

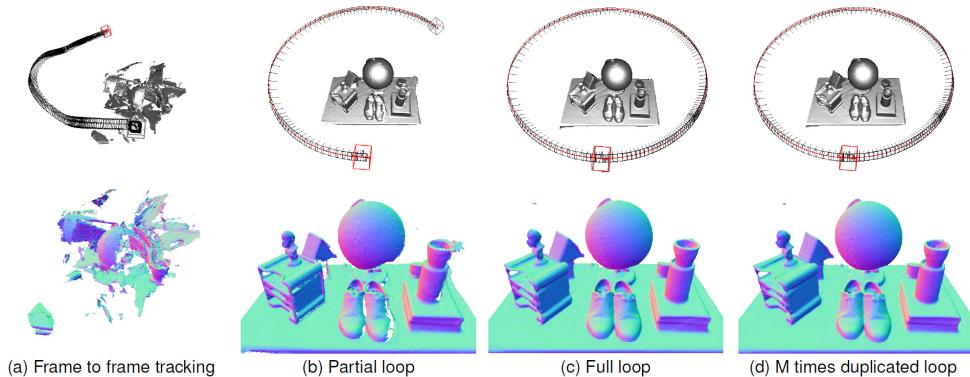
[CSE 4152] 고급 소프트웨어 실습 I

『GPS 수신기 위치 계산 문제』

실험 4: 비선형 방정식의 풀이 기초

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

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



KinectFusion: Real-Time Dense Surface Mapping and Tracking by R. Newcombe et al. (2011)

To implement the presented camera tracking method, you need to solve a nonlinear minimization problem, which is a variation of Iterative Closest Point (ICP).

$$\mathbf{E}(\mathbf{T}_{g,k}) = \sum_{\substack{\mathbf{u} \in \mathcal{U} \\ \Omega_k(\mathbf{u}) \neq \text{null}}} \left\| \left(\mathbf{T}_{g,k} \hat{\mathbf{V}}_k(\mathbf{u}) - \hat{\mathbf{V}}_{k-1}^g(\hat{\mathbf{u}}) \right)^T \hat{\mathbf{N}}_{k-1}^g(\hat{\mathbf{u}}) \right\|_2$$

지금까지 배운 컴퓨터공학 주제들은 주로 이산공간(discrete space)에서 정의된 문제들이다. 반면 현실 세계에서 부딪치는 대부분의 문제들은 연속공간(continuous space)에서 정의가 되며, 이를 효과적으로 해결하기 위해서는 다양한 형태의 수치 알고리즘(numerical algorithm)의 이해와 활용 능력이 필요하다. 또한 연속공간의 기본 수체계인 실수(real number)를 컴퓨터는 부동 소수점 숫자(floating-point number)를 통하여 근사적으로 표현하여 연산을 수행한다. 따라서, 우리가 머리속으로 생각하는 수학 계산과 실제로 컴퓨터가 수행하는 연산에는 상당한 괴리가 발생할 수 있다는 사실에 대처할 능력을 필요로 한다. 또한 종종 수치 계산은 방대한 양의 계산을 필요로 하기 때문에 GPU 컴퓨팅을 통하여 계산속도를 높일 필요가 있다.

1 GPS 수신기 위치 계산 문제

GPS 수신기는 세 개 이상의 GPS 위성으로부터 송신된 신호를 수신하여 위성과 수신기의 위치를 결정한다(그림 1 참조). 위성에서 송신된 신호와 수신기에서 수신된 신호의 시간차를 측정하면 위성과 수신기 사이의 거리를 구할 수 있는데, 이때 송신된 신호에는 위성의 위치에 대한 정보가 들어 있다. 최소한 세 개의 위성과의 거리와 각 위성의 위치를 알게 되면 수신기의 위치를 계산할 수 있다. 그러나 시계가 완전히 정확하지 않기 때문에 오차를 보정하고자 보통 네 개 이상의 위성을 이용해 위치를 결정한다.

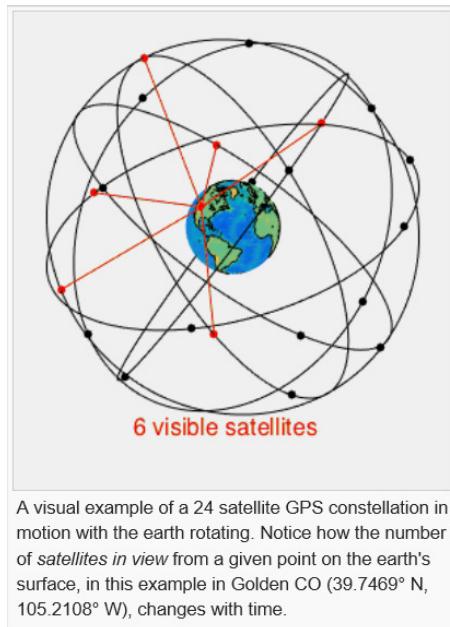


그림 1: 24개의 위성을 활용한 Global Positioning System의 예시도 (출처: https://en.wikipedia.org/wiki/Global_Positioning_System)

GPS 수신기는 위성으로부터 수신한 신호로부터 위성의 위치와 신호를 보낸 시간을 결정할 수 있다. i 번째 위성의 위치 좌표와 신호 송신 시간을 각각 $p_i = (x_i, y_i, z_i)$ 과 t_i 라고 하자 ($i = 1, 2, 3, 4$). 만약 수신기가 신호를 수신한 시간이 tr_i 이라면, 수신기와 i 번째 위성의 거리는 $r_i = C(tr_i - t_i)$ 가 된다 (여기서 C 는 빛의 속도). 따라서 수신 신호를 통하여 i 번째 위성의 위치 좌표와 수신기와 위성간의 거리를 알 수 있는데, 이를 다르게 표현하면 GPS 수신기는 p_i 를 중심으로 하고 반지름이 r_i 인 구의 표면상에 존재한다고 할 수 있다.

