[수치해석] NA-rootfinding1

F(t) = 0을 만족시키는 t의 값을 찾는 방법

수식으로 해결 가능한 경우에는 수식으로 해결하면 된다.

- **Analytic solution**이 존재, f(t) = 0이 **closed-form solution**을 갖는다.

Analysic solution이 없는 경우에는 Root finding하는 과정이 필요하다.

- A**pproximation**하여 추정값을 얻는다.

Root finding의 기본적인 과정

Step 1: Solution이 존재할 만한 구간을 Rough하게 잡아놓는 과정

1. **Grapical method**: 그래프에서 f(t) = 0을 찾는다.
2. **Incremental search**: Initial value (x0)에서부터 Step size (dx)씩 증가하며 Solution이 있을법한 구간을 찾는다.
    a. **[x, dx] 구간**에서 $f(x) * f(dx) < 0$인지 확인하고 결과가 음수라면 [x, dx] 구간에 Solution이 있음을 의미한다.
    b. **Multiple roots (중근)**의 경우에는 **[x, dx] 구간**에서 solution이 존재함에도 불구하고 $f(x) * f(dx) >= 0$일 수 있어 **Incremental search는 Multiple roots를 쉽게 찾을 수 없다.**
        i. Multiple roots를 다루기 위해 [x, dx] 구간의 $f'(x) * f('dx) < 0$인지를 추가로 확인할 수 있다.
        ii. $f'(x) * f('dx) < 0$이라면 [x, dx] 구간 내에 변곡점 $f''(x) = 0$이 존재함을 의미하고 이는 극값이 존재할 가능성이 있음을 의미한다.
    c. **Step size (dx)의 크기**가 중요하다.
        i. 너무 작으면 Step 1이 너무 오래걸린다.

        ii. 너무 크면 중간 Solution을 놓필 수 있다.

3. **Experience**
4. **Simplified model의 Solution을 그대로 사용: 선형 / 근사 이용하여 쉽게**

Step 2: Step 1에서 찾은 구간을 기반으로 정확한 Solution을 찾는 과정

1. **Bracketing method**
2. **Open method**

Bracketing method

Step 1에서 얻은 구간 [a, b]에 Solution이 존재한다는 가정 하에, **Step 1의 구간 [a, b]를 벗어나지 않으면서 탐색 구간을 줄여가며 Solution을 찾는 방법**이다.

초기 구간 [a, b]에 Solution이 반드시 존재한다는 가정이 있기 때문에 **[a, b] 내에서 구간을 줄여가다 보면 반드시 수렴하게 된다는 장점**이 있다.

하지만, **느리고 Multiple roots를 다루지 못 한다는 단점이 있다.**

1. Bisection method

직전 Iteration에서 주어진 구간의 중간 값과 양쪽 끝 값을 이용하여 구간을 반씩 줄여가는 방법

- Half Interval method, Bolzano method라고도 부른다.

과정

a. 먼저 **주어진 구간 [a, b]에 대해** $f(a) * f(b) < 0$**인지 확인**한다.

b. 중간 값 $\frac{a+b}{2}$에 대해 $f(a) * f(\frac{a+b}{2}) < 0$인지 확인한다.

    i. (b)가 참이라면 다음 구간을 $[a, \frac{a+b}{2}]$로 변경하고 (b)로 돌아간다.

c. 중간 값 $\frac{a+b}{2}$에 대해 $f(b) * f(\frac{a+b}{2}) < 0$인지 확인한다.

    i. (b)가 참이라면 다음 구간을 $[\frac{a+b}{2}, b]$로 변경하고 (b)로 돌아간다.

d. Relative error of x가 Desired relative error보다 작으면 반복을 중단한다.

특징

- 각 Iteration마다 구간이 반씩 줄어든다.
- 간단하다.
- n번째 Iteration에 대해 Maximum error는 $\frac{b-a}{2^n}$이다.
    - 초기 구간이 [a, b]
    - n번째 Iteration일 때 구간의 길이가 $\frac{b-a}{2^n}$이다.
- 느리다.
- Multiple roots 경우에는 Root를 찾을 수 없다.

