수치해석
(2019학년도 1학기)

[7주/1차시 학습내용]: Ch.7 Optimization (최적화), 최적화 개념에 대해 학습한다.

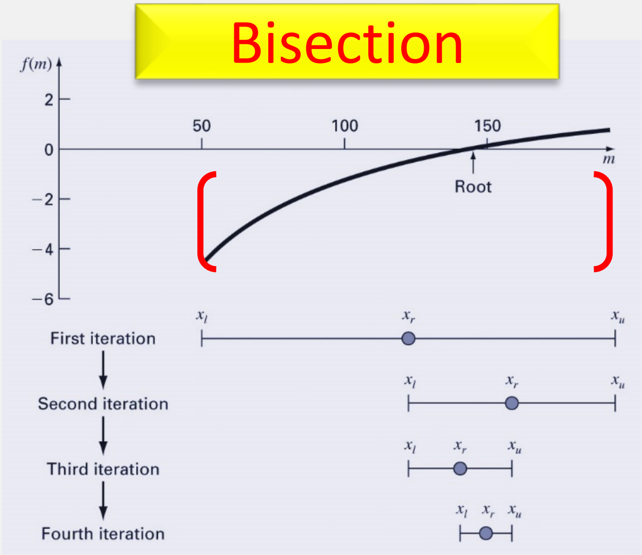
** 중간고사 시험범위는 1주차 1차시부터 7주차 1차시 (여기)까지

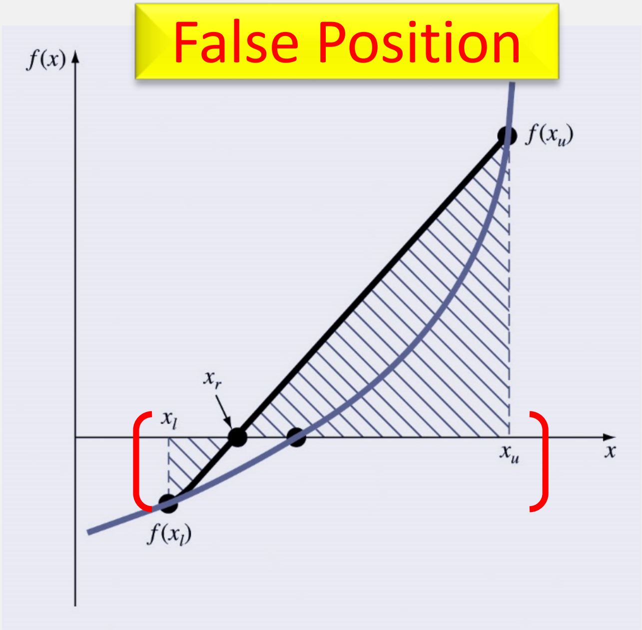
국민대학교 소프트웨어학부

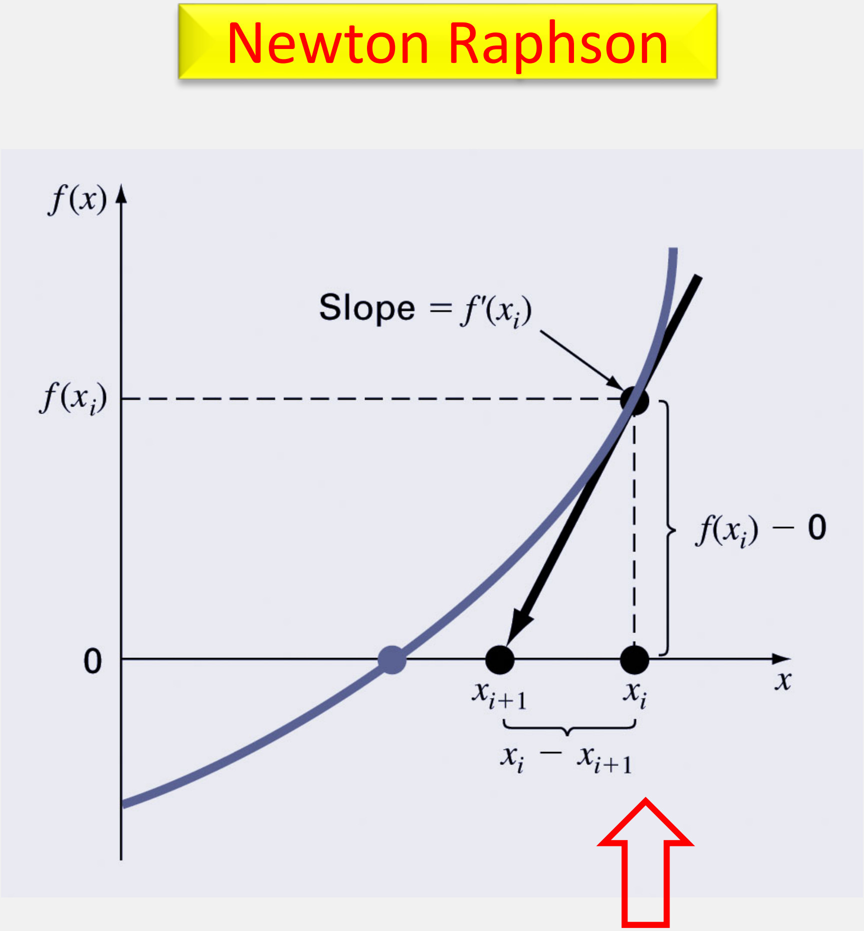
수치 알고리즘의 근사화 전략

- 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
  - Newton-Raphson, Secant Method, Modified Secant Method