수학적으로 표현하면, i 번째 위성을 기준으로 GPS 수신기의 위치 좌표 (x, y, z) 는 다음과 같은 조건을 만족해야 한다.

$$(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2 = \{C(tr_i - t_i)\}^2$$

인공위성에서 수신한 시간 정보에 오차가 없다면 네 개의 위성에 대한 구 표면을 교차시켜 교차점을 찾음으로써 GPS 수신기의 위치 좌표를 구할 수 있다. 일반적으로 (실제로 GPS 수신기 위치 찾은 문제에서) 두 구의 표면을 교차시키면, 원에 해당하는 곡선이 나오고 이를 세 번째 구와 교차시키면 (일반적으로) 두 점이 생성되며, 마지막으로 네 번째 구와 교차시켜 정확한 GPS 수신기의 위치를 구할 수 있다.

실제로는 우리가 사용하는 범용 GPS 수신기의 시간이 정확하지 않아 위성이 보내주는 정확한 시간간에는 b 만큼의 차이가 있다. 따라서 이러한 편차 b 를 고려하여 다음과 같은 식을 사용하여 GPS 수신기의 위치를 찾게 된다.

$$\begin{aligned}(x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2 &= \{C(tr_1 + b - t_1)\}^2 \\(x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2 &= \{C(tr_2 + b - t_2)\}^2 \\(x - x_3)^2 + (y - y_3)^2 + (z - z_3)^2 &= \{C(tr_3 + b - t_3)\}^2 \\(x - x_4)^2 + (y - y_4)^2 + (z - z_4)^2 &= \{C(tr_4 + b - t_4)\}^2\end{aligned}$$

이는 다시 다음과 같아

$$\begin{aligned}f_1(x, y, z, b) &= (x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2 - \{C(tr_1 + b - t_1)\}^2 = 0 \\f_2(x, y, z, b) &= (x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2 - \{C(tr_2 + b - t_2)\}^2 = 0 \\f_3(x, y, z, b) &= (x - x_3)^2 + (y - y_3)^2 + (z - z_3)^2 - \{C(tr_3 + b - t_3)\}^2 = 0 \\f_4(x, y, z, b) &= (x - x_4)^2 + (y - y_4)^2 + (z - z_4)^2 - \{C(tr_4 + b - t_4)\}^2 = 0\end{aligned}$$

네 개의 미지수 x, y, z , 그리고 b 에 대해 정의된 네 개의 다변수 함수 f_1, f_2, f_3, f_4 로 정의된 비선형 연립 방정식 (system of nonlinear equations)의 근을 구하는 문제로 귀결된다. (참고로 군사 등의 목적으로 더욱 정밀한 위치 추적이 필요할 경우, 최대 12개까지의 위성 신호를 사용하며, 이때에는 비선형 방정식 시스템을 푸는 것이 아니라 주어진 비용 함수에 대한 최소화(minimization) 문제를 풀어야 한다.)

본 실습에서는 앞으로 3주간 이러한 문제를 풀기 위한 기본 이론 및 해결 방법, 그리고 몇 가지 수치 알고리즘에 대하여 배운 후, 공개 소프트웨어를 사용하여 위의 문제를 해결해주는 프로그램을 작성해보도록 한다.

2 비선형 방정식의 풀이 방법

앞절에서 제시한 방정식의 풀이 방법을 이해하기 전에 좀 더 쉬운 형태의 방정식의 풀이 문제에 대하여 알아보자.

임의의 함수 $f: \mathbb{R} \rightarrow \mathbb{R}$ 에 대해, $f(\alpha) = 0$ 을 만족하는 값 α 를 구하라.

여기서 실수 공간 \mathbb{R} 에 대해 정의된 함수 $f(x)$ 는 다음과 같이 일반적인 형태를 가지는 비선형 함수 (nonlinear equation)이며,

$$\begin{aligned} f(x) &= x^2 - 3x - 2 = 0 \\ f(x) &= x^5 + 2x^2 - 7x + 5 = 0 \\ f(x) &= x - a \sin x - b = 0 \\ f(x) &= e^x - e^{-x} - 3x = 0 \end{aligned}$$

위에서 $x = \alpha$ 와 같이 함수 값을 0으로 만들어 주는 값을 방정식의 근 (root)이라 한다 (그림 2 참조). 또한 방정식 $f(x) = 0$ 을 푸는 것을 근 찾기 (root finding)라고 한다.

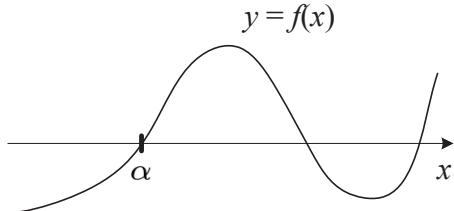


그림 2: 방정식의 근

물론 주어진 함수에 대한 근은 함수 형태에 따라 한 개도 존재하지 않을 수 있고, 또는 여러 개가 존재할 수도 있다. 근을 찾는 방법은 그 방식에 따라 몇 가지로 분류할 수 있는데, 본 실습에서는 그 중 가장 기본적인 몇 가지 방법에 대하여 연습하여 본다.