2. Linear Interpolation method

직전 추정값과 현재 추정값의 평균 기울기를 이용하여 Root를 예측하는 방법

- False position method라고도 부른다.

과정

a. 먼저 **주어진 구간 [a, b]에 대해** $f(a_n) * f(b_n) < 0$**인지 확인**한다.

b. $p_{n+1} = a_n - \frac{f(a_n)(b_n - a_n)}{f(b_n) - f(a_n)}$을 계산한다.

    i. p값은 $(a_n, f(a_n))$와$(b_n, f(b_n))$를 이은 직선의 X절편이다.

c. $f(a_n) * f(p_{n+1}) < 0$이라면 $a_{n+1} = a_n, b_{n+1} = p_{n+1}$으로 다음 구간을 정하고, (a)로 돌아간다.

d. $f(b_n) * f(p_{n+1}) < 0$이라면 $a_{n+1} = a_n, b_{n+1} = p_{n+1}$으로 다음 구간을 정하고, (a)로 돌아간다.

특징

- **Bisection method보다 Convergence speed가 빠르다**.
- f(x)의 **Curvature**에 따라 Convergence speed가 달라진다.
    - Linear Interpolation method는 기본적으로 Linear하기 때문에 **f(x)의 Curvature가 작아 linear function에 가까워 질수록 Convergence speed가 빠르다.**
    - Linear Interpolation method의 f(x)의 Curvature가 클수록 Convergence speed가 느려지기 때문에 **Modified Linear Interpolation method**가 사용되기도 한다.

3. Modified Linear Interpolation method

Linear Interpolation method에서 두 연속된 Iteration에서 a_n, b_n 중 같은 것이 고정되는 구간으로 연속적으로 선택되면 고정된 쪽의 함수값에 0.5배를 하여 p_{n+1}을 계산하는 방법.

- (n-1)번째 Iteration에서 $a_n = a_{n-1}$이고, n번째 Iteration에서도 $a_{n+1} = a_n$인 경우
- (n+1)번째 Iteration에서 p_{n+2}를 구할 때, $p_{n+2} = \frac{0.5*f(a_{n+1})*(b_{n+1}-a_{n+1})}{f(b_{n+1})-0.5*f(a_{n+1})}$와 같이 고정된 쪽 ($a_{n+1}$)의 함숫값에 0.5배를 하여 p를 계산한다.

특징

- Linear Interpolation method에 비해 수렴 속도가 빠르다.
- Curvature가 큰 경우, 구간의 한쪽이 계속하여 고정되는 경향이 있다. 이를 보완하기 위한 방법이다.

Open method

Step 1에서 얻은 구간 [a, b]에서 임의의 Initial point를 잡고 Regular iteration을 반복하여 Root를 찾는 반복

Step 1에서 얻은 구간 [a, b]는 Initial point만 잡는 용도이고, 이후 **탐색 구간이 [a, b]로 제한되지 않기 때문에 수렴이 보장되지 않는다.**

하지만, **Bracketing method보다 빠르며 Multiple roots를 다룰 수 있다는 장점**이 있다.

Multiple roots를 다룰 수는 있으나 **일반 Root를 찾을 때에 비해 Multiple root를 찾는 경우 Convergence speed가 다소 느려진다.**

1. Newton-Raphson method

x_i에서의 접선의 X절편을 다음 추정치 x_{i+1}로 사용하는 방법

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

- x_i에서의 접선을 구하면 $y = f'(x_i)(x - x_i) + f(x_i)$이다. 접선의 x절편을 x_{i+1}이라고 하면 위 수식을 얻을 수 있다.

과정

a. Initial value를 이용하여 다음 추정값을 찾는다.
b. (a)의 추정값에서의 접선을 이용하여 X절편을 찾고, 해당 X절편을 다음 추정치로 사용한다.
c. Relative error of x가 Desired error보다 작을 때까지 (b) 작업을 반복한다.

특징

- **Quadratic convergence로 굉장히 빠른 Convergence speed**를 갖는다.
- 도함수를 사용하기 때문에 **미분 계산이 어려운 함수인 경우에는 불리**하다.
- Initial value에 따라 **Root를 찾지 못 할 수 있기에 Initial guess가 중요**하다.
- **Cycling, Wandering, Overshooting**이 발생할 수 있다.

