운 포맷으로 저장하라. 이 결과 파일에는 자신이 구한 각 근이 얼마나 정확한 것인지를 보이는 내용을 포함해야 한다.

숙제 3-3

(i) minpack에서 제공하는 적절한 FORTRAN 함수를 사용하여 다음과 같이 정의되는 비선형 방정식 시스템의 근을 찾아주는 프로그램 (프로그램 3-3)을 작성하라.

$$\begin{aligned} \frac{\sin(xy+\frac{\pi}{6})+\sqrt{x^2y^2+1}}{\cos(x-y)} &= -2.8 \\ \frac{xe^{xy+\frac{\pi}{6}}-\sin(x-y)}{\sqrt{x^2y^2+1}} &= 1.66 \end{aligned}$$

초기값: $(20.0, 0.0)$

(ii) 자신이 구한 결과는 이름이 `roots_found_3-3.txt`인 파일에 이해하기 쉬운 포맷으로 저장하라. 이 결과 파일에는 자신이 구한 각 근이 얼마나 정확한 것인지를 보이는 내용을 포함해야 한다.

숙제 3-4

수업 시간에 다룬 FORTRAN 함수 `GESPP()`와 `SOLVE`를 사용하여 선형 방정식의 근을 구하는 프로그램 (프로그램 3-4)을 작성하라.

(i) 다음과 같은 선형 방정식 $Ax = b$에 대하여,

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

이름이 `linear_system_3-4.txt`에는 다음과 같은 형식으로 선형 방정식이 저장되어 있다. (주의: 여러분의 프로그램은 n은 최대 32까지 처리할 수 있도록 작성하라)

a) 이 파일의 첫 줄에는 n값이 정수 형태로 주어져 있다.

b) 다음 n^2줄에는 A 행렬의 원소들이 행 우선 순위(row-major order)로 부동 소수점 형태로 주어져 있다.

c) 다음 n줄에는 b 벡터의 원소들이 부동 소수점 형태로 주어져 있다.

(ii) 이제 이 입력 파일을 읽어들여 해당 선형 방정식의 근 $\hat{x}$를 구한 후, 그 결과와 이 근의 오차에 대한 척도를 이름이 `solution_3-4.txt`에 다음과 같은 형식으로 저장하라.

a) 이 파일의 첫 줄에 n 값을 정수 형태로 저장하라.

b) 다음 n줄에는 $\hat{x}$ 벡터의 원소들을 부동 소수점 형태로 저장하라(이때 각 수자는 유효수자가 6자리가 되도록 출력할 것).

c) 다음 마지막 줄에는 오차에 대한 척도를 부동 소수점 형태로 저장하라(이 수자는 유효수자가 6자리가 되도록 출력할 것). 여기서 오차에 대한 척도는 다음과 같이 정의된다.

$$\frac{||A\hat{x} - b||}{||b||} \text{ where } ||a|| = \sqrt{\sum_{i=0}^{n} a_i^2} \text{ for } a = (a_1, a_2, \cdots, a_n)^t$$

(iii) 조교가 제공하는 선형 방정식들에 대하여 실험을 해본 후, 오차에 대한 척도를 통하여 자신이 얼마나 정확한 근을 구했는지에 대하여 보고서 형식의 파일에 분석하여 제출하라.