2.1 Newton-Raphson 방법

첫 번째로 살펴볼 방법은 방정식의 풀이 방법은 널리 쓰이고 있는 Newton-Raphson 방법이다.

방법

이 방법은 아래와 같이 주어진 초기값 x_0 에 대해 간단한 반복문을 통하여 근에 수렴하는 (근에 수렴하기 희망하는) 수열값 x_n ($n = 1, 2, 3, \dots$)을 구하게 된다.

```

 $x_0 \leftarrow$  an initial guess;
for  $n = 0, 1, 2, 3, \dots$  {
     $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ ;
}

```

이때 반복문이 무한정 수행이 되는 것이 아니라 적절한 순간에 종료를 해야하는데, 일반적으로 다음과 같은 종료 조건을 사용한다.

- (a) 현재 구한 x_{n+1} 에 대해 함수 값이 충분히 작은가? $\rightarrow |f(x_{n+1})| < \delta$ ¹⁾
- (b) 충분히 많은 회수만큼 반복문을 수행하였는가? $\rightarrow n \geq N_{max}$ ²⁾
- (c) 현재 구한 x_{n+1} 이 직전에 구한 x_n 에 비해 더 이상 의미 있는 진전을 하지 않는가? $\rightarrow |x_{n+1} - x_n| < \epsilon$ ³⁾

풀이 예

$f(x) = x^3 - 2x^2 + x - 3$ 이라고 하자. 이 함수에 대해 반복식은 다음과 같다.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{x_n^3 - 2x_n^2 + x_n - 3}{3x_n^2 - 4x_n + 1}$$

지금 초기값 $x_0 = 3.0$ 을 사용하면, 찾고자 하는 근 $x_{root} = 2.1745594102929799$ 에 아주 빠른 속도로 수렴함을 알 수 있다. 아래 그림은 이 방정식에 대하여 Newton-Raphson 방법을 적용한 예를 보여주고 있다.

i	xn1	f(xn1)	xn1-xroot
0	3.000000000000000e+000	9.0000e+000	8.2544e-001
1	2.437500000000000e+000	2.0369e+000	2.6294e-001
2	2.2130327163151096e+000	2.5636e-001	3.8473e-002
3	2.1755549387214881e+000	6.4634e-003	9.9553e-004
4	2.1745601006664459e+000	4.4791e-006	6.9037e-007
5	2.1745594102933126e+000	2.1574e-012	3.3262e-013
계속하려면 아무 키나 누르십시오 . . .			

그림 3: Newton-Raphson 방법을 사용한 풀이 예

-
- 1) 근의 조건임.
 - 2) 근에 수렴을 하였건 아니건 유한한 시간 안에 종료해야함.
 - 3) 반복문을 수행해도 x_{n+1} 값에 변화가 없다면 ...

직관적인 원리

그러면 Newton-Raphson 방법은 왜 이렇게 빠른 속도로 근에 수렴할까? 그림 4를 보면 이 방법의 직관적인 원리를 쉽게 이해할 수 있다. 현재까지 구한 근에 대한 근사 추정 값 x_n 을 고려하자. 잘 알다시피 이 함수에 대해 $x = x_n$ 인 지점에서의 접선의 방정식은 $y = f(x_n) + f'(x_n)(x - x_n)$ 이다. 이 직선이 x 축과 만나는 지점은 $y = f(x_n) + f'(x_n)(x - x_n) = 0$ 으로부터 $x = x_n - \frac{f(x_n)}{f'(x_n)}$ 이 됨을 쉽게 알 수 있다. 바로 이 값이 Newton-Raphson 방법의 반복문에서 구하는 다음 추정값 x_{n+1} 이 되는데, 직관적으로 x_n 보다 근에 더 접근한 것을 알 수 있다. 즉 다시 말해서 Newton-Raphson 방법은 현재까지 구한 $x = x_n$ 지점에서 함수 그래프에 대한 접선을 그어 그 직선이 x 축과 만나는 지점을 x_{n+1} 로 설정하는 과정을 반복하여 우리가 원하는 근 $x = \alpha$ 에 수렴하게 하는 방식을 취한다.

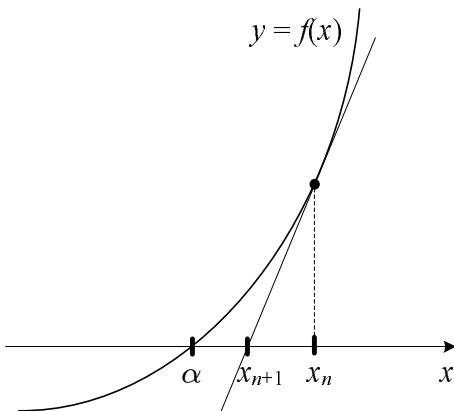


그림 4: Newton-Raphson 방법의 원리

2.2 Secant 방법

두 번째로 살펴볼 방법은 방정식의 풀이 방법은 Newton-Raphson 방법 대용으로 사용할 수 있는 Secant 방법이다.

