고급 SW 실습 I
Nonlinear Equation Root Finding
(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 snnnnnnL04.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 snnnnnnL04.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.
  - 다음 실습 시간에 담당 조교의 양해하에 잠깐 시간을 내어 이전 주 실습 결과를 검사 받을 수 있다(감점이 있을 것임).
- 실습 프로그램을 제출했는데 검사를 받지 않은 경우
  - 반드시 검사를 받아야 한다.
  - 그렇지 않으면 제출물을 무효화 할 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.

실습 프로그램 구성 및 입출력

◆ 실습 프로그램

- ✦ 두 개의 파일을 작성한다.
  - ◆ function.h
    - Include files, 필요한 정의 등을 포함한다.
    - 문제 별로 수식 $f(x)$, $f'(x)$ 계산 함수, 그리고 해당 문제와 관련해서 출력에 필요한 문자열을 정의한다.
  - ◆ snnnnnnL04.cpp
    - 강의에서 언급한 세가지 방법에 대한 함수, 그리고 main 외 몇 가지 함수들을 작성한다.
- ✦ 사전 게시한 파일들을 다운받아 이름을 바꾸어 사용한다[(1)]
  - ◆ function_std.h → function.h
  - ◆ snnnnnnL04_std.cpp → snnnnnnL04.cpp
  - ◆ snnnnnnL04_IN_std.txt → snnnnnnL04_IN.txt

(1) nnnnnn은 자신의 학번 뒤 6자리

◆ 데이터 입출력

- stdin과 stdout을 사용하여 프로그램을 작성하고, redirection을 이용하여 데이터를 입력 받고, 결과를 출력한다.
- 예를 들어 입력 데이터가 in.txt에 저장되어 있고, 실행 결과를 out.txt에 저장하기를 원한다면, 명령 프롬프트에서 다음과 같이 입력한다

```
snnnnnnL04 < in.txt > out.txt <ent>
```

여기서, 실행 파일은 snnnnnnL04.exe이고, 세 파일 모두 같은 폴더에 있다고 가정한다.

◆ 입력 데이터 형식

✦ 아래 예를 통하여 설명한다

```
1 ← test case 개수
1   1   2   4.0   1.0      (L1)
3.0 0   0.0                (L2)
50  1.0e-6  1.0e-5  1      (L3)
```

test case 하나에 대한 입력 세트 (L1–L3)

a b c (없으면 안 적음)

L1 `1   1   2   4.0   1.0`

- 4.0 1.0 → 수식에 필요한 parameters
- 2 → parameter 개수(3 개까지(조정 가능))
- 1 (두 번째) → 풀이 방법 : 1 (NEWTON_R)
- 1 (첫 번째) → 문제 번호 : 1 (PROBLEM1)

문제 1의 수식

$f(x) = x^2 - ax + 4 - \ln(bx) = 0$에서 $a = 4.0$, $b = 1.0$ (불필요한 경우 parameter 개수를 0으로 설정).

✦입력 예(계속)

```
1   ← test case 개수
L1  1   1   2   4.0   1.0
L2  3.0   0   0.0
L3  50   1.0e-6   1.0e-5   1
```

test case 하나에 대한 입력 세트

```
L2  3.0   0   0.0
```

초기값 x_0

두번째 초기값 x_1이 필요한 경우 추가(secant 또는 bisection (a,b))

이미 알고 있는 근의 값

좌측 COVERGENCE_CHECK가 0일 경우 0.0

이 값을 1(CONVERGENCE_CHECK)로 설정하면, 근에 근접했을 때 강의자료 14쪽 수식에서 c 값이 대략 일정한 값인지 조사할 수 있다. 이를 위해서는 반드시 이미 알고 있는 근의 값을 추가해야 한다.

✦입력 예(계속)

이미 알고 있는 근의 값
좌측 COVERGENCE_CHECK가
경우 0.0

```
1  ← test case 개수
L1  1   1   2   4.0   1.0
L2  3.0   0   0.0
L3  50   1.0e-6   1.0e-5   1
```

test case 하나에 대한 입력 세트

```
L3  50   1.0e-6   1.0e-5   1
```

- 50 → 최대 반복 회수 $N_{\max}$
- 1.0e-6 → 종료 조건 $|f(x_{n+1})| < \delta$의 $\boldsymbol{\delta}$ 값
- 1.0e-5 → 종료 조건 $|x_{n+1} - x_n| < \epsilon$의 $\boldsymbol{\epsilon}$ 값
- 1 → 이 값을 1(OUTPUT_STEPS)로 설정하면, 매 iteration마다 중간 실행 결과를 출력한다. Convergence rate 등을 조사하려면 이를 1로 세팅하자. 0으로 설정하면 최종 결과만 출력한다.