  - Cycling: 같은 값들을 반복하게됨
  - Wandering: Root가 있는 반대 방향으로 Divergence
  - Overshooting: 한번에 큰 Step으로 x가 이동
- **Convergence speed가 빠르지만, Stable하지 않다.**

Error Analysis

- N-R Method가 접선의 X절편을 찾는 방법이기 때문에 아래와 같이 쓸 수 있다.

$$0 = f'(x_i)(x_{i+1} - x_i) + f(x_i)$$

- 실제 Root 값을 x라고 할 때, Root의 함숫값 $f(x_r) = 0$**을 Taylor series의 1차 전개로 표현한 식**은 아래와 같다.

$$0 = f(x_i) + f'(x_i)(x_r - x_i) + \frac{f''(Epsilon)}{2!}(x_r - x_i)^2$$

- 위 두 식을 연립해서 아래의 식을 얻을 수 있다.

$$0 = f'(x_i)(x_r - x_{i+1}) + \frac{f''(Epsilon)}{2!}(x_r - x_i)^2$$

- $E_{t,i+1} = x_r - x_{i+1}, E_{t,i} = x_r - x_i$ 이라고 하면, $E_{t,i+1}$은 아래와 같이 정의된다.

$$E_{t,i+1} = -\frac{f''(x_r)}{2f'(x_r)}E_{t,i}^2$$

- $E_{t,i+1} \approx E_{t,i}^2$**이므로 N-R Method를 Quadratic convergence라고 한다.**

N-R Method의 Multiple roots 처리

아래 N-R Method의 Update 식에서 $f(x) = f'(x) = 0$인 경우 **Zero-division** 문제가 발생하게 된다.

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

일반적으로 $f(x) = f'(x) = 0$**인 경우에 f(x) = 0이 f′(x) = 0보다 빨리 이루어진다**고 한다.

- 때문에 **N-R Method에서 f(x) = 0이 되는 것을 확인하자마자 N-R Method의 반복을 멈추면 Zero-division 문제가 발생하지 않게 된다.**

다른 방법으로 $U(x) = \frac{f(x)}{f'(x)}$ **라는 함수를 N-R Method의 Update 식에서 f(x) 대신 사용하는 방법**이 있다.

- f(x)가 원래 m개의 Mutiple roots를 가졌다면 f′(x)는 (m-1)개의 Multiple roots를 가지게 되고, 결과적으로 U(x)는 Multiple roots를 가질 수 없게 되어 Zero-division 문제가 해결된다.

$U(x) = \frac{f(x)}{f'(x)}$ **함수**를 이용한 Update 식은 아래와 같다.

$$x_{n+1} = x_n - \frac{U(x)}{U'(x)} = x_n - \frac{f(x_i)f'(x_i)}{f'(x_i)^2 - f(x_i)f''(x_i)}$$

2. Secant method

$$p_{n+2} = p_{n+1} - \frac{f(p_{n+1})(p_{n+1} - p_n)}{f(p_{n+1}) - f(p_n)}$$

- Linear Interpolation method와 동일한 방식, Update 방식만 다르다.

과정

a. p_0, p_1이 Initial values로 주어지고, 위 공식에 따라 p_2를 계산한다.

b. p_n, p_{n+1}에 대해 p_{n+2}를 위 공식에 따라 계산한다.

c. (b)를 Relative error of p가 Desired error보다 작을 때까지 반복한다.

특징

- Convergence speed가 Newton-Raphson method보단 **느리지만, Stable**하다.
- **Linear Interpolation method보다 빠르다**.

Convergence

$$e_{k+1} = (\frac{1}{2}\frac{f'(x)}{f(x)})^{0.618} e_k^{1.618}$$

- **f(x)의 도함수를 구하기 복잡한 경우 N-R Method보다 효율적**일 수 있다.

