수치해석
(2019학년도 1학기)

[5주/1차시 학습내용]: False Position (가위치법)

Prof. Sang-Chul Kim

국민대학교 소프트웨어학부

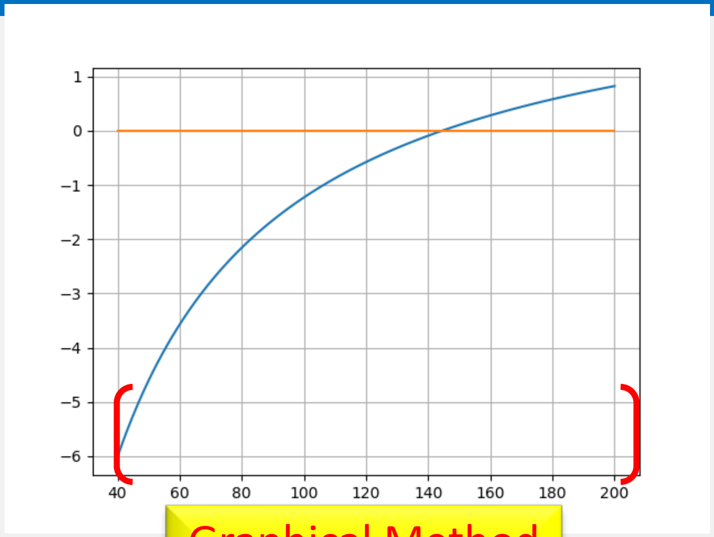
수치 알고리즘의 근사화 전략

- trial and error의 반복을 통해서 참 값에 근사한다.
- Bracketing Method (구간법)
  - Graphical Method : 구간에 대한 정의 탄생
  - 증분법 : 증분점에 대한 fitting optimization 필요
  - 이분법 : 구간을 찾아내는 것이 아니라, 근을 찾아내려고 노력함
  - 가위치법 : 이분법과 같이 근을 찾아내려고 노력함
    - 가위치법은 선형보간법 (Linea Interpolation) 이라고도 함
- Open Method (개방법)
  - 구간법에서 꼭 필요한 구간을 더 이상 사용하지 않음
  - 어떤 점도 근이 될 수 있음
  - 구간법보다 빠르게 근을 찾는 알고리즘을 제공함

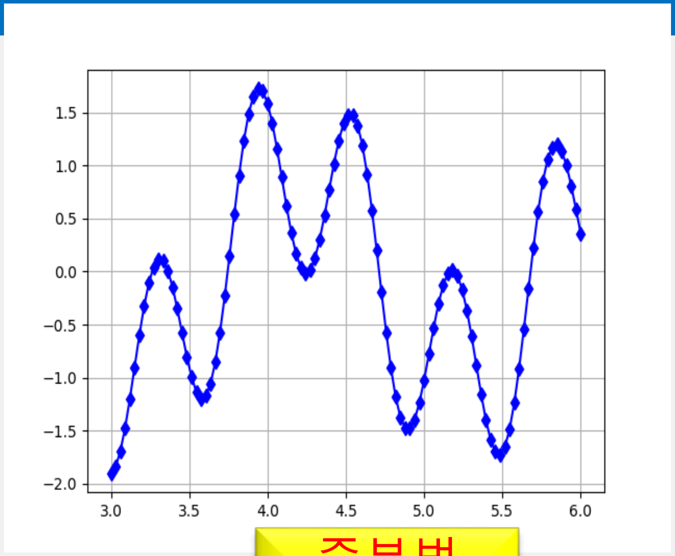
가위치법 알고리즘

- 1: x_l 과 x_u 의 구간을 정한다
  - $x_l = 40$, $x_u = 200$
- 2: $f(x_l)$의 값과 $f(x_u)$의 값의 부호가 변하면 구간으로 정한다
- 3: 가위치법에서는 구간이 정해지면, $x_r = x_u - \frac{f(x_u)(x_l - x_u)}{f(x_l) - f(x_u)}$ 을 새로운 근이라고 정한다
- 4. $f(x_l)$의 값과 $f(x_r)$의 값의 부호 변화를 체크한다
  - 변하지 않으면, $x_l = x_r$로 대체함
  - 변하면, $x_u = x_r$로 대체함
- 5. 단계 1로 간다