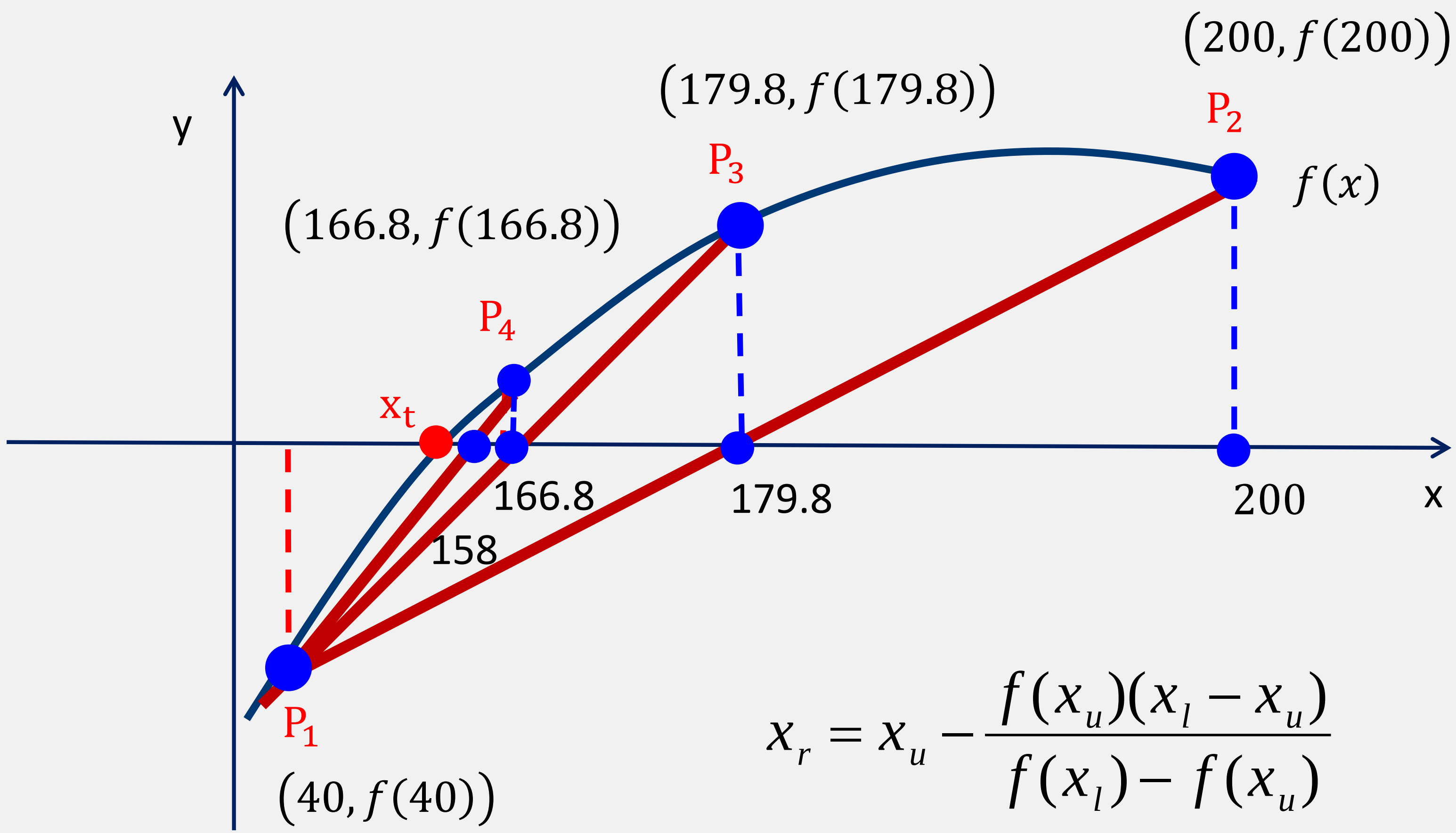
구간법(Bracketing)과 개방법(Open)







가위치법의 Trial and Error (시행착오)



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

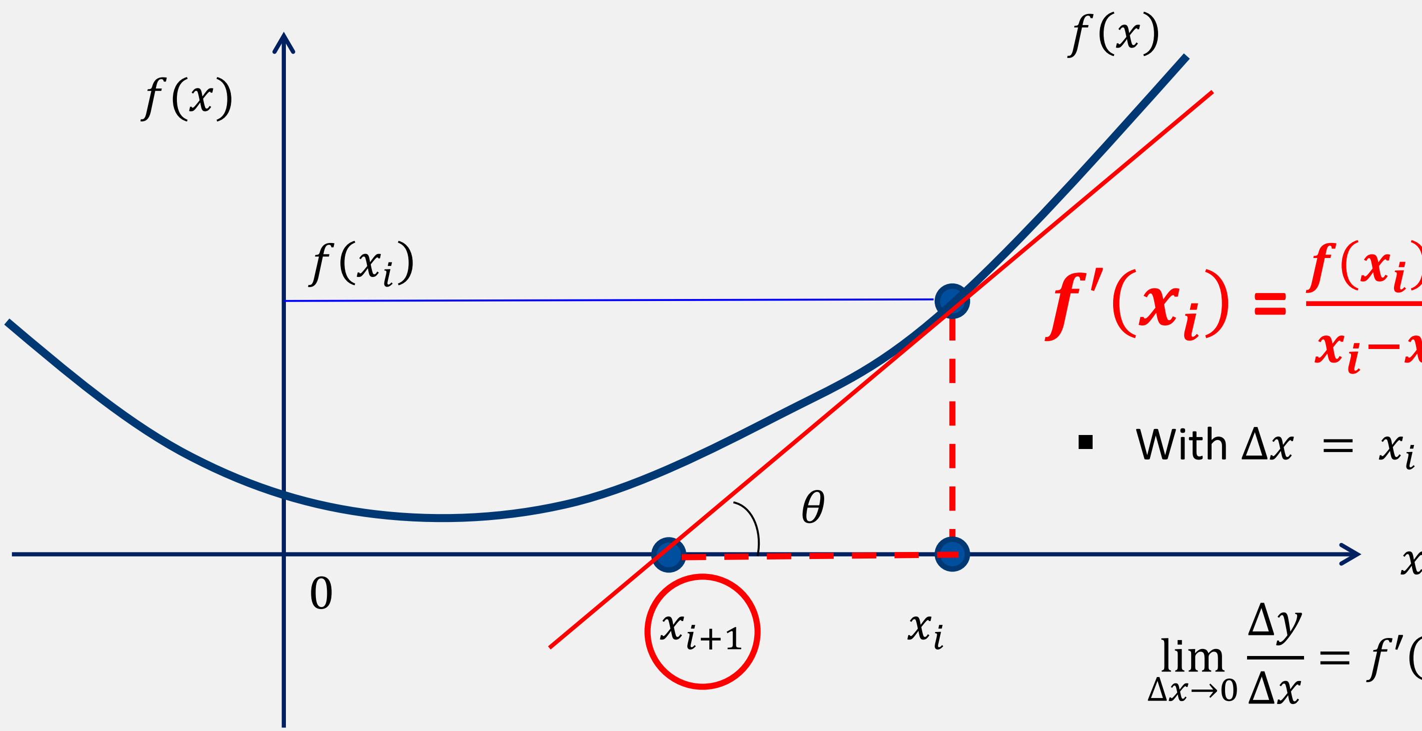
가위치법의 Trial and Error (시행착오)

- [40, 200] 구간에서 P1과 P2 점을 잇는 직선이 x축과 만나는 점인 179.8을 근사해로 인정하지 못하니, 179.8에서의 부호를 판단함.
  - → +로 부호가 변경됨 → upper bound를 200에서 179.8로 변경함
- 새로운 구간 [40, 179.8]이 생겼음. P1과 P3 점을 잇는 직선이 x축과 만나는 점인 166.8을 근사해로 인정하지 못하니, 166.8에서의 부호를 판단함.
  - → +로 부호가 변경됨 → upper bound를 179.8 에서 166.8로 변경함
- 새로운 구간 [40, 166.8]이 생겼음. P1과 P4 점을 잇는 직선이 x축과 만나는 점인 158을 근사해로 인정하지 못하니, 158에서의 부호를 판단함.
  - → +로 부호가 변경됨 → upper bound를 166.8 에서 158로 변경함

Newton-Raphson Algorithm

- 미분계수 $f'(x_i)$ 는 접점 x_i에서의 접선의 기울기를 충실히 따른 알고리즘
- 구하고자 하는 것은 x_{i+1}

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



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

- With $\Delta x = x_i - x_{i+1} \rightarrow 0$

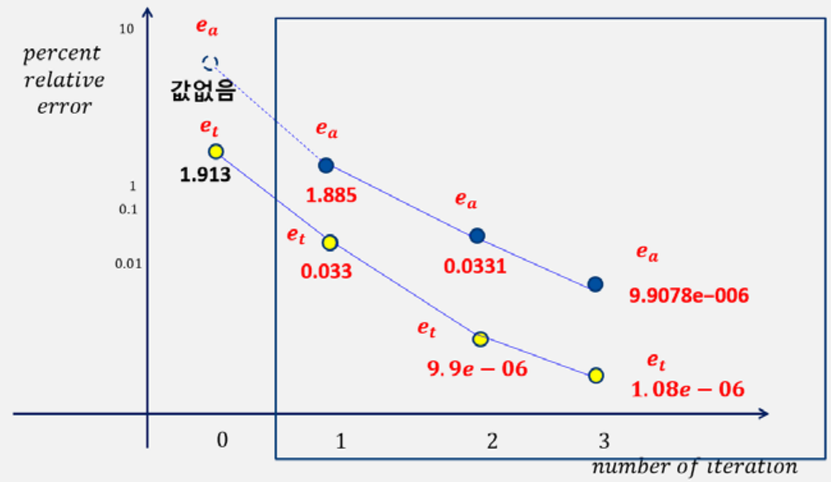
$$\lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} = f'(x_i)$$