3. Modified Secant method

N-R Method의 업데이트 공식에 기반한다.

f(x)의 **도함수를 구하기 복잡한 경우 N-R Method이 비효율적인 것을 보완하기 위해 N-R Method의 업데이트 공식에서의 f′(x)을 approximation 한다.**

$$f'(x) = \frac{f(x + \delta x_i) - f(x_i)}{\delta x_i}$$

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} = x_i - \frac{f(x_i)\delta x_i}{f(x + \delta x_i) - f(x)}$$

- Secant method와 비교해서 **최근 추정치를 사용하지 않고,** $x_i, \delta x$**를 사용**한다는 점이 다르다.

4. Fixed point iteration

f(x) = 0 꼴의 문제를 x = g(x)의 형태로 변환하여 x = g(x)를 만족하는 Root x를 찾는 방법

$$x_{k+1} = g(x_k)$$

- 위 수식대로 Iteration된다.

과정

a. x_i를 이용하여 $g(x_i)$를 얻는다.

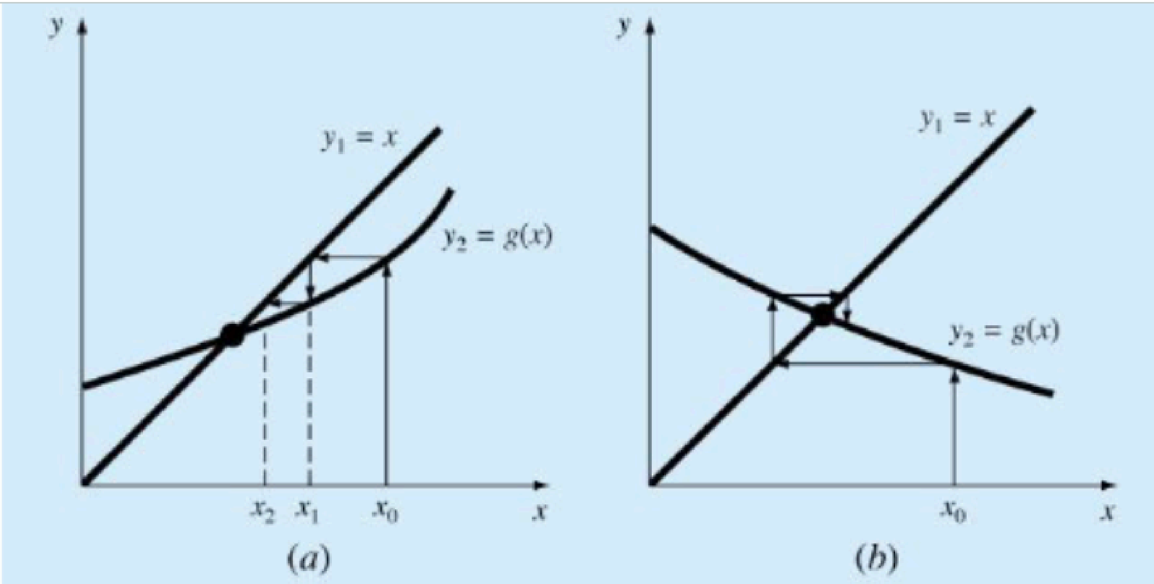
b. $g(x_i)$를 다음 x_{i+1}로 사용한다.

특징

- **Contraction mapping이면 수렴한다.**
  - **Contraction mapping**: 각 Iteration마다 $\mid g(p) - g(x_{k+1}) \mid \leq L \cdot \mid p - x_k \mid$ (L은 0보다 크고 1보다 작은 실수)

