수치해석
(2019학년도 1학기)

[6주/2차시 학습내용]: Newton-Raphson 코딩 및 에러 계산

국민대학교 소프트웨어학부

Ch. 6. Open Methods
Coding for Newton-Raphson

Problem of significant vertebrae injury if the free-fall velocity exceeds 36 m/s after 4 s of free fall.

Coding for Newton-Raphson

```
import math
import numpy as np

def sech(x):
    return np.cosh(x)**(-1)
def newton_raphson(func, dfunc, xr):
    maxit=50
    es=1.0e-5
    iter=0
    while(1):
        xrold=xr
        xr=np.float(xr-func(xr)/dfunc(xr))
        iter=iter+1
        if xr != 0:
            ea=np.float(np.abs((np.float(xr)-np.float(xrold))/np.float(xr))*100)
        if np.int(ea <= es) | np.int(iter >= maxit):
            break
    root=xr
    fx=func(xr)
    return root, fx, ea, iter

if __name__ == '__main__':
    g=9.81; cd=0.25; v=36; t=4;
    fp=lambda m: np.sqrt(g*m/cd)*np.tanh(np.sqrt(g*cd/m)*t)-v
    dfp=lambda m: (1/2)*np.sqrt(9.81/(m*0.25))*np.tanh(np.sqrt(9.81*0.25/m)*4)-
9.81*4/(2*m)*(sech(np.sqrt(9.81*0.25/m)*4))**2
    root, fx, ea, iter=newton_raphson(fp, dfp, 140)
    print('root weight= ', root)
    print('f(root weight, should be zero) =', fx)
    print('ea = should be less than 1.0e-4', ea)
    print('iter =', iter)
```

Console Debugging

$$f'(m) = \frac{1}{2} \cdot \sqrt{\frac{g}{mc_d}} \cdot tanh\left(\sqrt{\frac{gc_d}{m}} \cdot t\right) - \frac{gt}{2m} \cdot sech^2\left(\sqrt{\frac{gc_d}{m}} \cdot t\right)$$

$$f(m) = \sqrt{\frac{gm}{c_d}} \cdot tanh\left(\sqrt{\frac{gc_d}{m}} \cdot t\right) - 36$$

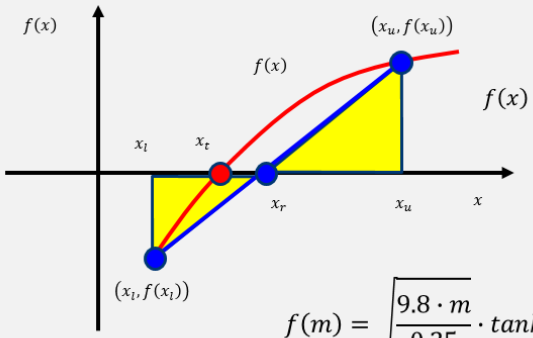
```
g=9.81; cd=0.25; t=4; v=36
import numpy as np
m=200
f
f1=0.5*np.sqrt(g/(m*cd))*np.tanh(np.sqrt(g*vd/m)*t)-g*t/(2*m)*(np.cosh(np.sqrt(g*cd/m)*t))**(-2)
f1
xrnew=200-f/f1
m=xrnew
f=np.sqrt(g*m/cd)*np.tanh(np.sqrt(g*cd/m)*t)-v
f1=0.5*np.sqrt(g/(m*cd))*np.tanh(np.sqrt(g*cd/m)*t)-g*t/(2*m)*(np.cosh(np.sqrt(g*cd/m)*t))**(-2)
xrnew=122.03-f/f1
xrnew
```

root weight= 142.7376189663234
f(root weight, should be zero) = -2.8928707251907326e-07
ea = should be less than 1.0e-4 9.907775669827273e-06
iter = 3

가위치법

False Position (가위치법)

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



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

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

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.