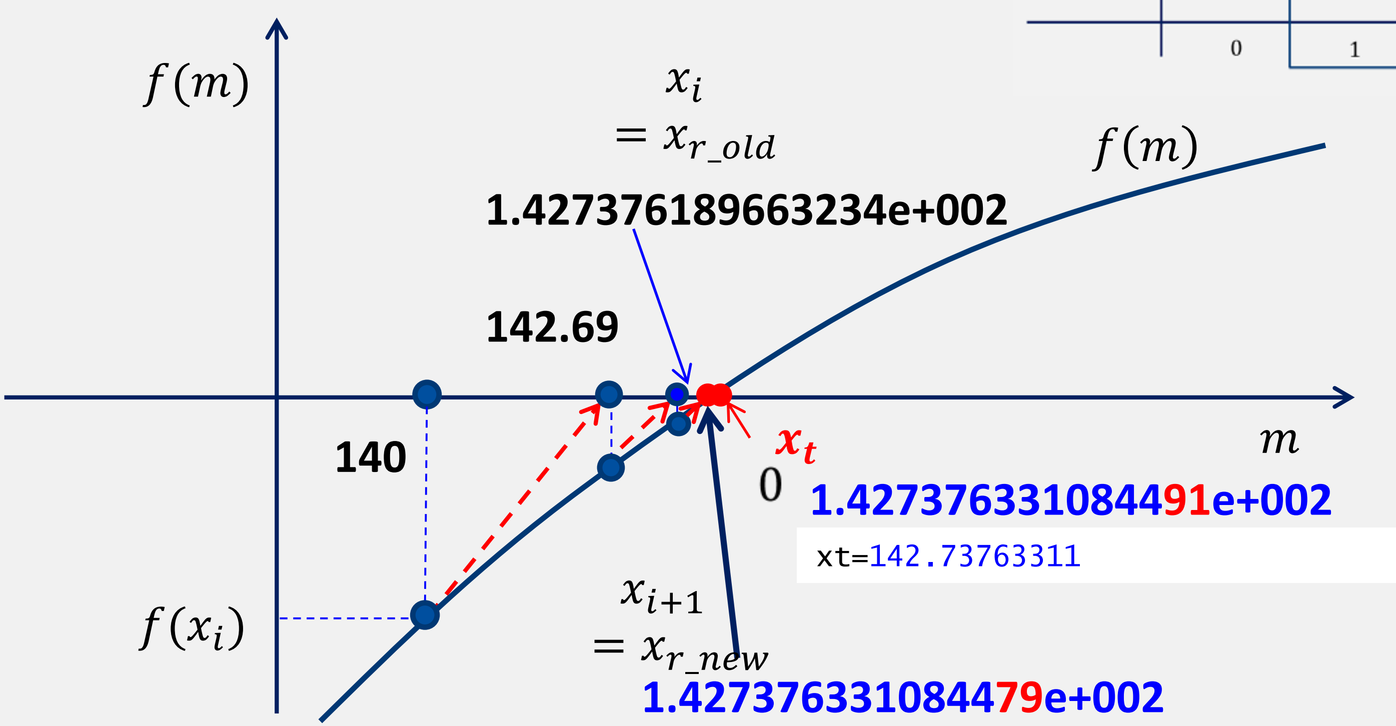
Newton-Raphson 방법

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

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

e_a and e_t





가위치법과 접선의 기울기

- 가위치법은 두 점을 지나는 직선으로부터 근사해를 구하는 기법이다
- 가위치법은 **두 점이 있으면 직선을 그을 수 있다**는 해법을 제시했다.
- **두 점을 지나는 직선**은 기울기와 절편을 가지고 있다.
- 두 점에서 한 개의 점이 다른 점에 가까이 붙게 된다고 가정하면, 가까이서 보면 두 개의 점으로 보이지만, 멀리서 보면 하나의 점같이 보이게 된다. 이 점을 접점이라 한다.
- 두 점이 아직 완벽히 붙지 않았음으로, 가까이서 보게 되면 **두 개의 점이 있음으로 직선을 그을 수 있게 되고**, 이 직선은 접점에서의 접선이 되게 된다.
- **접선은 기울기를 가지게 되고, 이 기울기 값이 접점에서의 미분 계수가 된다**

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

미분을 사용하는 이유

- 근사해를 구하는 과정에서 미분
  - 미분을 사용하면 f(x)=0 을 찾는 근사해를 구하는 과정에서 알고리즘의 연산횟수를 줄여주는 장점이 있다
  - Newton-Raphson 방법은 미분법으로 접선의 기울기를 이용함으로 직선을 사용하는 가위치법보다 빠르게 근사해에 접근했다.
- 최적화 과정에서의 미분
  - f’(x)=0에서의 미분의 의미는 순간변화율이 0이 되는 지점을 찾아 함수의 최대값과 최소값을 제공해주는 의미가 있다.
  - 자원들이 확실히 어떤 한계를 넘지 않고, 가능한 한 직면하는 요구사항의 대부분을 만족시키면서, 제조과정 (Manufacturing Operation) 의 이익을 최대로 하기 위한 것

Optimization (최적화 개념)

[7주/1차시 학습내용]: Ch.7 Optimization (최적화),
최적화 개념에 대해 학습한다

국민대학교 소프트웨어학부

Optimization (최적화 개념)

- 3차원 곡선 그래프를 통해 최적화 개념에 대해 알아본다

$$\mathrm{y} = f(x) = x^3 - 9x^2 + 24x - 7$$

$$\begin{aligned} y' &= f'(x) \\ &= 3x^2 - 18x + 24 \\ &= 3(x^2 - 6x + 8) = 3(x-2)(x-4) \end{aligned}$$

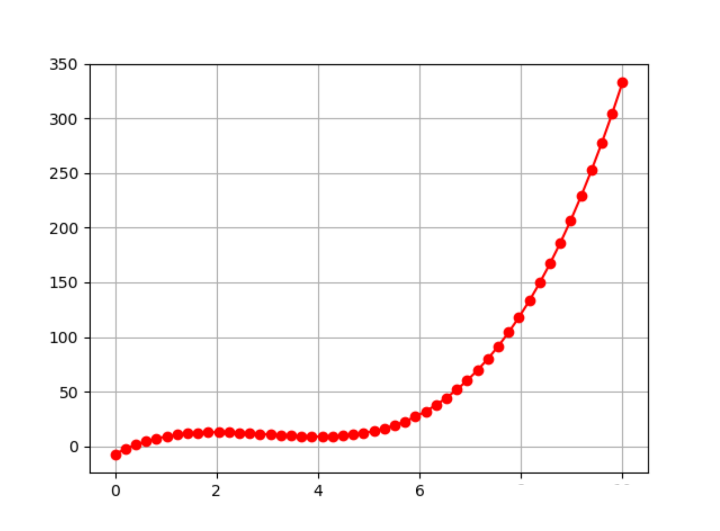
$$\mathrm{y} = f(x) = x^3 - 9x^2 + 24x - 7 = 0$$