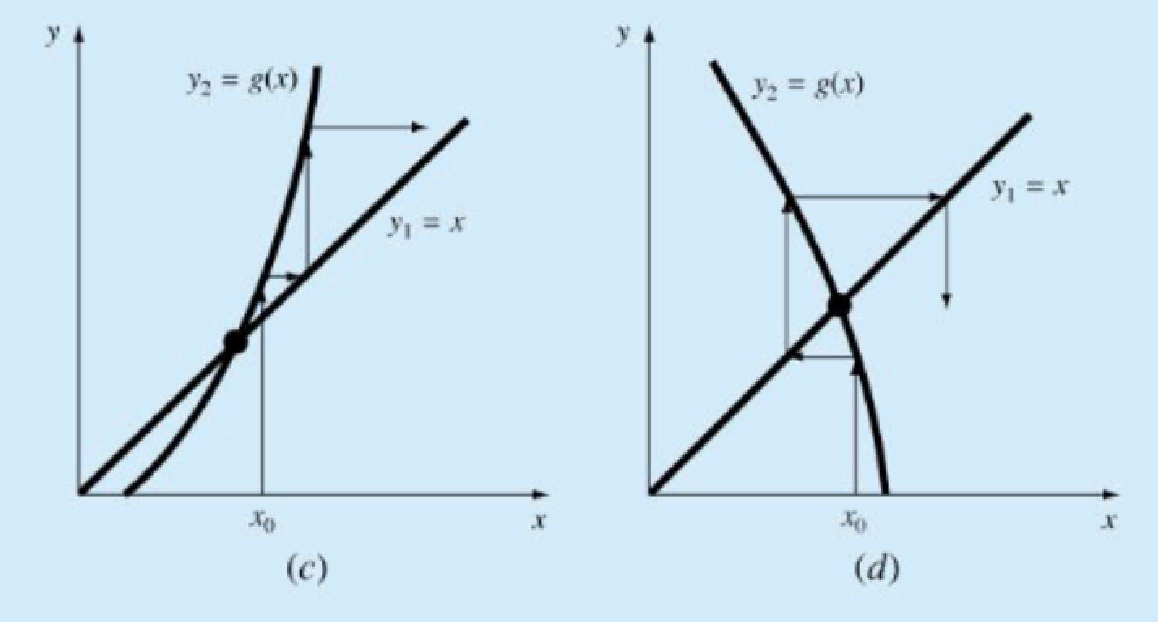
수렴성

수렴하는 경우



- (a)는 계단식 수렴, (b)는 나선형 수렴

수렴하지 않는 경우



- (c)는 계단식 발산, (d)는 나선형 발산

5. Muller method

Secant method를 일반화한 방법으로, Secant method와 달리 최근 세 개의 추정값을 이용하여 곡선을 근사하는 방법

$$p_{n+3} = p_{n+2} - \frac{2c}{b + sign(b)\sqrt{b^2 - 4ac}}$$

$$c = f(p_2),$$

$$b = \frac{(p_0 - p_2)^2[f(p_1) - f(p_2)] - (p_1 - p_2)^2[f(p_0) - f(p_2)]}{(p_0 - p_2)(p_1 - p_2)(p_0 - p_1)},$$

$$a = \frac{(p_1 - p_2)[f(p_0) - f(p_2)] - (p_0 - p_2)[f(p_1) - f(p_2)]}{(p_0 - p_2)(p_1 - p_2)(p_0 - p_1)}.$$

위 공식과 a, b, c를 얻는 과정은 다음과 같다.

1. 우리가 근사하고자 하는 함수를 $f(x) = ax^2 + bx + c$ 라고 하자.
    a. 위 함수는 p_{n+2}, p_{n+1}, p_n을 통과한다.
2. 1-a에 의해 함수 f(x)를 다음과 같이 작성할 수 있다.
    a. $f(x) = a(x - p_{n+2})^2 + b(x - p_{n+2}) + c$
3. 2-a의 f(x)는 p_{n+1}, p_n을 통과한다.
    a. $f(p_{n+1}) = at_1^2 + bt_1 + c$
    b. $f(p_n) = at_0^2 + bt_0 + c$
    c. $t_1 = p_{n+1} - p_{n+2}$
    d. $t_0 = p_n - p_{n+2}$
4. 3-a와 3-b의 연립방정식을 불면 a, b가 나오고, c도 나오게 된다.
5. (4)에서 얻은 a, b, c값을 이용하여 f(x)의 x절편을 찾으면 위 수식이 나온다.

과정

a. Initial value p_0, p_1, p_2가 주어진다.
b. 위 업데이트 공식에 맞게 p_{n+3}을 업데이트 한다.