방법

Newton-Raphson 방법은 널이 쓰이는 우수한 방법이나 몇 가지 문제가 존재한다. 그 중 하나는 매번 반복문이 수행 될 때마다 임의의 지점 x_n 에서 미분값 $f'(x_n)$ 을 계산해주어야 한다는 점이다. 실제로 다양한 응용 문제를 수치적으로 풀고자 할 때 미분값을 계산하는 과정은 비용이 많이 들거나, 또는 매우 어려운 경우가 종종 발생한다. 따라서 Secant 방법은 반복식

에서 사용하는 미분값 $f'(x_n)$ 을 다음과 같이 근사적으로 추정하여 사용하게 된다.

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$$

따라서 이를 Newton-Raphson 방법의 반복식에 대입하여, $x_{n+1} = x_n - f(x_n) \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$ 와 같은 반복문을 사용하게 되며, 따라서 Secant 방법을 요약하면 다음과 같다.

```
x0,x1 ← two initial guesses;
```

```
for n = 1,2,3,⋯ {
```

$$x_{n+1} = x_n - f(x_n) \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})};$$

```
}
```

직관적인 원리

Secant 방법은 매번 반복문 수행 시 Newton-Raphson 방법과는 달리 두 개의 근에 대한 추정값 x_{n-1} 과 x_n 을 사용하여 다음 추정값 x_{n+1} 을 구하게 된다. 그럼 5는 Secant 방법의 직관적인 원리를 도시하고 있다. 전기한 바와 같이 Secant 방법은 Newton-Raphson 방법에서 설명한 접선의 기울기 $f'(x_n)$ 을 $\frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$ 로 근사를 한다. 따라서 이전 방법에서 사용한 접선을 Secant 방법에서는 함수 그래프에서 $x = x_{n-1}$ 과 $x = x_n$ 지점을 지나는 직선으로 근사화하여, x 축과의 교점을 구해 다음 추정값 x_{n+1} 을 구함을 확인할 수 있다.

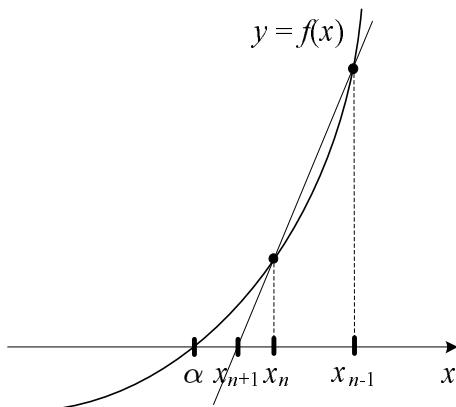


그림 5: Secant 방법의 원리

2.3 Newton-Raphson 방법과 Secant 방법의 비교

전기한 바와 같이 Secant 방법은 Newton-Raphson 방법과는 달리 매번 미분값을 계산할 필요가 없다는 장점이 있지만, 일반적으로 추정값 x_{n+1} 이 근에 수렴하는 속도가 느려져 더 많은 회수로 반복문을 수행해주어야 하는 경우가 발생한다. 여기서 이론적인 측면에서 수렴 속도란 어떻게 정의될까?

지금 어떤 방법을 사용하건 위에서 기술한 방식대로 주어진 초기값(들)로 부터 반복문 수행을 통해 구한 수열 $x_0, x_1, x_2, x_3, \dots$ 이 근 $x = \alpha$ 에 수렴한다고 할 때, 매번 구한 값 x_n 에는 $\varepsilon_n = |x_n - \alpha|$ 만큼의 절대 오차 (absolute error)가 존재하게 된다. 이 경우 Newton-Raphson 방법은 2차 수렴 (quadratic convergence)을 한다고 하는데, 이는 수학적으로 다음을 의미하며, 사실 어렵지 않게 증명할 수 있다.

$$\lim_{n \rightarrow \infty} \frac{\varepsilon_{n+1}}{\varepsilon_n^2} = c \quad (c > 0)$$

여기서 c 는 양수인 상수를 의미하는데, 이를 쉽게 설명하면, 어느 정도 반복문이 수행된 후에는 절대 오차가 대략적으로 $\varepsilon_{n+1} \approx c\varepsilon_n^2$ 와 같은 모습으로 감소하게 됨을 의미한다. 예를 들어, $c = 10$ 인 경우 Newton-Raphson 방법을 사용하면 대략적으로 아래와 같은 패턴으로 오차가 감소하는데,

$$\begin{aligned}\varepsilon_0 &\approx 0.01 = 10^{-2} \\ \varepsilon_1 &\approx 10\varepsilon_0^2 = 10^{-3} \\ \varepsilon_2 &\approx 10\varepsilon_1^2 = 10^{-5} \\ \varepsilon_3 &\approx 10\varepsilon_2^2 = 10^{-9} \\ \varepsilon_4 &\approx 10\varepsilon_3^2 = 10^{-17} \\ &\vdots\end{aligned}$$

Newton-Raphson 방법은 매우 빠른 속도로 근에 수렴함을 알 수 있다.⁴⁾

반면에 Secant 방법은 차수가 1.62 (정확히 말해서 $\frac{1+\sqrt{5}}{2}$)인 속도로 근에 수렴한다. 즉 수학적으로 다음과 같이 표현할 수 있는데,

$$\lim_{n \rightarrow \infty} \frac{\varepsilon_{n+1}}{\varepsilon_n^{1.62}} = c \quad (c > 0),$$

4) Newton-Raphson 방법은 항상 2차 수렴을 하는 것은 아니고 근이 단근 (simple root)이어야 한다는 조건이 있다.