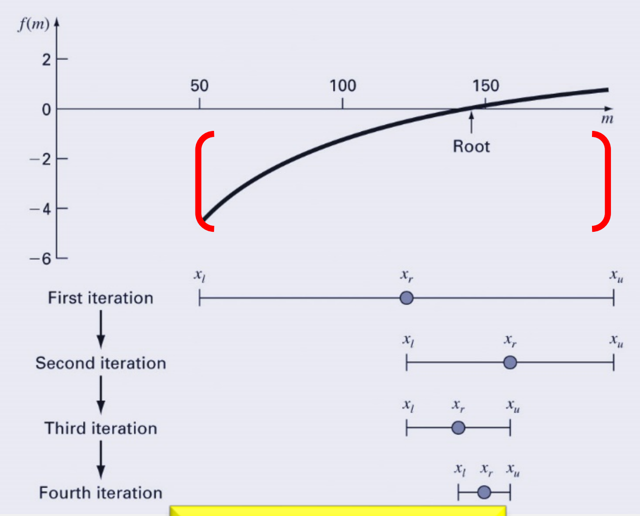
구간법(Bracketing)



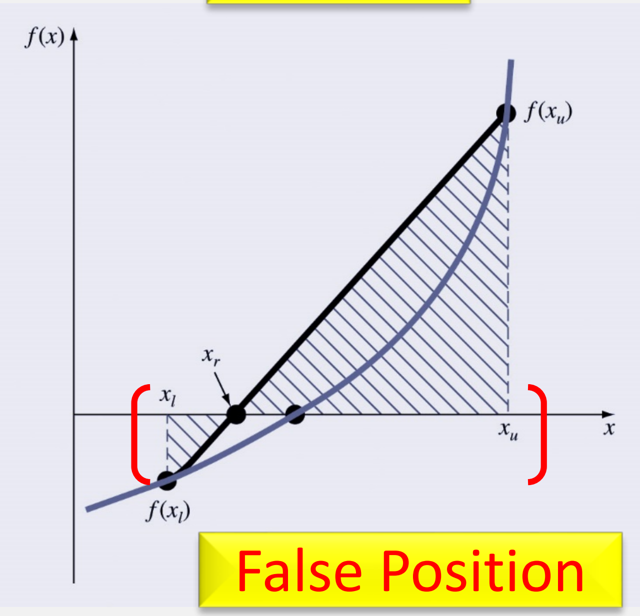
Graphical Method



증분법



Bisection

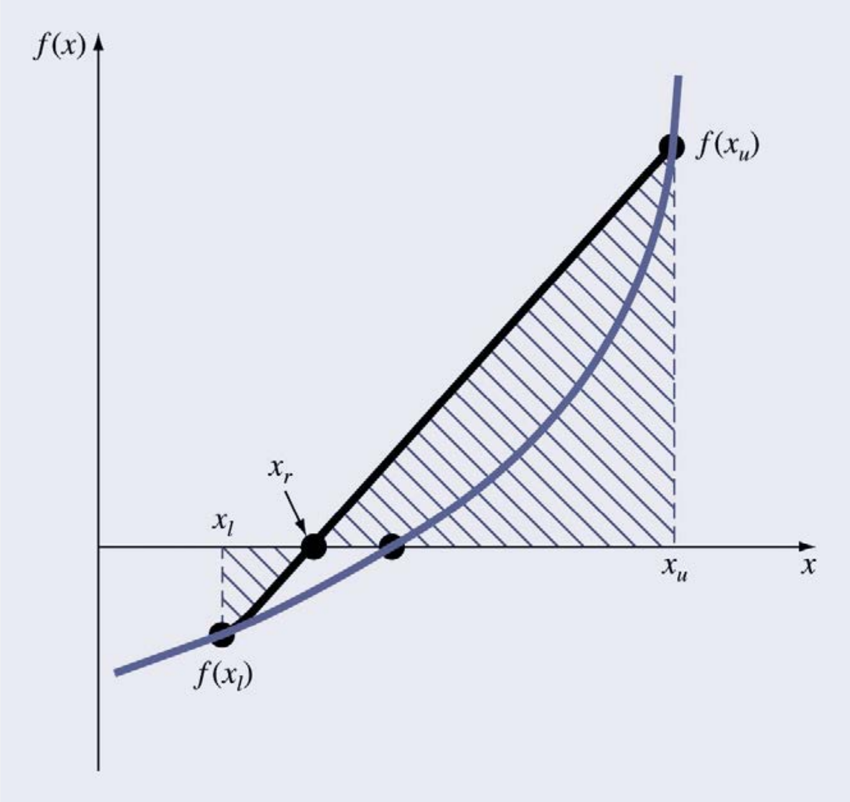


False Position

False Position (가위치법)