실습 1 프로그램 실행해 보기

- 실행 맛보기
  - 다운 받은 두 파일을 VS 프로젝트에 등록하고 빌드하자.
  - 이미 Newton-Raphson 함수가 작성되어 있으며, 이를 호출하기 위하여 필요한 모든 것이 완성되어 있다.
  - 앞의 예에 대한 결과를 조사하자(입력: in.txt, 출력: out.txt).

```
Root finding for f1(x) = x^2 -ax +4 -ln(bx) = 0 by the Newton-Raphson..
a = 4.000000e+00 b = 1.000000e+00
 n              xn1              |f(xn1)|
 0  3.0000000000000000e+00  9.86e-02
 1  3.0591673732008657e+00  3.69e-03
 2  3.0571060546915998e+00  4.48e-06
 3  3.0571035499843625e+00  6.61e-12
ans: x= 3.0571035499843625e+00 |f(x)|= 6.6098237994e-12 iteration=  3
```

  - 이제 프로그램이 어떻게 구성되어 있고, 어떤 순서로 실행되는지 디버거를 이용하여 꼼꼼히 살펴보자[(1)].

(1) 조교가 실습 검사할 때 프로그램 구성에 대해 질문할 수도 있다.

✦아래와 같이 같은 입력을 두 번 반복 입력해서 실행해 보자.

```
2
1 1 2 4.0 1.0
3.0 0 0.0
50 1.0e-6 1.0e-5 1

1 1 2 4.0 1.0
3.0 0 0.0
50 1.0e-6 1.0e-5 1
```

(입력 A)

```
2
1 1 2 4.0 1.0
3.0 0 0.0
50 1.0e-6 1.0e-5 0

1 1 2 4.0 1.0
3.0 0 0.0
50 1.0e-6 1.0e-5 0
```

(입력 B)

◆입력 A와 입력 B에 대해 실행한 결과의 차이를 알아보고, 프로그램 구성을 다시 한번 익히도록 한다.

- 아래와 같이 두 번째 테스트의 δ와 ϵ 값을 아주 작게 설정하여 실행한 결과를 조사해보자.

```
2
1 1 2 4.0 1.0
3.0 0 0.0
50 1.0e-6 1.0e-5 1

1 1 2 4.0 1.0
3.0 0 0.0
50 1.0e-20 1.0e-20 1
```

- $f(x_n)$ 값이 보다 0에 접근한 값을 얻을 수 있을 것이다.

- 앞에서 정밀하게 구한 값을 이 식의 근이라고 가정하고 다음과 같은 입력을 만들어 실행해 보자.

```
2
1 1 2 4.0 1.0
3.0 1 3.0571035499994738323
50 1.0e-6 1.0e-5 1
```

- 다음과 같은 출력이 나올 것이다.

```
Root finding for f1(x) = x^2 -ax +4 -ln(bx) = 0 by the Newton-Raphson..
a = 4.000000e+00 b = 1.000000e+00
 n               xn1               |f(xn1)|     e1         c=e1/(e0*e0)
 0  3.0000000000000000e+00  9.86e-02
 1  3.0591673732008657e+00  3.69e-03 2.063823e-03 6.329163e-01
 2  3.0571060546915997e+00  4.48e-06 2.504697e-06 5.880445e-01
 3  3.0571035499843625e+00  6.61e-12 3.697931e-12 5.894520e-01
ans: x= 3.0571035499843625e+00 |f(x)|= 6.6098237994e-12 iteration=  3
```

- 위 결과를 볼 때 이 수식의 경우 대략 $e_n \approx 5.9 e_{n-1}^2$ 정도의 rate로 근에 수렴함을 짐작할 수 있다.