앞에서와 같이 어느 정도 반복문이 수행된 후에는 절대 오차가 대략적으로 $\varepsilon_{n+1} \approx c\varepsilon_n^{1.62}$ 와 같은 모습으로 감소하게 된다. 이를 Newton-Raphson 방법에서와 같은 조건 ($\varepsilon \approx 0.01$ 과 $c = 10$) 하에 오차 감소 과정을 보면 다음과 같은데, 역시 속도가 Newton-Raphson 방법보다는 느림을 알 수가 있다.

$$\begin{aligned}\varepsilon_0 &\approx 0.01 = 10^{-2} \\ \varepsilon_1 &\approx 10\varepsilon_0^{1.62} = 5.76 \cdot 10^{-3} \\ \varepsilon_2 &\approx 10\varepsilon_1^{1.62} = 2.35 \cdot 10^{-3} \\ \varepsilon_3 &\approx 10\varepsilon_2^{1.62} = 5.51 \cdot 10^{-4} \\ \varepsilon_4 &\approx 10\varepsilon_3^{1.62} = 5.53 \cdot 10^{-5} \\ \varepsilon_5 &\approx 10\varepsilon_4^{1.62} = 1.17 \cdot 10^{-6} \\ \varepsilon_6 &\approx 10\varepsilon_5^{1.62} = 2.46 \cdot 10^{-9} \\ &\vdots\end{aligned}$$

3 실습 문제

앞에서 배운 Newton-Raphson 방법과 Secant 방법을 사용하여 비선형 방정식을 풀어보자.

실습 문제 1-1

- (i) 방정식 $f_1(x) = x^2 - 4x + 4 - \ln x = 0$ 의 근을 Newton-Raphson 방법 (프로그램 1-1)과 Secant 방법 (프로그램 1-2)을 사용하여 구하는 프로그램을 작성 하라.

- 각 프로그램 작성 시 풀고자 하는 함수와 미분 함수 (Newton-Raphson 방법의 경우)는, 이름이 `function.cpp`인 파일에 다음과 같은 이름과 프로토타입을 가지는 함수로 정의하라 (조교가 이 파일만 다른 함수를 정의한 파일로 대체하여 프로그램이 정확히 수행이 되는지 확인할 예정임).

```
// Function f(x)
double _f(double x) {
    :
}
// 1st derivative of f(x)
double _fp(double x) {
    :
}
```

- 두 프로그램에서 사용하는 종료 조건 인자 δ , N_{max} , 그리고 ϵ 은 이름이 `my_solver.h`인 헤더 파일에서 다음과 같이 지정한 후, 이 파일을 적절히 사용하라 (δ 값으로 10^{-6} 정도를 사용해보고, 실제 실습 시 사용할 값은 자신이 정할 것).

```
#define DELTA 0.000001
#define Nmax 50
#define EPSILON 0.00001
```

- 그 외의 모든 코드는 두 프로그램 각각에 대해 `program1_1.cpp`와 `program1_2.cpp`인 파일에 저장하라.
- 각 프로그램을 수행시키면, 처음에 초기값 (Newton-Raphson 방법은 한 개, 그리고 Secant 방법은 두 개)을 콘솔 윈도우에서 입력 받도록 하라.

- 주어진 초기값(들)에 대해 방정식을 푸는 과정을 그림 3와 비슷한 형태의 내용으로 (내용은 각자 자유롭게 설계) 이름이 `result.txt`인 파일에 출력하라.
 - 마지막으로 자신의 프로그램에서 구한 근을 적절한 포맷과 내용을 사용하여 콘솔 윈도우에 출력하라.
- (ii) 두 방법에 대한 초기값을 각각 $x_0 = 3.0$ 과 $x_0 = 2.0, x_1 = 4.0$ 으로 설정하여 자신이 작성한 프로그램을 수행시켜 자신의 프로그램이 원하는 근을 정확히 찾고 있는지 분석하라. 과연 근이 맞는지 어떻게 확인할 수 있을까? 논리적으로 타당한 방법으로 분석한 내용을 보고서에 기술하라.
- (iii) 위에서 산출한 결과를 볼 때, 각 방법의 근의 수렴 속도가 과연 앞에서 설명한 속도로 수렴하는지 비교 분석한 후 그 결과를 보고서에 기술하라.⁵⁾
- (iv) 위에서 제시한 초기값 외에 임의의 초기값들을 사용하여 자신이 작성한 프로그램을 수행한 후, 이 두 방법이 항상 임의의 초기값에 대해 빠르게 수렴하는지 보고서에 기술하라.
- (v) 위에서 작성한 프로그램을 사용하여 함수 $f_2(x) = x + 1 - 2 \sin \pi x = 0$ 에 대해 초기값을 각각 $x_0 = 0.75$ 과 $x_0 = 0.5, x_1 = 1.0$ 으로 설정하여 위 문제를 반복하라.

[참고] 부동 소수점 숫자의 저장 및 연산은 모두 `double` 타입의 변수 및 연산을 사용하라.

5) 여기서 수렴 속도는 원하는 정밀도를 가지는 근 추정값을 구하는데 필요한 반복문의 수행 회수를 의미한다고 하자.