가위치법과 Newton-Raphson 방법

- 가위치법은 두 점을 지나는 직선의 개념을 준 알고리즘으로 역할을 톡톡히 했다.
- Newton-Raphson 방법은 미분법으로 접선의 기울기를 이용함으로 직선을 사용하는 가위치법보다 빠르게 근사해에 접근했다.
- 미분의 위력을 실감하는 방법이 바로 Newton-Raphson 방법이다
- 미분을 사용하면 연산반복횟수를 줄이는 장점이 있다.

root weight= 142.7376189663234
f(root weight, should be zero) = -2.8928707251907326e-07
ea = should be less than 1.0e-4 9.907775669827273e-06
iter = 3

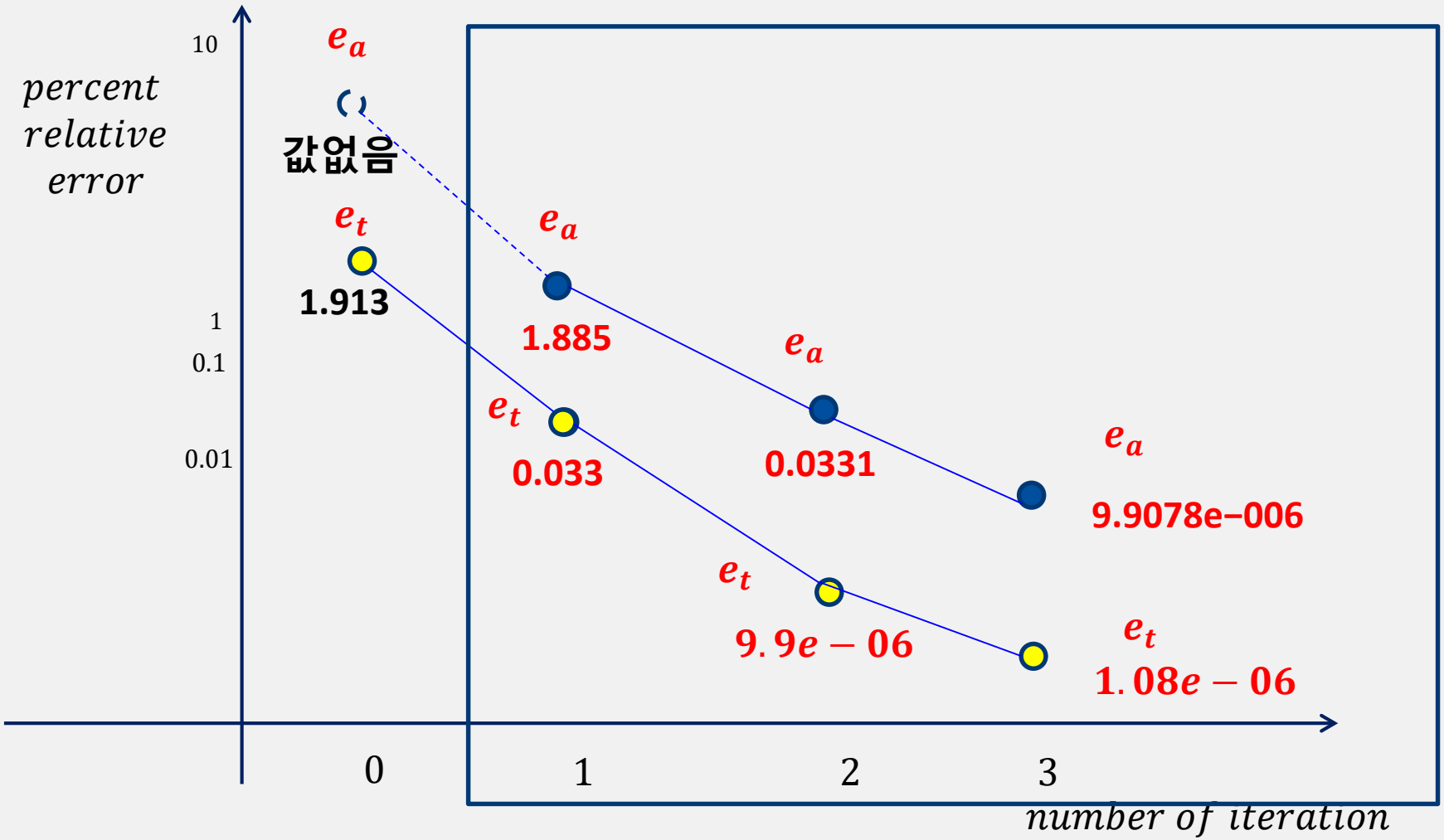
Get the real root

```
import numpy as np
import math
from scipy.optimize import fsolve
fm=lambda m: np.sqrt(9.81*m/0.25)*np.tanh(np.sqrt(9.81*0.25/m)*4)-36
m=fsolve(fm, 1)
print("Real Root= ", m)
```