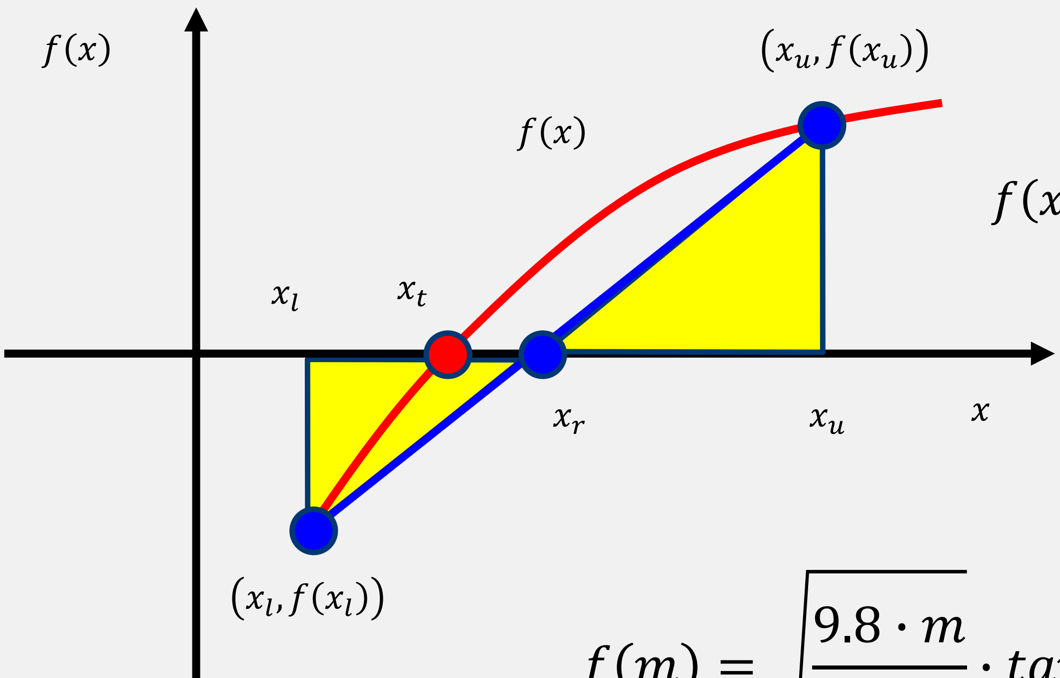
- **It determines the next guess not by splitting the bracket in half but by connecting the endpoints with a straight line and determining the location of the intercept of the straight line (x_r).**

$$x_r = x_u - \frac{f(x_u)(x_l - x_u)}{f(x_l) - f(x_u)}$$



False Position (가위치법)

- f(m)=0을 만족하는 m을 구하기가 쉽지 않다.
- 수치해석에서는 근사값을 이용하여 근을 구한다.