실습 문제 1-2

다음과 같은 다항식에 대한 방정식을 고려하자.

$$f_3(x) = x^4 - 11.0x^3 + 42.35x^2 - 66.55x + 35.1384 = 0$$

지금 어떤 근 분리 (root separation) 방법을 사용하여, 이 방정식은 네 개의 서로 다른 실근을 가지며, 각 실근은 [1.02, 1.48], [1.95, 2.37], [3.11, 3.73], [3.83, 4.61] 구간에 존재한다는 사실을 밝혀냈다. 이 사실을 근거로 Newton-Raphson 방법을 사용하여 모든 실근을 구하라.

[참고 1] 부동 소수점 숫자의 저장 및 연산은 모두 `double` 타입의 변수 및 연산을 사용하라.

[참고 2] 초기값의 좋은 후보로서 각 구간의 중점을 고려할 수 있는데, 자신의 판단 하에 사용하라.

[참고 3] 다음 주 실습에서는 이 방정식의 근을 공개 소프트웨어를 사용하여 해결해볼 예정임.

실습 문제 1-3

주어진 양의 실수 α 에 대한 제곱근 (square root) $\sqrt{\alpha}$ 는 $f(x) = x^2 - \alpha = 0$ 과 같은 비선형 방정식의 양의 근을 구하므로서 얻을 수 있다. 이러한 형태의 방정식에 대하여 Newton-Raphson 방법과 Secant 방법 각각을 적용하여 $\sqrt{2}$ 를 구하여 보자.

- (i) 이 문제를 풀어주는 프로그램 작성 시 double 타입의 부동 소수점 숫자를 사용하라.
- (ii) 두 방법 각각에 대하여 공정한 비교가 가능하도록 초기값 x_0 을 100.0 부근에서 적절히 동일하게 설정한 후, 근에 충분히 수렴할 때까지 반복을 하면서, 각 줄에 a) 반복 회수 i (0부터 시작), b) 현재까지 추정한 값 x_i , 그리고 c) 정확한 값에 대한 절대 오차 $|x_i - \sqrt{2}|$ 를 출력하라. $\sqrt{2}$ 에 대한 정확한 값으로 다음 값을 사용하고,

$$\sqrt{2} = 1.4142135623730950488$$

double 타입의 숫자를 출력할 때에는 십진수 유효숫자 15개를 출력하라.

- (iii) 보고서에 각 방법에 대한 결과를 첨부하고, 각 방법의 근에 대한 수렴 속도가 이론적으로 습득한 내용과 일치하는지를 비교분석하여 기술하라.

실습 문제 1-4

- (i) **실습 문제 1-1**에서 작성한 자신의 코드의 single-precision 버전, 즉 모든 계산을 float 타입의 계산을 수행하는 프로그램을 작성하라. 여러분이 작성한 코드는 **실습 문제 1-1**과 동일한 방식으로 파일에 저장을 하는데, single-precision 버전임을 강조하기 위해 해당하는 파일 이름 앞에 sp_-를 붙일 것(즉 파일 이름이 sp_function.cpp, sp_program1_1.cpp, 그리고 sp_program_2.cpp이어야 함).
- (ii) 다음 $f_1(x) = \ln x - 1 = 0$ 과 같이 근을 알고 있는 비선형 방정식을 고려하자(잘 알다시피 근은 $e = 2.718281828459045235360287471352\cdots$ 임). 이 방정식의 근을 Newton-Raphson 방법을 사용하여 구하려 하는데, 적절한 초기 값 x_0 에 대해 자신이 작성한 double-precision 버전과 single-precision 버전 각각을 사용하여 근을 구하여 보자. 이때 부동 소수점 연산의 정밀도가 다른 두 방법이 구한 근의 값이 정확한 근 e 와 비교하여 어떤 차이가 있는지, 자신이 알아낸 사실을 보고서에 상세히 기술하라.

실습 문제 1-5

다음과 같은 음대수 함수에 의해 정의되는 Three-leaved clover 곡선을 고려하자(그림 6 참조).

$$f_5(x, y) = x^4 + 2x^2y^2 + y^4 - x^3 + 3xy^2 = 0$$

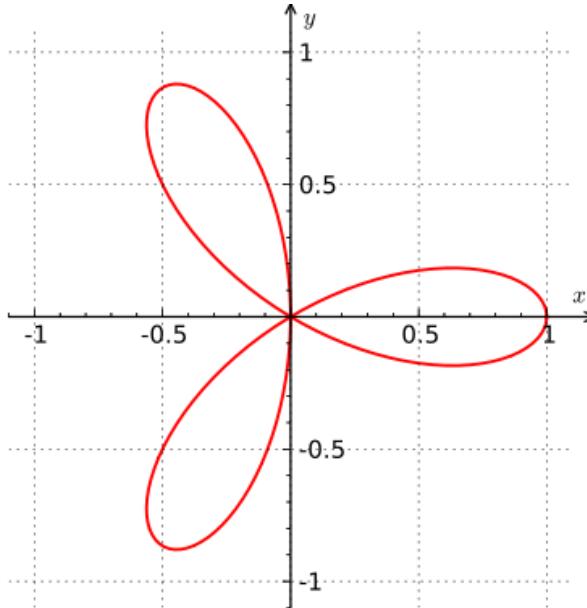


그림 6: Three-leaved clover 곡선

이제 이 곡선을 2차원 평면에서 움직이고 있는 임의의 물체라고 가정하고(실제로 시간의 함수인 이동 변환과 회전 변환을 통하여 이 물체가 임의의 방식으로 움직이게 할 수 있음), 직선 $y = ax + b$ 에 의해 정의되는 광선과 이 물체와의 2차원 평면상에서의 모든 교차점을 Newton-Raphson 방법을 사용하여 구하려 한다.