프로그램 설명

- function.h
  - 이 파일의 내용을 간략히 살펴보자.
  - Define 문 및 변수

```
#define       OUTPUT_STEPS 1  // 입력 형식 설명 참조
#define CONVERGENCE_CHECK 1  // 입력 형식 설명 참조

#define PARAMETER_N       3       // 최대 3 개까지
char pidx[PARAMETER_N+1] = "abc"; // 변경 가능

#define NEWTON_R    1
#define SECANT      2
#define BISECTION   3
// (*)풀이 방법 추가 시 여기에 이의 심볼 및 id를 추가해야 한다

const char *NRstr = "Newton-Raphson method";
const char *SCstr = "Secant method";
const char *BSstr = "Bisection method";
// (*)풀이 방법 추가 시 이를 설명하는 문자열을 추가해야 한다
```

✦ 문제 정의

```
// 함수 포인터
double(*_F)(double, double, double, double);
double(*_FP)(double, double, double, double);

//문제 1 정의 : f(x) = x^2 -4x +4 -lnx = 0
#define PROBLEM1 1
const char *LabF1str = "f(x) = x^2 -ax +4 -ln(bx) = 0";
Double _LabF1(double x, double a, double b, double c) {
  return x*x - a*x + 4 - log(b*x);     // f(x) 값 계산
}
Double _LabFP1(double x, double a, double b, double c) {
  return 2*x - a - (1/x);     // f(x)의 1차 미분 값 계산
}

// 문제 추가 시 위와 동일한 형태의 코드를 빠짐 없이 작성해야 한다.
// 문제 번호는 기존 번호와 다르게 정한다(위 붉은 색)
```

◆ snnnnnnL04.cpp

✦ Newton-Raphson method 함수

```
void NewtonRaphson (
  double x1, double p[], int maxIter,
  double delta, double epsilon, int showSteps,
  int chkConvergence, double root,
  double *ans, double *abs_error, int *iteration
);
```

- Parameter는 앞에서 입력 정의할 때를 기억하면 모두 그 의미를 이해할 수 있을 것이다.
- Secant, bisection 함수 모두 초기 값이 한 개 더 추가되는 것을 제외하고는 Newton-Raphson 함수와 동일하다.
- 새로운 방법을 추가할 경우에도 함수 parameter는 동일할 것이다.
- 함수 내 코드는 스스로 이해할 수 있을 것이다.

- 함수 **functionSelection( )**
  - 이 함수는 입력으로 받은 해결해야 할 문제의 id를 입력 받아 이에 대응하는 $f(x)$와 $f'(x)$를 계산하는 함수 포인터를 **_F**, **_FP**로 설정하는 함수이다.
  - 문제 1에 대한 예를 프로그램에 보였으니, 새로운 문제를 위한 코드를 어렵지 않게 추가할 수 있을 것이다.
- 함수 **finalOutput( )**
  - 최종 해를 출력하는 함수이다.
  - 이 역시 문제 1에 대한 예를 프로그램에 보였으니, 새로운 문제에 대해서도 쉽게 코드를 추가할 수 있을 것이다.
  - 경우에 따라서 근 외에 추가로 계산해서 출력할 값이 있을 수 있는데, 이를 대비하여 수식 정의용 parameter 배열도 입력으로 전달된다.

- 함수 **showMethod_Parameters( )**
  - 문제에 적용할 풀이 방법과 혹시 있다면 수식에 사용한 parameter들을 출력하는 함수이다.
- Main 함수
  - 여기서는 입력을 읽어 어떤 문제인지, 어떤 풀이 방법을 사용할지 등을 결정하여 해당 함수를 호출하여 근을 구하고 이를 출력한다.
  - 프로그램에 충분한 주석을 포함하였으므로 스스로 이해할 수 있을 것이다.

실습 2 Secant, Bisection 함수 작성

- Secant, Bisection 함수 작성
  - 이제 이 두 함수를 작성하자.
  - 프로그램에 작성할 부분을 명시하였고, Newton-Raphson 함수도 제공하였으므로 문제 없이 작성할 수 있을 것이다.
  - function.h는 이미 세 방법에 대한 설정을 완료한 프로그램이므로 수정할 필요 없다.
  - 다음 쪽에 보이는 입력에 대해 오류 없이 그리고 출력도 동일하게 실행 될 수 있도록 필요한 곳을 모두 추가 또는 작성하여야 한다.