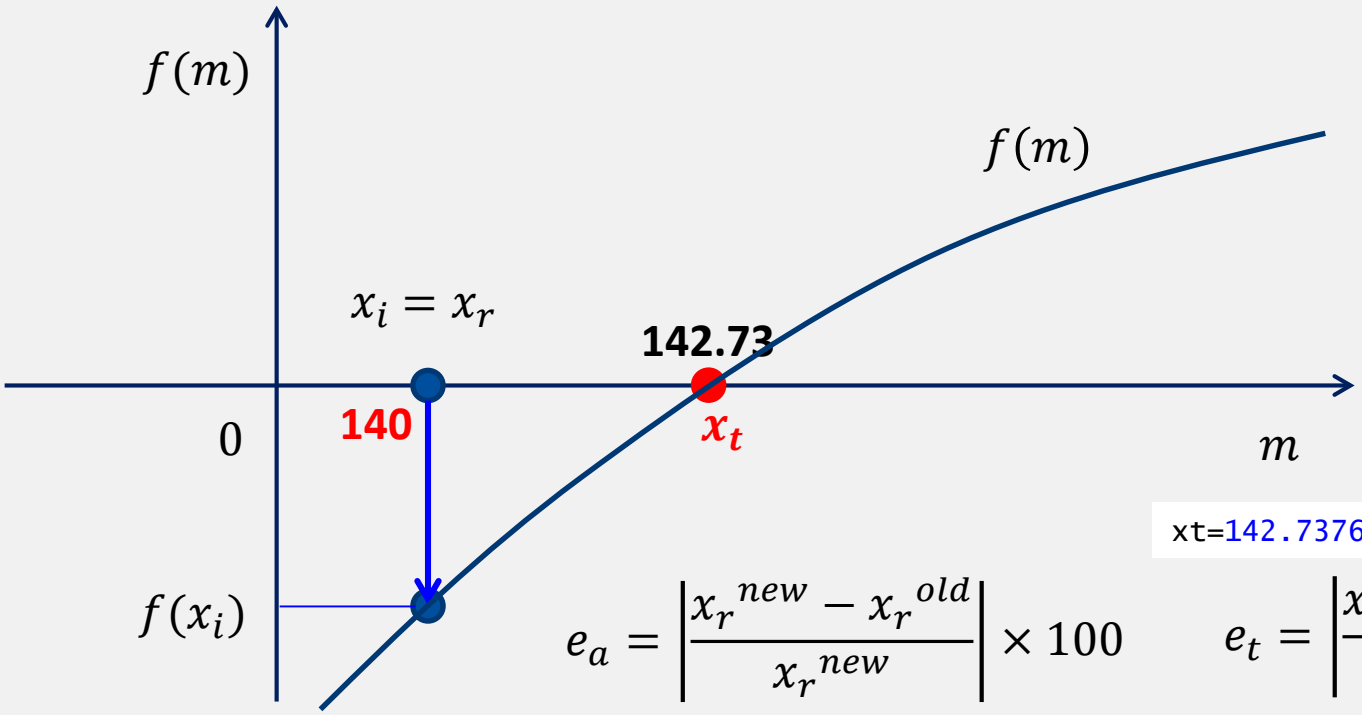
```
Real Root=  [142.73763311]
```

```
xt=142.73763311
```

e_a and e_t (상대 오차외 절대 오차)



Error 그래프 그리기 (iter=0)