- (i) 대략적인 직선의 모양을 생각한 후, 직선에 대한 계수 a 와 b 를 직관적으로 적절히 설정한다.
- (ii) 직선 식의 y 를 $f_5(x, y) = 0$ 에 대입하여 비선형 방정식 $f(x) = 0$ 을 생성한다.
- (iii) 해당 직선에 대하여 실근이 몇 개가 있을지를 직관적으로 판단한 후, 각 경우에 대하여 적절한 초기값 x_0 를 설정한다.
- (iv) 각 초기값에 대하여 Newton-Raphson 방법을 사용하여 근을 구한 후, 이를 통하여 해당 교차점에 대한 이차원 좌표 (x^*, y^*) 를 구한다.

- (v) 이렇게 구한 각 좌표값을 원래의 함수에 대입하여 오차 $|f_5(x^*, y^*)|$ 의 크기가 만족할 만한 수준인지를 각 근에 대하여 확인한다.

[참고 1] 자신이 작성할 프로그램 `program1_5.cpp`에서 직선을 대입하여 생성한 비선형 방정식의 함수에 대해 다음과 같은 프로토타입을 고려하자.

```
double f(double x, double a, double b);
```

또한 그에 대한 1차 도함수도 아래와 같이 정의할 수 있다.

```
double fp(double x, double a, double b);
```

[참고 2] 위의 (i)부터 (v)까지의 과정을 서로 다른 상황에 대하여 세 차례 반복하여 이번 주 숙제의 기초 자료로 활용하라.

[참고 3] 세 가지 경우 각각에 대하여 위의 (i)부터 (v)까지의 과정의 수행 결과를 보고서에 간결히 기술하라.

4 숙제

제출 마감: ?월 ?일 오후 ?시 정각

제출물 및 방법: 조교가 실습 시간에 공지

이번 주의 숙제는 다음과 같다.

숙제 1

비선형 방정식의 근을 찾아주는 방법의 하나로 Bisection 방법을 들 수가 있다. 이 방법은 근을 포함하는 초기 구간 $[a_0, b_0]$ 을 시작 구간으로 하여,⁶⁾ 매 반복문에서 현재까지 구한 구간 $[a_n, b_n]$ 을 상황에 맞게 적절히 $[a_n, \frac{a_n+b_n}{2}]$ 또는 $[\frac{a_n+b_n}{2}, b_n]$ 으로 반으로 분할하여 다음 구간 $[a_{n+1}, b_{n+1}]$ 로 사용하는 방식을 취한다. 이때 각 구간의 중점 $x_n = \frac{a_n+b_n}{2}$ 은 Newton-Raphson 방법과 Secant 방법에서 계산하는 수열에 해당한다.

- (i) 자료 조사를 통하여 Bisection 방법을 정확히 이해한 후, 이 방법을 사용하여 방정식의 근을 구해주는 프로그램을 작성하라 (프로그램 1-3).
- (ii) 작성 방법은 **실습 문제 1-1**에서 기술한 내용과 동일하다. 다만 해당 코드를 이름이 `program1_3.cpp`인 파일에 저장하고, Secant 방법에서처럼 두 개의 값 (여기서는 초기 구간의 경계값 a_0 와 b_0)을 읽어들이도록 하라.
- (iii) 프로그램이 완성되면, 실습 시간에 사용한 세 개의 함수 $f_1(x)$, $f_2(x)$, 그리고 $f_3(x)$ 에 대해 적절한 초기 구간을 사용하여 올바르게 근에 수렴하는지에 대한 분석 내용을 보고서에 기술하라.
- (iv) Bisection 방법의 수렴 속도는 선형적인, 즉 $\epsilon_{n+1} \approx \frac{1}{2}\epsilon_n$ 형태를 보인다. 위의 세 함수에 대한 방정식을 대상으로 Newton-Raphson 방법, Secant 방법, 그리고 Bisection 방법을 적용해보고, 과연 각 방법이 이론적인 수렴 속도를 보이는지를 분석하고 그 내용을 보고서에 기술하라.⁷⁾

[참고 1] 부동 소수점 숫자의 저장 및 연산은 모두 `double` 타입의 변수 및 연산을 사용하라.

6) 그 구간에서 함수 $y = f(x)$ 연속이고, $f(a_0)f(b_0) < 0$ 인 조건을 만족하며, 그 구간에 정확히 한 개의 근이 존재한다고 가정함.

7) 앞에서와 같이 수렴 속도는 원하는 정밀도를 가지는 근 추정값을 구하는데 필요한 반복문의 수행 회수를 의미한다고 하자.