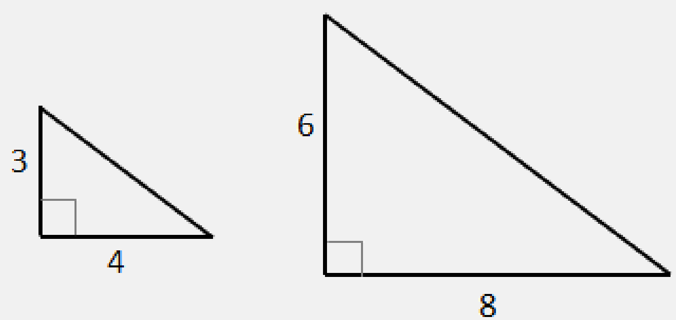


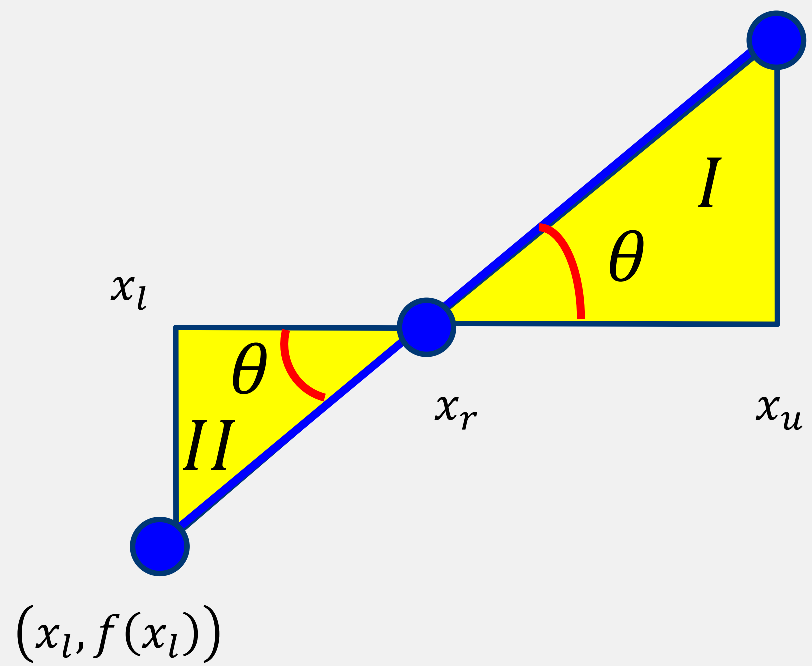
$$f(x) = \sqrt{x \cdot c_1} \cdot tanh\left(\sqrt{\frac{1}{x} \cdot c_2}\right) - c_3$$

$$f(m) = \sqrt{\frac{9.8 \cdot m}{0.25}} \cdot tanh\left(\sqrt{\frac{9.8 \cdot 0.25}{m}} \cdot 4\right) - 36$$

Similarity in Triangles (닮은꼴 삼각형)

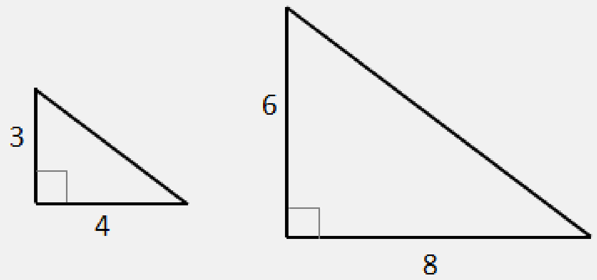
- Similarity in Triangles (닮은꼴 삼각형)을 이용하여 가위치법을 설명한다.
- Triangles I and II are Similar
- Triangle I's $Height/Base$ = Triangle II's $Height/Base$