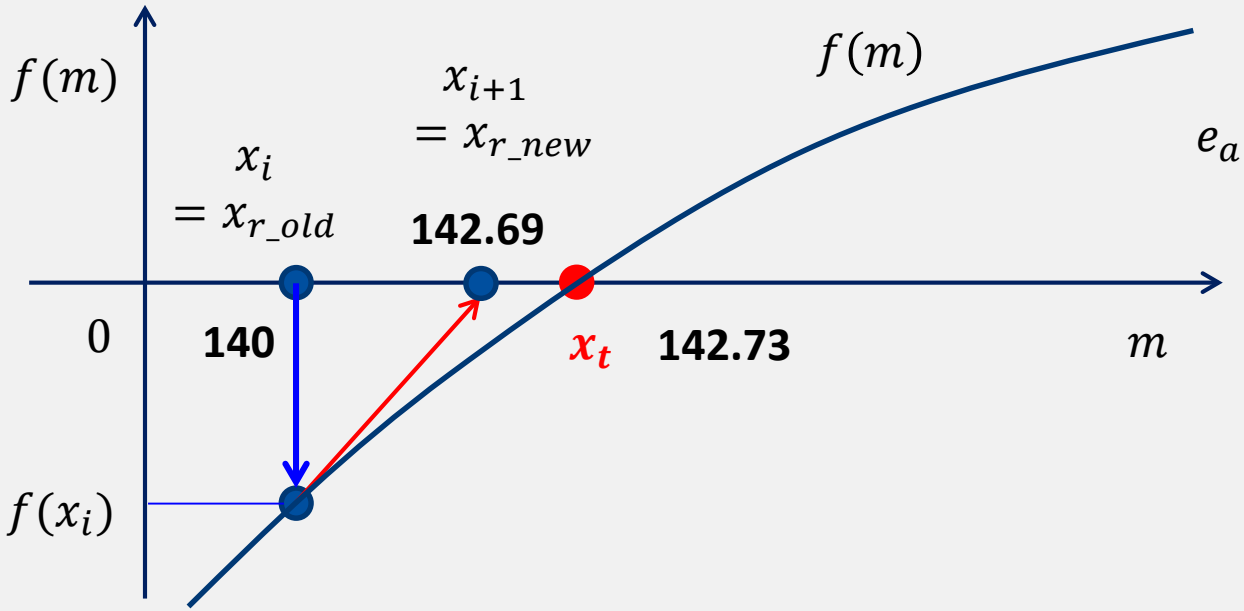



```
xt=142.73763311
```

$$e_a = \left|\frac{x_r{}^{new} - x_r{}^{old}}{x_r{}^{new}}\right| \times 100 \qquad e_t = \left|\frac{x_t - x_r}{x_t}\right| \times 100$$

xr_old	*xr_new*	*ea*	*et*
값 없음	**140**	$\left\|\frac{140-\text{없음}}{140}\right\| \times 100$=값없음	$\left\|\frac{x_t-140}{x_t}\right\| \times 100$=1.913

Error 그래프 그리기 (iter=1)