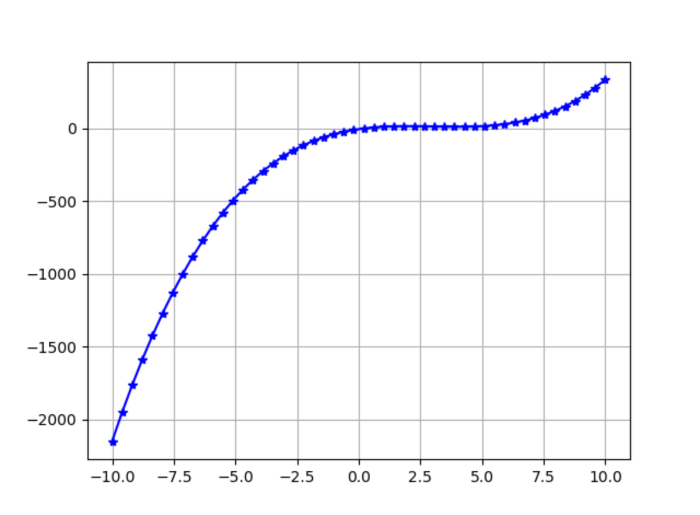
$$y' = f'(x) = 3(x-2)(x-4) = 0$$

Draw f(x) : 3차원 곡선 그래프 시각화

- x1의 범위는 0에서 10까지, x2의 범위는 -10에서 10까지 곡선 그래프를 보는 것이다.

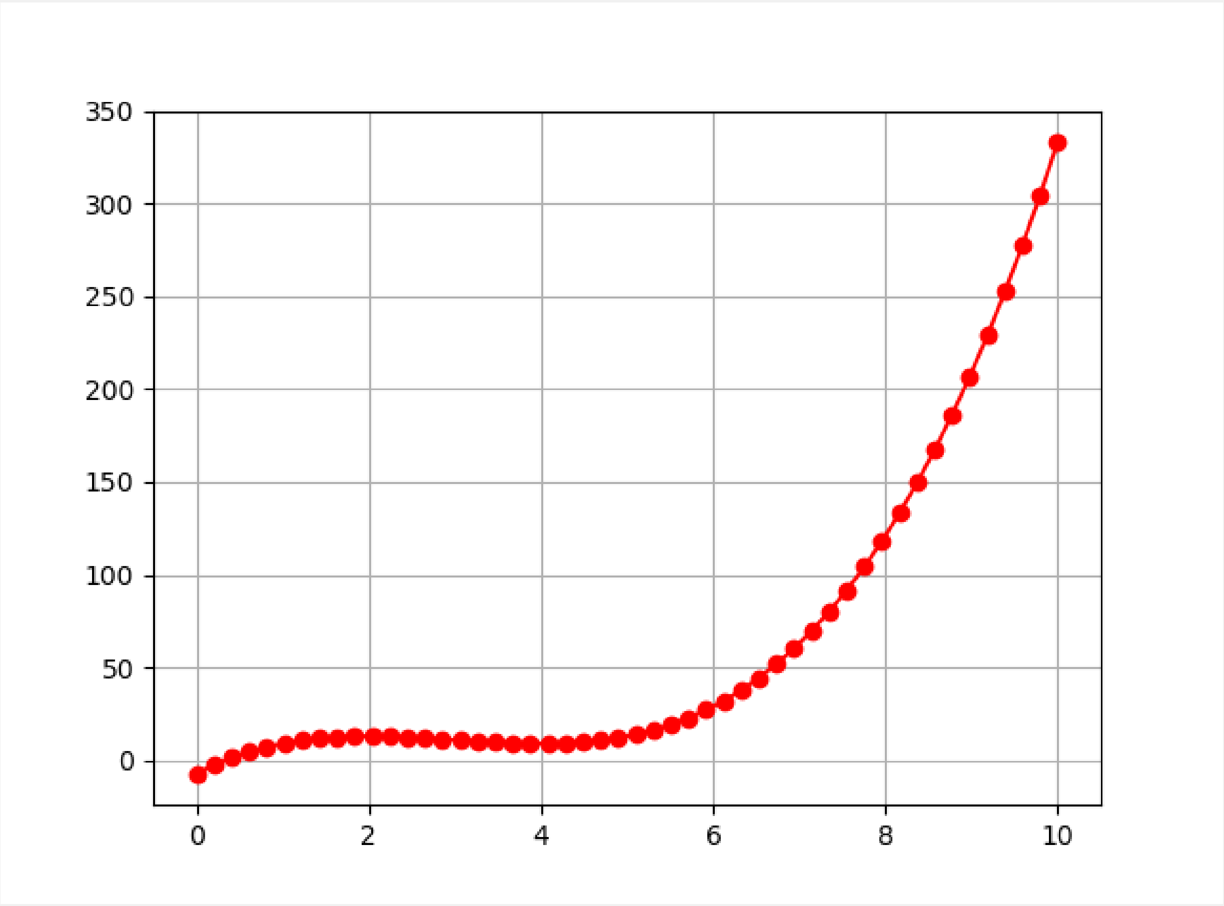
```python
import numpy as np
import matplotlib.pyplot as plt
x1=np.linspace(0,10,50)
x2=np.linspace(-10,10,50)
y1=x1**3-9*x1**2+24*x1-7
y2=x2**3-9*x2**2+24*x2-7
plt.figure(1)
plt.plot(x1, y1, 'ro-')
plt.grid()
plt.show()
plt.figure(2)
plt.plot(x2, y2, 'b*-')
plt.grid()
plt.show()
```