- 프로그램 작성을 완료하면 다음 입력에 대해 실행해 보자

Test_02_01_in.txt

```
3
1 1 2 4.0 1.0
3.0 0 0.0
50 1.0e-6 1.0e-5 1

1 2 2 4.0 1.0
2.0 4.0 0 0
50 1.0e-6 1.0e-5 1

1 3 2 4.0 1.0
2.0 4.0 0 0
50 1.0e-6 1.0e-5 1
```

Newton-Raphson

Secant method

Bisection method

- 다음 쪽에 이 입력에 대한 결과를 보이는데 이와 동일하게 출력되어야 한다.

앞 쪽의 테스트에 대한 출력(1/2)

- 첫번째 테스트는 앞에서 실습한 것이므로 생략한다.
- 두번째 secant 방법에 의한 결과

```
Root finding for f1(x) = x^2 -ax +4 -ln(bx) = 0 by the Secant method
a = 4.000000e+00 b = 1.000000e+00
 n              xn1             |f(xn1)|
 0  4.0000000000000000e+00  2.61e+00
 1  2.4192186458890025 95e+00  7.08e-01
 2  2.7560397122060127 37e+00  4.42e-01
 3  3.3170225040011245 28e+00  5.35e-01
 4  3.0097689593710317 27e+00  8.22e-02
 5  3.0506707097257623 75e+00  1.15e-02
 6  3.0572890416599198 82e+00  3.32e-04
 7  3.0571028439288747 69e+00  1.26e-06
 8  3.0571035499175396 31e+00  1.38e-10
ans: x= 3.0571035499175396 31e+00 |f(x)|= 1.3796119802e-10 iteration=  8
```

- Newton-Raphson method에 비해 더 많이 반복하는데, 정확도도 떨어져 iteration을 좀더 실행해야 할 필요가 있다.

✦ 앞 쪽의 테스트에 대한 출력(2/2)

◆ 두번째 bisection 방법에 의한 결과

```
Root finding for f1(x) = x^2 -ax +4 -ln(bx) = 0 by the Bisection method
a = 4.000000e+00 b = 1.000000e+00
 n              xn1              |f(xn1)|
 0  3.00000000000000000e+00  9.86e-02
 1  3.50000000000000000e+00  9.97e-01
 2  3.25000000000000000e+00  3.84e-01
 3  3.12500000000000000e+00  1.26e-01
 4  3.06250000000000000e+00  9.67e-03
 5  3.03125000000000000e+00  4.55e-02
 6  3.04687500000000000e+00  1.82e-02
 7  3.05468750000000000e+00  4.31e-03
 8  3.05859375000000000e+00  2.67e-03
 9  3.05664062500000000e+00  8.27e-04
10  3.05761718750000000e+00  9.18e-04
11  3.05712890625000000e+00  4.53e-05
12  3.05688476562500000e+00  3.91e-04
13  3.05700683593750000e+00  1.73e-04
14  3.05706787109375000e+00  6.38e-05
15  3.05709838867187500e+00  9.22e-06
16  3.05711364746093750e+00  1.80e-05
17  3.05710601806640625e+00  4.41e-06
ans: x= 3.05710601806640625e+00 |f(x)|= 4.4106975188e-06 iteration= 17
```

◆ Convergence Test

✦ 아래 데이터에 대해 프로그램을 실행해 보자

Test_02_02_in.txt

```
3
1 1 2 4.0 1.0
3.0 1 3.057103549994738323
50 1.0e-6 1.0e-5 1

1 2 2 4.0 1.0
2.0 4.0 1 3.057103549994738323
50 1.0e-6 1.0e-5 1

1 3 2 4.0 1.0
2.0 4.0 1 3.057103549994738323
50 1.0e-6 1.0e-5 1
```

✦ Test_02_02_in.txt에 대한 출력(1/2)

◆ Secant method

```
Root finding for f1(x) = x^2 -ax +4 -ln(bx) = 0 by the Secant method
a = 4.000000e+00 b = 1.000000e+00
 n              xn1                |f(xn1)|     e1          c=e1/e0^1.62
 0  4.0000000000000000e+00  2.61e+00
 1  2.4192186458890025 95e+00  7.08e-01 6.378849e-01 7.016340e-01
 2  2.7560397122060127 37e+00  4.42e-01 3.010638e-01 6.237004e-01
 3  3.3170225040011245 28e+00  5.35e-01 2.599190e-01 1.817235e+00
 4  3.0097689593710317 27e+00  8.22e-02 4.733459e-02 4.198954e-01
 5  3.0506707097257623 75e+00  1.15e-02 6.432840e-03 9.007689e-01
 6  3.0572890416599198 82e+00  3.32e-04 1.854917e-04 6.587368e-01
 7  3.0571028439288747 69e+00  1.26e-06 7.060659e-07 7.837178e-01
 8  3.0571035499175396 31e+00  1.38e-10 7.719869e-11 7.120011e-01
ans: x= 3.0571035499175396 31e+00 |f(x)|= 1.3796119802e-10 iteration=  8
```