특징

- 빠른 수렴 속도를 갖는다

Convergence speed

N-R Method > Secant method > Linear Interpolation method > Bisection method

---

Error analysis

1. Linear convergence

$$|p - p_{n+1}| <= M \cdot |p - p_n|$$

- p_i: 각 Itertion i에서의 추정치
- p: True solution
- M: 0 ~ 1 사이의 실수값

2. Quadratic convergence

$$|p - p_{n+1}| <= M \cdot |p - p_n|^2$$

- 직전 Iteration에서의 Error의 제곱에 비례하여 Error가 줄어든다.
- N-R Method의 수렴 속도가 빠른 이유

Accelerating convergence

Aitken's method

선형적으로 점점 p에 수렴하는 수열 $p_{n=0}^{\infty}$이 있을 때, 아래 수식을 만족하는 수열 $q_{n=0}^{\infty}$은 더 빠르게 p로 수렴한다.

$$q_n = p_n - \frac{(p_{n+1} - p_n)^2}{p_{n+2} - 2p_{n+1} + p_n}$$

증명

$\frac{(p_{n+1} - p_n)^2}{p_{n+2} - 2p_{n+1} + p_n}$을 생각해보자.

- 분모 $p_{n+2} - 2p_{n+1} + p_n$은 $(p_{n+2} - p_{n+1}) - (p_{n+1} - p_n)$이 된다.

- $(p_{n+2}-p_{n+1})-(p_{n+1}-p_n)$은 (n+1)번째 Iteration과 n번째 Iteration의 p 변화량 차이이다.
- 분자 $(p_{n+1}-p_n)^2$은
    - n번째 Iteration의 p 변화량의 제곱이다.

P가 선형 수렴으로 가정 되어있기 때문에, $|p-p_{n+1}| = M \cdot |p-p_n|$ 으로 나타낼 수 있다.

$e_{n+1} = |p-p_{n+1}|, e_n = |p-p_n|$ 이라고 하면 $e_{n+2} = Me_{n+1} = M^2 e_n$이라는 공식이 만들어진다.

- 분모는 결국 $(p+e_{n+2}-p-e_{n+1})-(p+e_{n+1}-p-e_n) = (e_{n+2}-e_{n+1})-(e_{n+1}-e_n)$이 된다.
    - e_n으로 정리하면 $M^2e_n - Me_n - Me_n + e_n = (M-1)^2 e_n$이 된다.
- 분자 역시 $(e_{n+1}-e_n)^2 = (M-1)^2 e_n^2$이 된다.

결국 아래와 같이 정리된다.

$$\frac{(p_{n+1}-p_n)^2}{p_{n+2}-2p_{n+1}+p_n} = \frac{(M-1)^2 e_n^2}{(M-1)^2 e_n} = e_n$$

결론적으로 아래 형태를 갖는다.

$$q_n = p_n - \frac{(p_{n+1}-p_n)^2}{p_{n+2}-2p_{n+1}+p_n} = p_n - e_n = p_n - (p_n - p) = p$$

- $q_n = p_n - e_n$으로 근사되고 Aitken's method를 통해 많은 step을 거치지 않아도 바로 solution이 될 가능성이 높아진다.

Fail-safe methods

Open method에서 **Divergence의 위험이 있기에 Convergence를 보장하기 위해 추가로 제안되는 함수**들

1. Conbination of Newton and Bisection

a. Bracket을 설정한다.
b. N-R method를 적용해 x_{i+1}을 찾는다.
c. (b)에서 **찾은 x_{i+1}가 (a)의 Bracket 내에 존재하고, N-R method에 의해 된 Update 정도가 작은지** 확인한다.
d. (c)의 조건을 만족하면 **Bisection method의 방식으로 업데이트**한다.

2. Ridder's method

a. Bracket을 설정한다.
b. p_n, p_{n+1}에 대해 $p_{n+2} = \frac{p_{n+1}+p_n}{2}$로 잡는다.
c. 지수함수를 가정하고 p_{n+2}, p_{n+1}, p_n을 이용하여 해당 지수함수를 찾는다.
d. (c)의 지수함수의 x절편을 p_{n+3}이라고 한다.
e. p_{n+3}, p_{n+1}, p_n를 이용하여 Bisection method의 업데이트 방식대로 업데이트한다.

위 두 방법은 Bracketing method에 속한다.