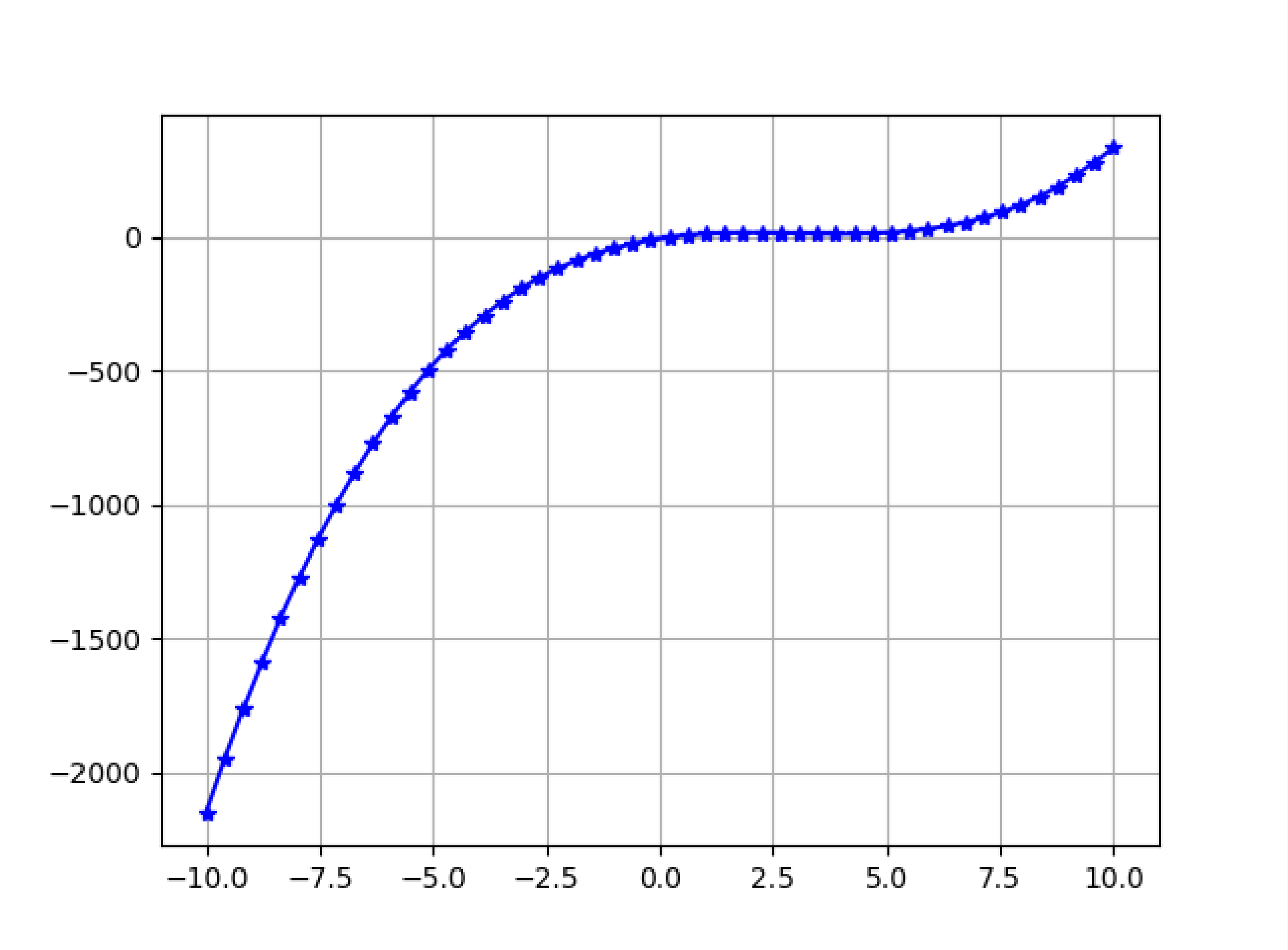
Draw f(x) : 3차원 곡선 그래프 시각화

- x1의 범위는 0에서 10까지 곡선 그래프를 보는 것이다.



Draw f(x) : 3차원 곡선 그래프 시각화

- x2의 범위는 -10에서 10까지 곡선 그래프를 보는 것이다.

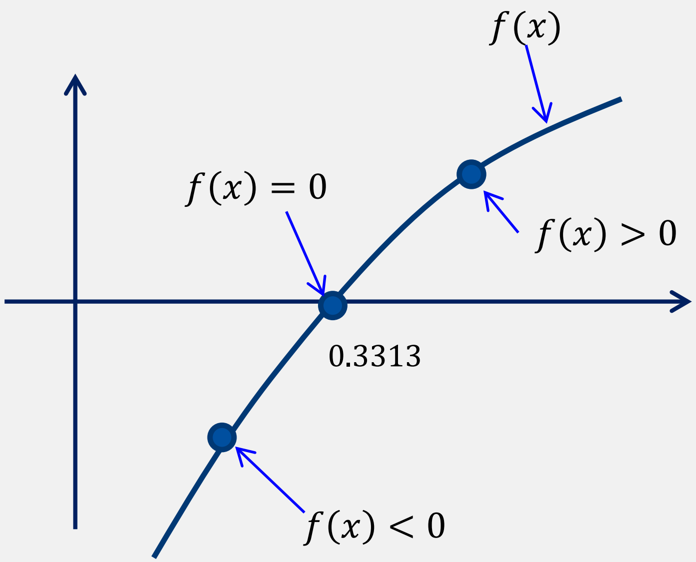


$f(x) = \mathbf{0}$: Root 또는 근사해를 구하는 과정

$$y = f(x) = x^3 - 9x^2 + 24x - 7 = 0$$

```
import numpy as np
from scipy.optimize import fsolve
fx=lambda x: x**3-9*x**2+24*x-7
x=fsolve(fx, 1)
print("Real Root= ", x)

Real Root=  [0.33131491]
```



- scipy.optimize.fsolve(func, x0) : Find the roots of a function.
- func : callable f(x, *args), A function that takes at least one (possibly vector) argument.
- x0 : ndarray, The starting estimate for the roots of func(x) = 0.

Newton Raphson for $f(x) = \mathbf{0}$: 근사해

```python
import numpy as np

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):
            root=xr
            fx=func(xr)

            return root, fx, ea, iter


if __name__ == '__main__':
    fp=lambda x: x**3-9*x**2+24*x-7
    dfp=lambda x: 3*x**2-18*x+24
    root, fx, ea, iter=newton_raphson(fp, dfp, 0.1)
    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)
```

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

https://github.com/SCKIMOSU/Numerical-Analysis/blob/master/3rd_newton.py

Newton Raphson for $f(x) = \mathbf{0}$: 근사해

```
root weight=  0.3313149095222536
f(root weight, should be zero) = -1.7763568394002505e-15
ea = should be less than 1.0e-4 4.208588441649749e-06
iter = 4
```

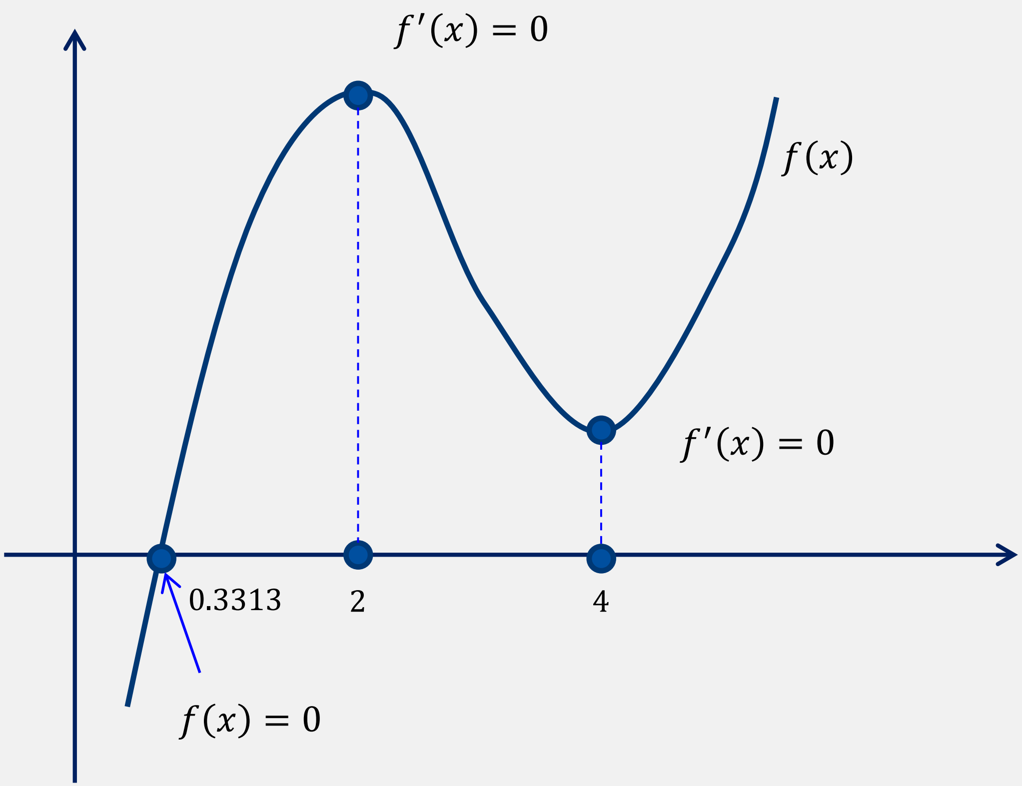
https://github.com/SCKIMOSU/Numerical-Analysis/blob/master/3rd_newton.py