✦ Test_02_02_in.txt에 대한 출력(2/2)

◆ Bisection method

```
Root finding for f1(x) = x^2 -ax +4 -ln(bx) = 0 by the Bisection method
a = 4.000000e+00 b = 1.000000e+00
 n              xn1               |f(xn1)|      e1          c=e1/e0
 0  3.000000000000000000e+00  9.86e-02
 1  3.500000000000000000e+00  9.97e-01 4.428965e-01 7.756023e+00
 2  3.250000000000000000e+00  3.84e-01 1.928965e-01 4.355340e-01
 3  3.125000000000000000e+00  1.26e-01 6.789645e-02 3.519839e-01
 4  3.062500000000000000e+00  9.67e-03 5.396450e-03 7.948059e-02
 5  3.031250000000000000e+00  4.55e-02 2.585355e-02 4.790844e+00
 6  3.046875000000000000e+00  1.82e-02 1.022855e-02 3.956343e-01
 7  3.054687500000000000e+00  4.31e-03 2.416050e-03 2.362065e-01
 8  3.058593750000000000e+00  2.67e-03 1.490200e-03 6.167919e-01
 9  3.056640625000000000e+00  8.27e-04 4.629250e-04 3.106462e-01
10  3.057617187500000000e+00  9.18e-04 5.136375e-04 1.109548e+00
11  3.057128906250000000e+00  4.53e-05 2.535626e-05 4.936605e-02
12  3.056884765625000000e+00  3.91e-04 2.187844e-04 8.628418e+00
13  3.057006835937500000e+00  1.73e-04 9.671406e-05 4.420519e-01
14  3.057067871093750000e+00  6.38e-05 3.567890e-05 3.689112e-01
15  3.057098388671875000e+00  9.22e-06 5.161323e-06 1.446604e-01
16  3.057113647460937500e+00  1.80e-05 1.009747e-05 1.956372e+00
17  3.057106018066406250e+00  4.41e-06 2.468072e-06 2.444249e-01
ans: x= 3.057106018066406250e+00 |f(x)|= 4.4106975188e-06 iteration= 17
```

일부를 제외하고 c값이 대략 0.5 보다 작음을 보이고 있다.

실습 3 $\sqrt{2}$ 구하기

- $\sqrt{2}$ 구하기
  - 다음 식에서 양의 근을 구하면 된다.

$$x^2 - 2 = 0 \qquad (3\text{-}1)$$

  - function.h에 이 문제를 PROBLEM2로 정하여 추가하고, snnnnnnL04.cpp 파일도 적절히 수정한다.
  - $\sqrt{2} = 1.41421356237309504880$를 정답으로 간주하자.
  - 수식 3-1의 근을 세가지 방법으로 구하되, convergence rate를 볼 수 있도록 입력 파일을 적절히 설정한다.
  - 또한, 구한 해의 오차 e_n이 order of 10^{-12} 이하가 되도록 각 풀이 방법의 $N_{\{max\}}, \delta, \epsilon$ 등을 조정하자.
  - 위의 결과 즉, 조건에 맞는 해를 구하면, convergence rate 예측한 후 조교의 검사를 받는다.

실습 4 Three-leaved Clover 곡선

- Three-leaved Clover 곡선
  - 전년도 실습 자료의 실습 문제 1-5를 해결하자.
  - function.h에 이 문제를 PROBLEM2로 정하여 추가하고, snnnnnnL04.cpp 파일도 적절히 수정한다.
  - $a = 1.0$, $b = -0.25$에 대하여 세가지 방법으로 해를 구하여 조교의 검사를 받는다.
  - 만일 해가 한 개 이상이라면 가능한 모든 해를 구해보자
  - 출력에는 x 뿐만 아니라 y 값도 포함되어야 한다.