$$e_a = \left|\frac{x_r{}^{new} - x_r{}^{old}}{x_r{}^{new}}\right| \times 100$$

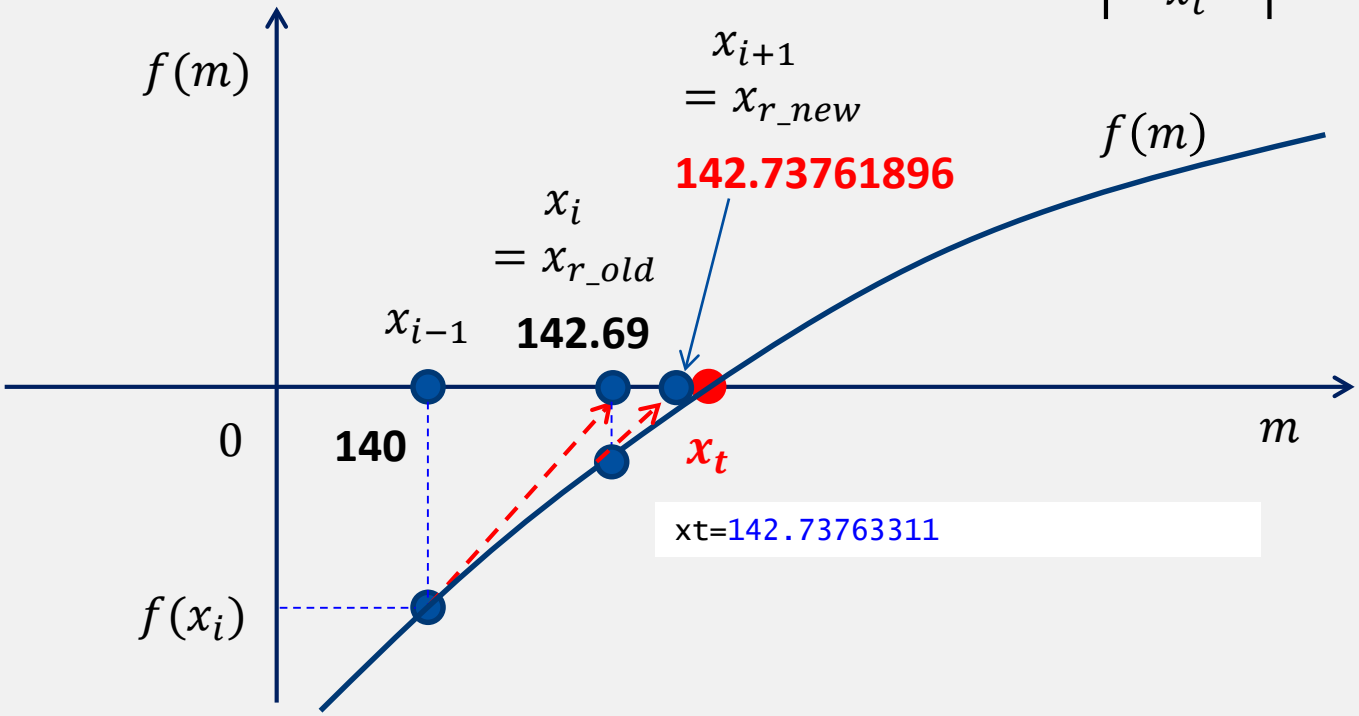
$$e_t = \left|\frac{x_t - x_r}{x_t}\right| \times 100$$

iter	xr_old	xr_new	ea	et
0	-	140	$\left\|\frac{140-\text{없음}}{140}\right\| \times 100$=값없음	$\left\|\frac{x_t-140}{x_t}\right\| \times 100$=1.913
1	140	142.6903	$\left\|\frac{142.6903-140}{142.6903}\right\| \times 100 = 1.885$	$\left\|\frac{x_t-142.6903}{x_t}\right\| \times 100 = 0.033$