$f'(x) = \mathbf{0}$: 최대, 최소를 구하는 과정

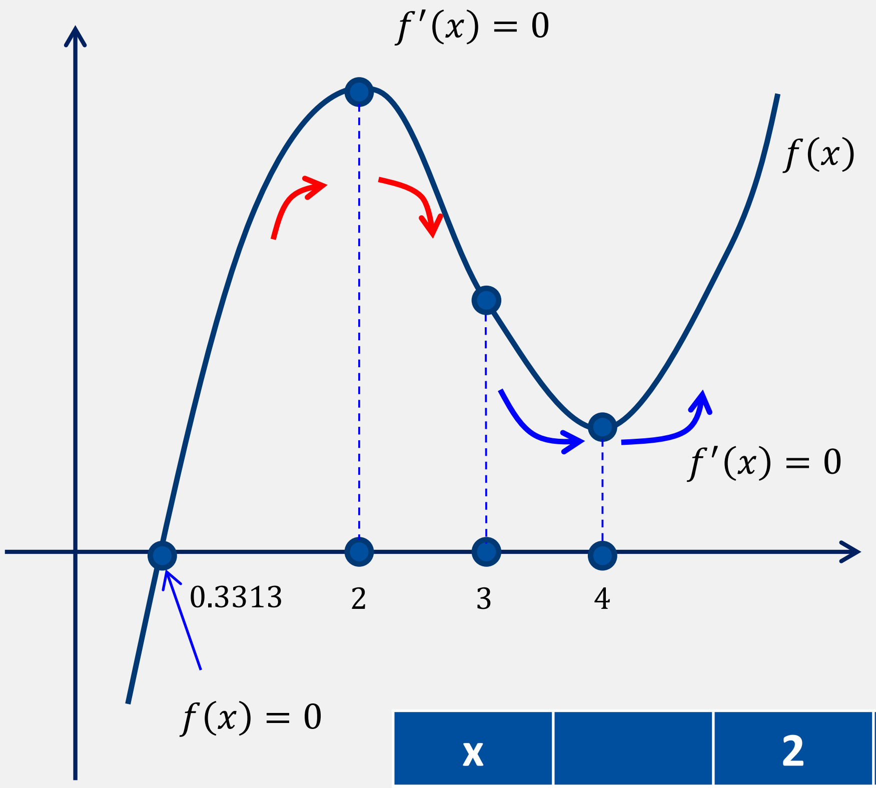
$$y' = 3x^2 - 18x + 24 = 3(x^2 - 6x + 8) = 3(x-2)(x-4) = 0$$
$$x = 2 \; or \; x = 4$$

```
import numpy as np
p = [3,-18,24]
xd1=np.roots(p)


array([4., 2.])
```



$f''(x) = \mathbf{0}$: 변곡점을 구하는 과정



$$f'(x) = 3(x-2)(x-4)$$

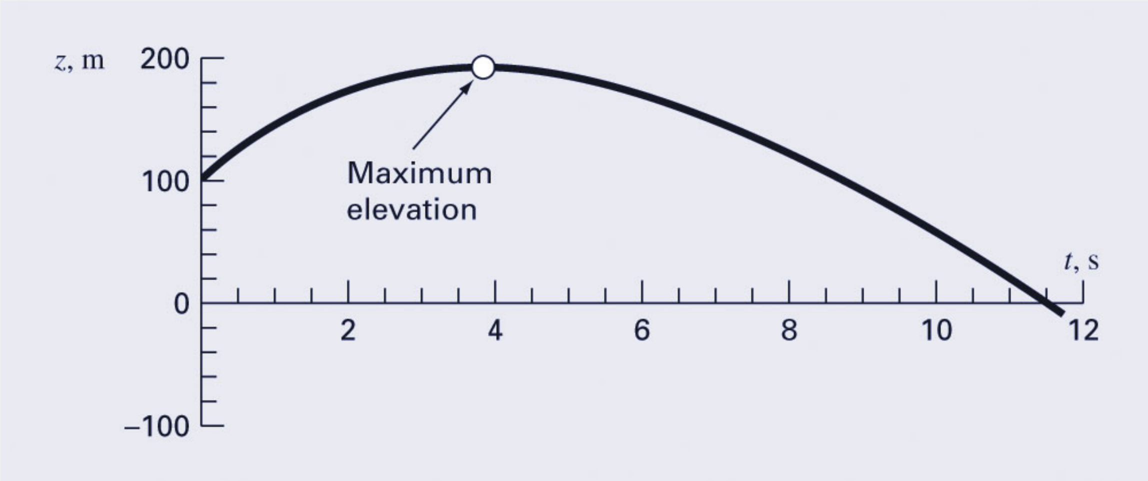
$$y'' = 6x - 18 = 6(x-3) = 0$$
$$x = 3$$

```
fp=lambda x: x**3-9*x**2+24*x-7
fp(2)=13
fp(4)=9
fp(3)=11
```

x		2		3		4	
$f'(x)$	+	0	−	−	−	0	+
$f''(x)$	−	−	−	0	+	+	+
$f(x)$		13		11		9	
의미	기울기가 감소하는 구간			변곡점	기울기가 증가하는 구간		

Optimization (최적화) $f'(x) = 0$을 만족하는 x를 찾아가는 과정

- **어떤 제약조건 (Constraints) 이 있을 수도 있는 상황에서 함수의 최대치와 최소치 (maxima and minima) 를 찾는 것**
- 자원들이 확실히 어떤 한계를 넘지 않고, 가능한 한 직면하는 요구사항의 대부분을 만족시키면서, **제조과정 (Manufacturing Operation) 의 이익을 최대로 하기 위한 것**



$$z = z_0 + \frac{m}{c}\left(v_0 + \frac{mg}{c}\right)\left(1 - e^{-(c/m)t}\right) - \frac{mg}{c}t$$

Optimization (최적화)

- 최적화이론과 그 해법은 일찍이 수학의 한 분야로서 유럽과 미국에서 여러 분야의 학자들에 의해 많이 연구돼 왔으며 제2차 세계대전 이후에는 산업 군사 행정 등의 여러 조직에 적극적으로 활용되기 시작하여 생활에 많은 변화를 가져왔다
- 사실은 우리 모두가 알게 모르게 최적화의 개념을 인식하고 있으며 그 해법 또한 나름대로 지니고 있다.
- 예를 들면 어떠한 물건을 구입하려 할 때 우리는 몇 가지 대안 중에서 재정적인 고려와 함께 구입 이유, 사용기간, 애프터서비스 사용 대상, 구입가격 등의 여러 조건을 비교 검토한 후 결정 내리게 된다.

http://www.aistudy.com/math/optimization.htm

Optimization (최적화)

- 물론 수학적인 기호나 컴퓨터를 통한 계산은 하지 않고 정형적인 모델은 수립하지 않더라도 그 방법론에 있어서는 최적화 기법이 그대로 적용되고 있는 셈이다.
- 더욱이 우리는 사회생활 주위에서 "최대의 효과" "최소의 비용" "최적의 선택" 등의 단어를 자주 접촉하고 있는 실정이다.
- 그러나 최적화기법을 체계적인 접근방법으로 이용, 의사결정을 하기는 그리 쉬운 일 이 아니며 또한 그 결정의 질을 평가하기도 무척 어려운 일이다
- **아마도 가장 강력한 최적화는 각 분야에서 최상의 알고리즘을 찾아내는 것일 것이다**