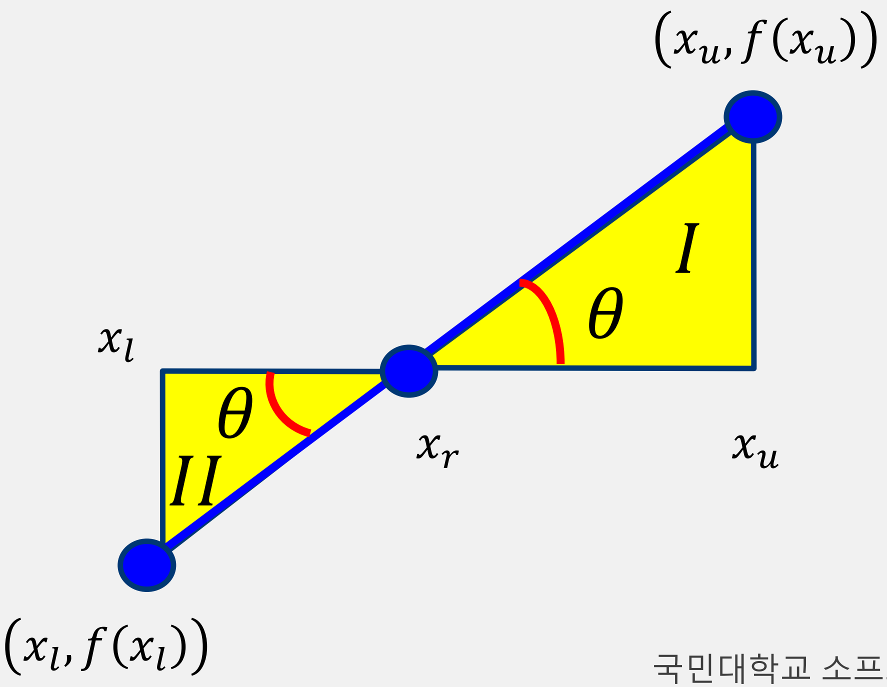


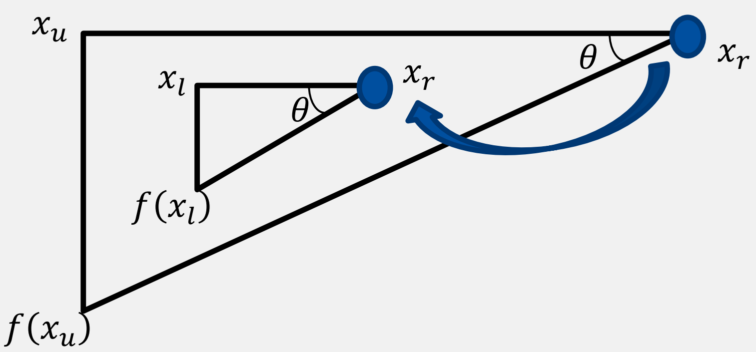


Similarity in Triangles (닮은꼴 삼각형)

- Similarity in Triangles (닮은꼴 삼각형)을 이용하여 가위치법을 설명한다.
- $$\frac{f(x_u)-0}{x_r-x_u}=\frac{f(x_l)-0}{x_r-x_l}$$







Calculate x_r

$$\boxed{\frac{f(x_u) - 0}{x_r - x_u} = \frac{f(x_l) - 0}{x_r - x_l}}$$

$$f(x_l) \cdot (x_r - x_u) = f(x_u) \cdot (x_r - x_l)$$

$$f(x_l) \cdot x_r - f(x_l) \cdot x_u = f(x_u) \cdot x_r - f(x_u) \cdot x_l$$

$$f(x_l) \cdot x_r - f(x_u) \cdot x_r = f(x_l) \cdot x_u - f(x_u) \cdot x_l$$

$$\left(f(x_l) - f(x_u)\right) \cdot x_r = x_u \cdot f(x_l) - x_l \cdot f(x_u)$$

Calculate x_r

$$x_r = x_u - \frac{f(x_u)(x_l - x_u)}{f(x_l) - f(x_u)}$$

- 최종식인 왼쪽과 같은 식 형태로 만들기 위해 x_u를 더하고 뺀다

$$\left(f(x_l) - f(x_u)\right) \cdot x_r = x_u \cdot f(x_l) - x_l \cdot f(x_u)$$

$$x_r = \frac{x_u \cdot f(x_l)}{f(x_l) - f(x_u)} - \frac{x_l \cdot f(x_u)}{f(x_l) - f(x_u)}$$

$$x_r = x_u + \frac{x_u \cdot f(x_l)}{f(x_l) - f(x_u)} - x_u - \frac{x_l \cdot f(x_u)}{f(x_l) - f(x_u)}$$

Calculate x_r

$$x_r = x_u + \frac{x_u \cdot f(x_l)}{f(x_l) - f(x_u)} - x_u - \frac{x_l \cdot f(x_u)}{f(x_l) - f(x_u)}$$

$$x_r = x_u + \frac{x_u \cdot f(x_l) - x_u \cdot f(x_l) + x_u \cdot f(x_u)}{f(x_l) - f(x_u)} - \frac{x_l \cdot f(x_u)}{f(x_l) - f(x_u)}$$

$$x_r = x_u + \frac{x_u \cdot f(x_u)}{f(x_l) - f(x_u)} - \frac{x_l \cdot f(x_u)}{f(x_l) - f(x_u)}$$