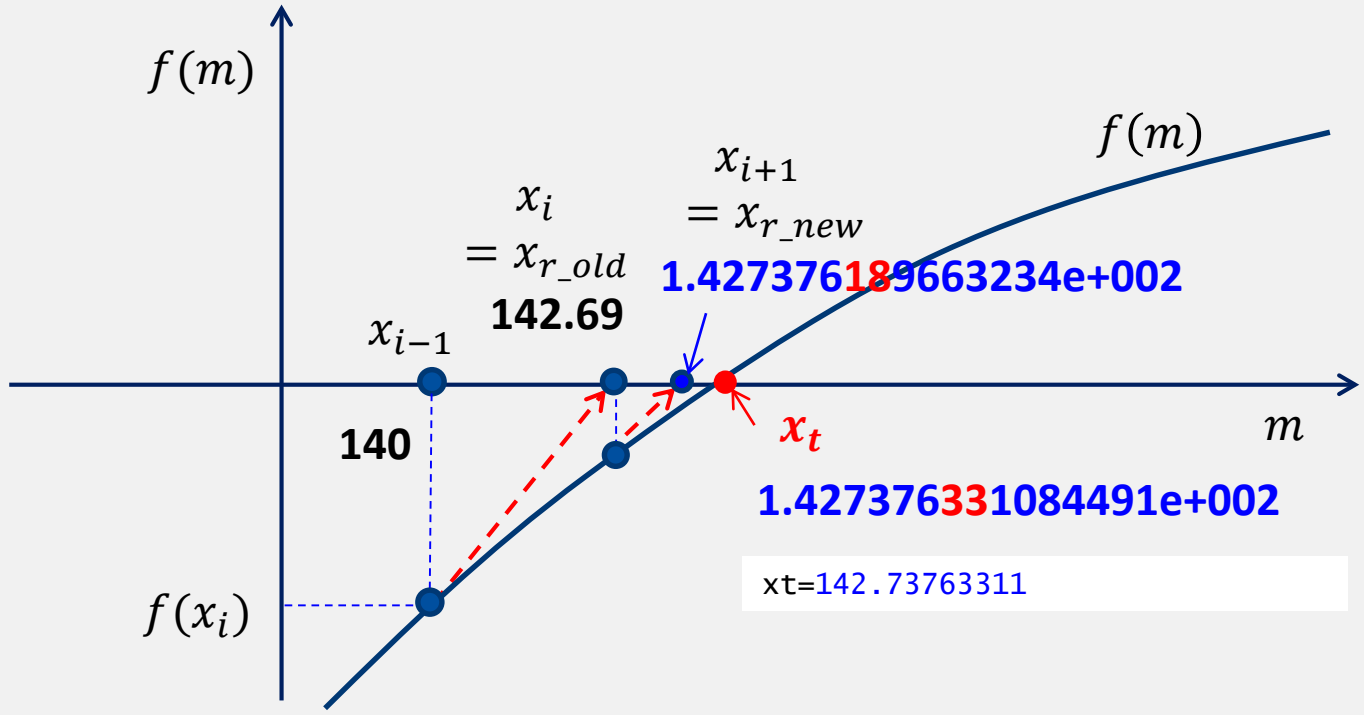
Error 그래프 그리기 (iter=2)

$$e_a = \left| \frac{{x_r}^{new} - {x_r}^{old}}{{x_r}^{new}} \right| \times 100$$