Optimization (최적화)

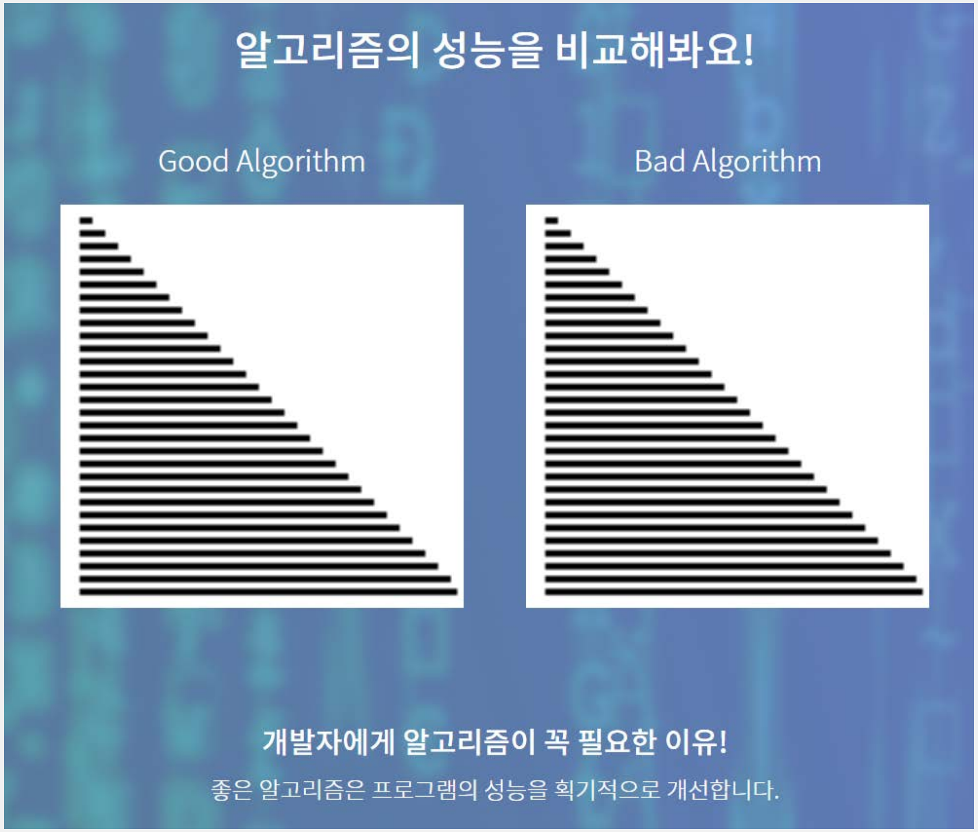
- 컴퓨터에서 최적화는 시스템의 효율이 증가되도록 변형시키는 것을 의미한다.
- 시스템 자원의 소모를 줄일 수도 있고 성능을 증가 시킬 수도 있다.
- 시스템은 단 하나의 컴퓨터일 수도 있고, 여러 대가 모인 것 일 수도 있고, 인터넷 같은 전체 네트워크 일 수도 있다.
- 최적화는 최대실행시간, 메모리 사용, 밴드 위스, 다른 자원들을 줄일 수 있게 한다.
- 그러한 목적들은 상호 배제 (서로 무관하게 발생) 일 수 있으며 tradeoff (교환, 균형을 취함) 를 요한다.

시스템 최적화

- 시스템의 구조 설계는 전적으로 그 성능에 영향을 줄 수 있다.
- Load balancing 은 많은 수의 서버에 로드를 전파한다.
- 소위 layer 4 router 를 사용해서 load balancing 은 투명하게 (예를 들면 사용자가 알지 못하게) 행해진다.
- Caching 은 복사 (duplicate) 를 피하기 위해 계산의 중간 산물을 저장한다.
- 전체 시스템을 최적화 하는 것은 보통 사람이 하게 되는데, 그 이유는 자동 optimizer 가 하기에는 시스템이 너무 복잡하기 때문이다.
- Grid computing 이나 distributed computing 은 많이 사용하는 컴퓨터로부터 휴 식중인 컴퓨터로 작업을 이동시킴으로써 전체 시스템을 최적화 하는 것을 목적으로 한다

알고리즘과 자료구조 최적화

- 알고리즘의 선택은 설계의 어떤 다른 요소보다도 효율성에 영향을 미친다.
- 보통, 단순한 알고리즘이 소량의 데이터에 더 적당한 반면에, 복잡한 알고리즘과 자료구조는 많은 자료를 가진 경우에 좋은 성능을 보인다.
- 소량의 데이터의 경우 더 복잡한 알고리즘의 초기에는 더 좋은 알고리즘의 장점을 뛰어넘을 수 있다.



순회판매원문제 (Traveling Salesman Problem)

- 많은 수의 도시가 있고, 한 도시에서 다른 도시로의 여행 경비를 알고 있을 때, 각 도시를 한 번만 방문하고 출발한 도시로 되돌아 오는데 <u>**가장 비용이 적게 드는 여행 경로는 무엇인가?**</u>
- 목적은 최소의 전체 가중치 (minimum total weight) 를 가지는 Hamiltonian cycle (모든 vertices를 통과 하는 cycle) 을 찾는 것이다.
- 그 cycle 은 여행 경로로 보통 표시된다.

Hamiltonian Cycle Problem

- William Rowan Hamilton 경은 1800 년 중반 12 면체의 모양에서 수수께끼 하나를 제시했다 (그림 1). 각 꼭지점에 도시 이름을 주고, 문제는 어떤 도시에서 출발하여, 간선들을 따라서 **한 도시를 한번만 방문하여,** 최초의 출발 도시로 돌아오는 것이다.

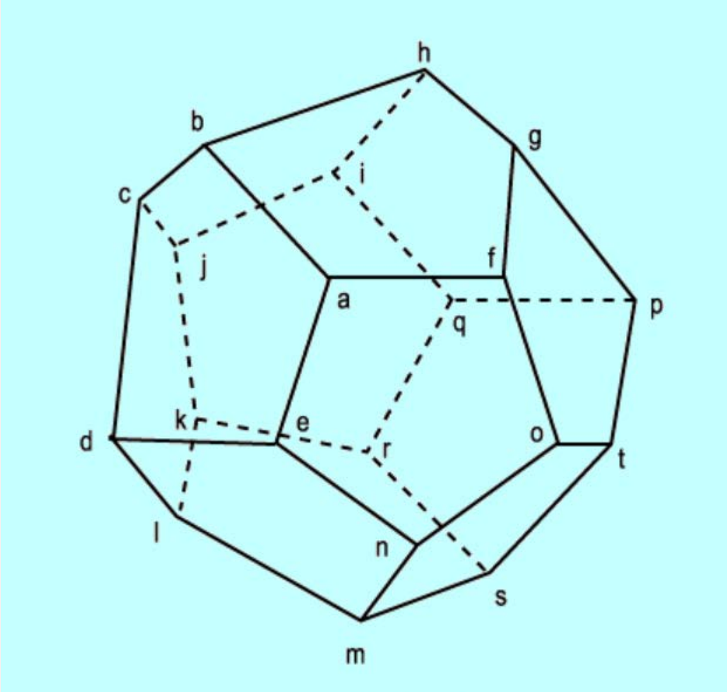


그림 1 해밀턴 퍼즐

Hamiltonian Cycle Problem

- 12 면체의 간선의 그래프는 그림 2 에 주어져 있다.
- 해밀턴의 수수께끼를 그림 2 의 그래프에서 사이클을 찾을 수 있다면 문제는 해결된다.
- 다만 각 정점은 한 번만 포함되어야 한다 (출발점과 끝나는 점은 두 번 나타나는 것을 제외하고). 그림 3 은 그 해이다.