Calculate x_r

$$x_r = x_u + \frac{x_u \cdot f(x_u)}{f(x_l) - f(x_u)} - \frac{x_l \cdot f(x_u)}{f(x_l) - f(x_u)}$$

$$x_r = x_u + \frac{x_u \cdot f(x_u) - x_l \cdot f(x_u)}{f(x_l) - f(x_u)}$$

$$x_r = x_u - \frac{f(x_u) \cdot (x_l - x_u)}{f(x_l) - f(x_u)}$$

$$\boxed{x_r = x_u - \frac{f(x_u) \cdot (x_l - x_u)}{f(x_l) - f(x_u)}}$$

코딩

- 가위치법
- https://github.com/SCKIMOSU/Numerical-Analysis/blob/master/false.py

```python
import numpy as np

def falseposition(func, xl, xu):
    maxit=100
    es=1.0e-4

    test=func(xl)*func(xu)

    if test > 0:
        print('no sign change')
        return [], [], [], []

    iter=0
    xr=xl

    ea=100

    while(1):
        xrold=xr
        #xr=np.float((xl+xu)/2)
        xr = np.float(xu-func(xu)*(xl-xu)/(func(xl)-func(xu)))
        iter=iter+1
        if xr != 0:
            ea=np.float(np.abs(np.float(xr)-np.float(xrold))/np.float(xr))*100
        test=func(xl)*func(xr)

        if test > 0:
            xl=xr
        elif test < 0:
            xu=xr
        else:
            ea=0

        if np.int(ea<=es) | np.int(iter>=maxit):
            break
    root=xr
    fx=func(xr)

    return root, fx, ea, iter

if __name__ == '__main__':

    fm=lambda m: np.sqrt(9.81*m/0.25)*np.tanh(np.sqrt(9.81*0.25/m)*4)-36
    root, fx, ea, iter=falseposition(fm, 40, 200)
    print('root=', root)
    print('f(root)=', fx)
    print('ea=', ea)
    print('iter=', iter)
```

Results of False Position

```
if __name__=='__main__':
    fm=lambda m: np.sqrt(9.81*m/0.25)*np.tanh(np.sqrt(9.81*0.25/m)*4)-36
    root, fx, ea, iter=falseposition(fm, 40, 200)
    print('root=', root)
    print('f(root)=', fx)
    print('ea=', ea)
    print('iter=', iter)
```

- root = 142.73783844758216 (False Position)
- f(root) = 4.20034974979e-06 (must be zero, False Position)
- estimated error= 7.78101379774408 8e-05 (must be zero error, False Position)
- iterated number to find root = 29 (False Position)

이분법과 가위치법의 비교

- False position has more error and more iteration than Bisection
- root = 142.73765563964844 (Bisection)
- root = 142.73783844758216 (False Position)
- f(root) = 4.60891335763e-07 (must be zero, Bisection)
- f(root) = 4.20034974979e-06 (must be zero, False Position)
- estimated error= 5.3450468252827136e-05 (must be zero error, Bisection)
- estimated error= 7.781013797744088e-05 (must be zero error, False Position)
- iterated number to find root = 21 (Bisection)
- iterated number to find root = 29 (False Position)
- 가위치법이 이분법에 비해 반복횟수를 많이 가짐에도 불구하고, 에러가 크다. 이는 왜? 실제 미분을 사용하지 않기 때문

구간법 알고리즘의 특징

- Graphical Method와 증분법의 공통점은 $f(x) = 0$ 을 지나는 **근사근 구간 (estimated root interval)** 을 찾아 준다
- 이분법과 가위치법의 공통점은 $f(x) = 0$ 이 되는 **근사근 (estimated root)**을 찾아 준다
- **Graphical Method, 증분법, 이분법, 가위치법**, 모두의 공통점은 $f(x) = 0$ 을 지나는 근사근을 구하기 위해 구간을 사용한다는 것이다.
- 가위치법은 구간을 상하직선으로 나누는 이분법에 비해 사선으로 구간을 나눈다.
- 다음에 나오는 개방법에서는 사선에서 발전된 미분법이 소개되는 데, 가위치법은 이러한 면에서 구간법에서 미분법을 소개하는 알고리즘에 가깝다고 얘기할 수 있다.