$$e_t = \left| \frac{x_t - x_r}{x_t} \right| \times 100$$

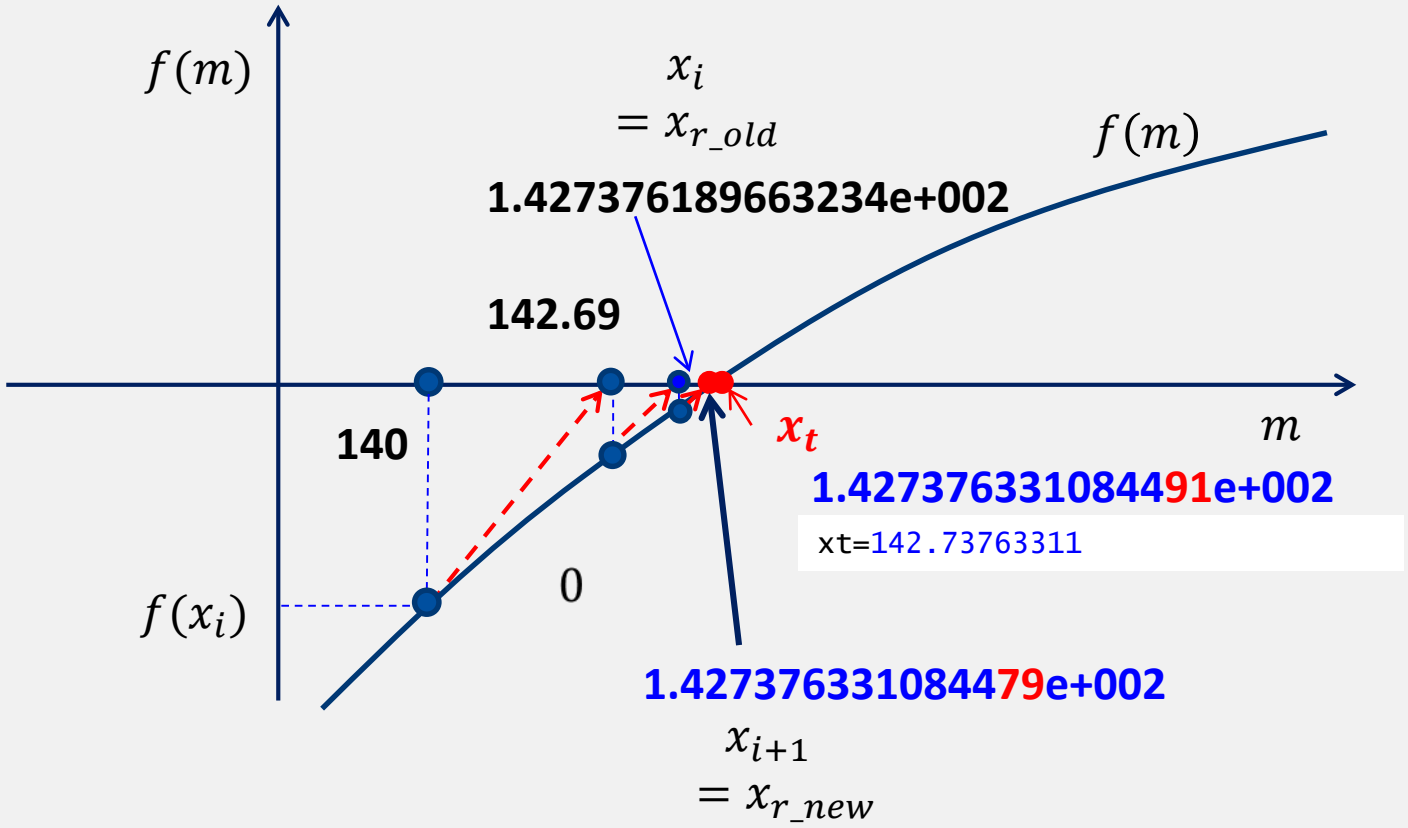


Error 그래프 그리기 (iter=2)



Error 그래프 그리기 (iter=3)

$$e_a = \left| \frac{{x_r}^{new} - {x_r}^{old}}{{x_r}^{new}} \right| \times 100 \qquad e_t = \left| \frac{x_t - x_r}{x_t} \right| \times 100$$



floating points format

```
'{:06.16f}'.format(xr)
'142.7376331084478807'
```

Padding numbers

Similar to strings numbers can also be constrained to a specific width.

Old
```
'%4d' % (42,)
```

New
```
'{:4d}'.format(42)
```

Output
```
  42
```

Again similar to truncating strings the precision for floating point numbers limits the number of positions after the decimal point.

For floating points the padding value represents the length of the complete output. In the example below we want our output to have at least 6 characters with 2 after the decimal point.