[참고 2] 조교는 여러분이 제출한 Newton-Raphson 방법, Secant 방법, 그리고 Bisection 방법에 대한 코드를 다음 중 적절한 방정식을 사용하여 작동 여부를 확인할 예정이다 (Secant 방법은 적절히 초기값 설정할 것).

$$f_4(x) = x^3 - 2x - 5, \quad x_0 = 2$$

$$f_5(x) = e^x - 2, \quad x_0 = 1$$

$$f_6(x) = x^6 - x - 1, \quad x_0 = 1$$

$$f_7(x) = x + \tan x, \quad x_0 = 3$$

$$f_8(x) = 2 - x^{-1} \ln x, \quad x_0 = \frac{1}{3}$$

$$f_9(x) = x^2 - \sin x, \quad x_0 = \frac{1}{2}$$

$$f_{10}(x) = 1 - 2xe^{-\frac{x}{2}}, \quad x_0 = 0$$

$$f_{11}(x) = e^x - 1.5 - \tan^{-1} x, \quad x_0 = -7$$

$$f_{12}(x) = x^{-1} - \tan x, \quad [a_0, b_0] = [0, \frac{\pi}{2}]$$

$$f_{13}(x) = x^{-1} - 2^x, \quad [a_0, b_0] = [0, 1]$$

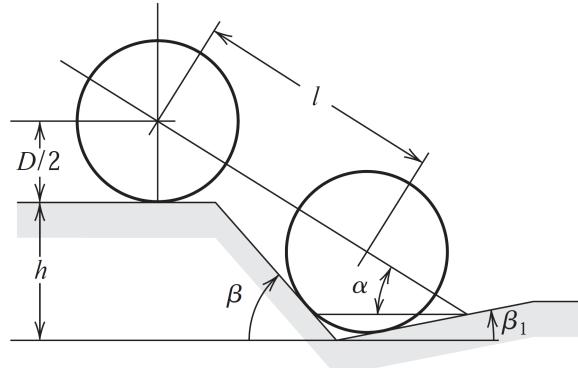
$$f_{14}(x) = 2^{-x} + e^x + 2 \cos x - 6, \quad [a_0, b_0] = [1, 3]$$

$$f_{15}(x) = \frac{x^3 + 4x^2 + 3x + 5}{2x^3 - 9x^2 + 18x - 2}, \quad [a_0, b_0] = [0, 4]$$

[참고 3] 여러분이 작성한 Newton-Raphson 방법, Secant 방법, 그리고 Bisection 방법은 다음 추 실습에서 아주 중요하게 쓰일 예정이므로 반드시 코드를 가지고 실습 수업에 참여할 것.

숙제 2

다음 그림은 협한 지형에서도 운행이 가능한 자동차의 설계에 관한 것이다. (참고: Bekker, M. G., *Introduction to Terrain Vehicle Systems*, University of Michigan Press, 1969.)



이 그림에 표시된 여러 인자에 대하여 다음과 같은 방정식이 성립한다고 하는데,

$$f(\alpha) = A \sin \alpha \cos \alpha + B \sin^2 \alpha - C \cos \alpha - E \sin \alpha = 0,$$

여기서 각 상수 값은 다음과 같이 정의된다.

$$A = l \sin \beta_1,$$

$$B = l \cos \beta_1,$$

$$C = (h + 0.5D) \sin \beta_1 - 0.5D \tan \beta_1,$$

$$E = (h + 0.5D) \cos \beta_1 - 0.5D.$$

만약 $l = 89$ 인치, $h = 49$ 인치, $D = 55$ 인치, 그리고 $\beta_1 = 11.5$ 도이라면, 각도 α 는 대략 33도 정도가 됨이 알려져 있다. 여러분이 작성한 Newton-Raphson 방법을 사용하여 정확한 각도 α 값을 구하라.

[참고 1] 숙제 1에서 자신이 작성한 Newton-Rapson 방법을 사용하여 이 문제를 해결해라.

[참고 2] 이를 위하여 어떠한 반복문을 사용하였는지 보고서에 관련 수식을 설명한 후, 자신이 구한 답에 수렴해가는 반복 결과와 함께 제출하라.

숙제 3

실습 문제 1-5를 다시 고려하자.

그림 6의 three-leaved clover 곡선상의 점의 좌표 $(x(t), y(t))$ 는 매개 변수 t 를 사용하여 다음과 같이 표현할 수 있다.

$$x = x(t) = \cos(3t) \cos t, \quad y = y(t) = \cos(3t) \sin t, \quad t \in [0, \pi]$$

실습 시간에 자신이 정한 직선 $y = ax + b$ 의 식에 t 의 함수인 $x(t)$ 와 $y(t)$ 의 식을 대입하면 이 곡선과 직선의 교차점을 t 에 대한 비선형 방정식 $f(t) = 0$ 으로 표현할 수 있다. 이러한 상황에서 Newton-Raphson 방법을 사용하여 모든 교차점을 구하려 하는데(답은 실습 시간에 자신이 만들었음), 과연 어떠한 방식으로 문제를 해결할 수 있을까?

[참고 1] 한 가지 방법은 구간 $[0, \pi]$ 을 잘게 나누어 각 구간 중점의 t 값을 초기값 t_0 로 하여 근을 찾아보는 것이다(비록 비효율적으로 보일 뿐만 아니라 모든 근을 찾아준다는 보장이 없지만).

[참고 2] 자신이 실습 시간에 만든 세 가지 경우에 대하여 이 문제를 해결한 후 자신이 취한 실험 방식과 그에 대한 결과를 보고서에 상세히 기술하여 제출하라. 필요 시 (손으로) 그림을 그려 설명할 것.