Old
```
'%06.2f' % (3.141592653589793,)
```

New
```
'{:06.2f}'.format(3.141592653589793)
```

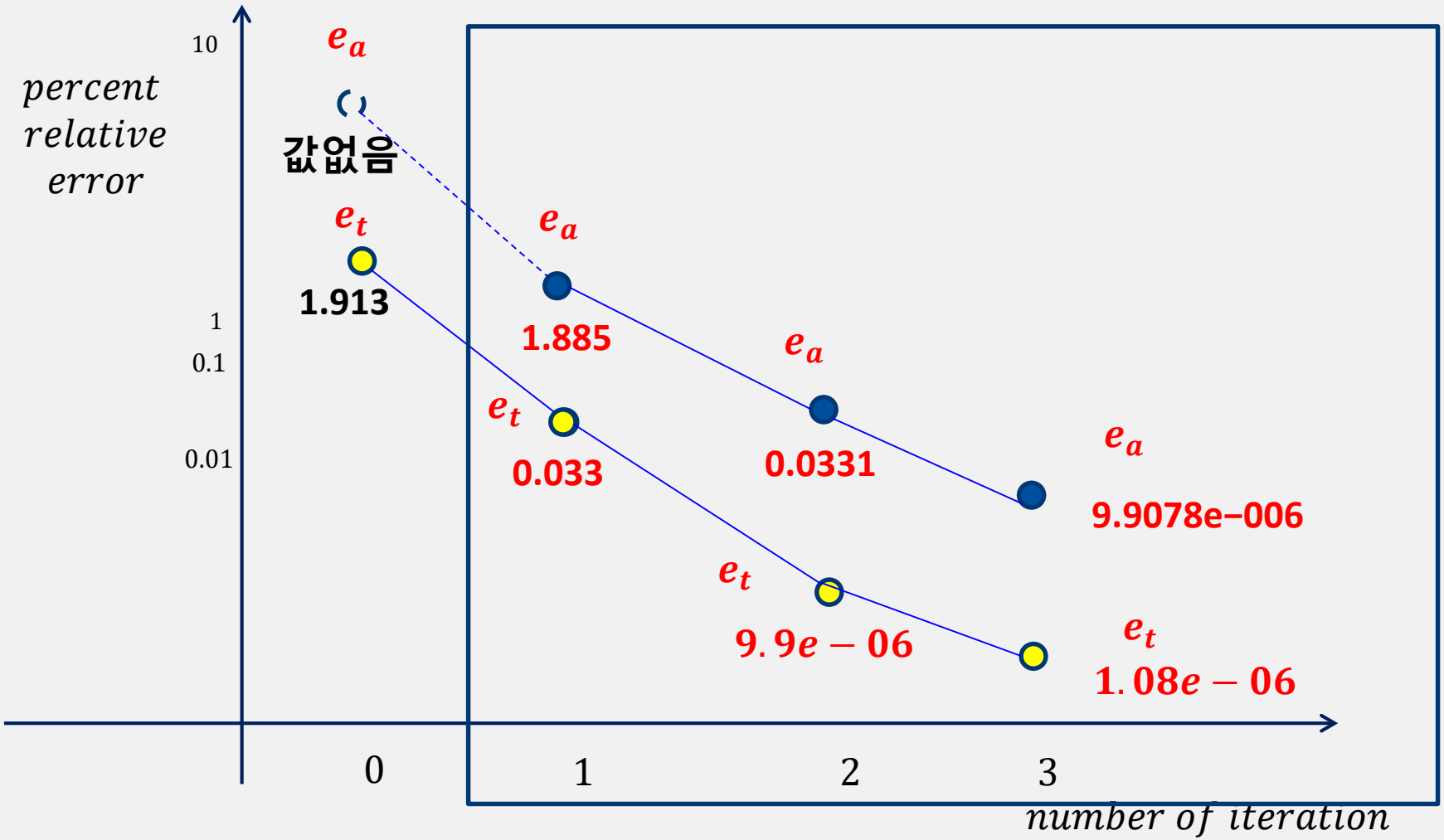
Output
```
003.14
```

Error 분석

x_t = 1.4273763310844 91e+002

iter	xr_old	xr_new	ea	et
0	-	140	$\left\lvert\frac{140-\text{없음}}{140}\right\rvert \times 100$=값없음	$\left\lvert\frac{x_t-140}{x_t}\right\rvert \times 100$=1.913
1	140	142.6903	$\left\lvert\frac{142.6903-140}{142.6903}\right\rvert \times 100 = 1.8854$	$\left\lvert\frac{x_t-142.6903}{x_t}\right\rvert \times 100 = 0.033$
2	142.6903	142.73761896	$\left\lvert\frac{142.73761896-142.6903}{142.73761896}\right\rvert \times 100 = 0.0331$	$\left\lvert\frac{x_t-142.7376}{x_t}\right\rvert \times 100 = 9.9e-06$
3	142.73761896	1.42737633108479e+002	$\left\lvert\frac{1.42737633108479e+002-1.42737618966323 4e+002}{1.42737633108479e+002}\right\rvert \times 100 = 9.9078e-006$	$\left\lvert\frac{x_t-1.42737633108479e+002}{x_t}\right\rvert \times 100 = 1.08e-06$

e_a and e_t


10
1
0.1
0.01
percent relative error
e_a
값없음
e_t
1.913
e_a
1.885
e_t
0.033
e_a
0.0331
e_a
9.9078e−006
e_t
9.9e − 06
e_t
1.08e − 06
0
1
2
3
number of iteration