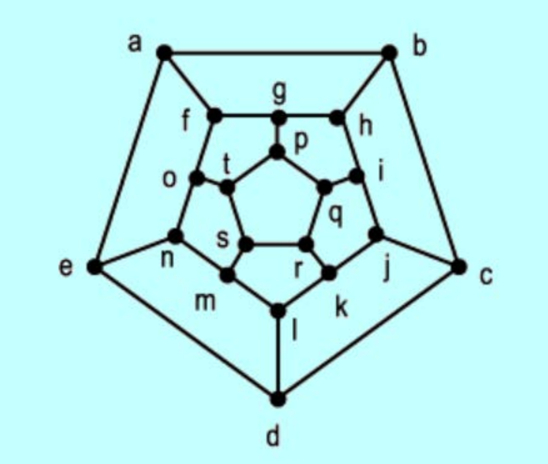


그림 2 해밀턴 퍼즐의 그래프

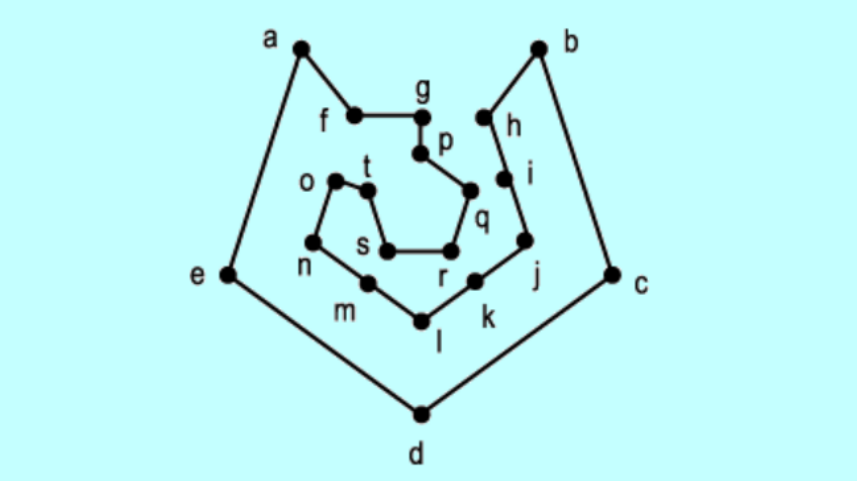


그림 3 그림 2 의 그래프에서 각 정점이 한번만 방문한다.

Computer Science의 최적화와 휴리스틱

- **<u>컴퓨터과학에서의 최적화</u>**
- 효율적인 실행 속도와 주파수 대역폭 (Bandwidth)를 증가시키고 메모리 요구량을 감소시키는 방향으로 어떤 시스템을 개선시키는 과정이다.
- 그 명칭에도 불구하고, 최적화가 반드시 문제에 대한 최적의 (**<u>Optimum</u>**) 해를 찾는 것을 의미하지는 않는다.
- 가끔 그것이 불가능할 경우에는 휴리스틱 (Heuristic) 알고리즘이 대신에 사용되어야 한다.

Computer Science의 최적화와 휴리스틱

- Heuristic 은 그리스어 "heutiskein" 가 어원이며 "to discover" 라는 의미를 가진다.
- 즉 이미 정립된 공식에 의해서가 아니라,
- 정보가 완전하지 않은 상황에서
- **노력을 통해서 시행착오 (trial and error) 를 거처, 또는 경험을 통해서 주먹구구식의 규칙 (Rule of Thumb) 을 통해 지식을 알게 되는 과정을 의미한다**.
- 잘 추측하는 기술 (art of good guessing) 이라고 표현하기도 한다
- 알고리즘 (Algorithm) 과는 달리 heuristic 은 해결책의 발견을 보장하지 않는다.

Optimization (최적화)

- 포탄이 최고로 올라갔을 때 걸리는 시간과 그 때의 높이는?
- 포탄의 공중 궤적을 z 라고 하면, $\frac{dz}{dt}$는 각 공중 궤적에서의 순간변화율이다.
- 순간 변화율이 0이 되는 곳에서 포탄이 최고의 높이를 가진다.

$$z = z_0 + \frac{m}{c}\left(v_0 + \frac{mg}{c}\right)\left(1 - e^{-(c/m)t}\right) - \frac{mg}{c}t$$

$$z = z_0 + \frac{m}{c}\left(v_0 + \frac{mg}{c}\right) - \frac{m}{c} \cdot \left(v_0 + \frac{mg}{c}\right) \cdot e^{-\left(\frac{c}{m}\right)t} - \frac{mg}{c}t$$

$$\frac{dz}{dt} = -\frac{m}{c} \cdot \left(v_0 + \frac{mg}{c}\right) \cdot e^{-\left(\frac{c}{m}\right)t} \cdot \left(-\frac{c}{m}\right) - \frac{mg}{c}$$

$$\frac{dz}{dt} = \left(-\frac{m}{c} \cdot v_0 - \frac{m}{c} \cdot \frac{mg}{c}\right) \cdot e^{-\left(\frac{c}{m}\right)t} \cdot \left(-\frac{c}{m}\right) - \frac{mg}{c}$$

Optimization (최적화)

- 포탄의 공중 궤적을 $\boldsymbol{z}$ 라고 하면, $\frac{\boldsymbol{dz}}{\boldsymbol{dt}}$는 각 공중 궤적에서의 순간변화율이다. 순간 변화율이 0이 되는 곳에서 포탄이 최고의 높이를 가진다

$$\frac{\boldsymbol{dz}}{\boldsymbol{dt}} = \left(\frac{m}{c} \cdot \frac{\boldsymbol{c}}{\boldsymbol{m}} v_0 + \frac{m}{c} \cdot \frac{\boldsymbol{c}}{\boldsymbol{m}} \frac{mg}{c}\right) \cdot e^{-\left(\frac{c}{m}\right)t} - \frac{mg}{c}$$

$$\frac{dz}{dt} = \left(v_0 + \frac{mg}{c}\right) \cdot e^{-\left(\frac{c}{m}\right)t} - \frac{mg}{c}$$

$$\frac{dz}{dt} = v_0 \cdot e^{-\left(\frac{c}{m}\right)t} + \frac{mg}{c} \cdot e^{-\left(\frac{c}{m}\right)t} - \frac{mg}{c}$$

$$\frac{\boldsymbol{dz}}{\boldsymbol{dt}} = v_0 \cdot e^{-\left(\frac{c}{m}\right)t} - \frac{mg}{c} \cdot \left(1 - e^{-\left(\frac{c}{m}\right)t}\right) \qquad \frac{\boldsymbol{dz}}{\boldsymbol{dt}} = \boldsymbol{0}, \qquad t = ?$$

$$\boldsymbol{0} = \frac{\boldsymbol{dz}}{\boldsymbol{dt}} = v_0 \cdot e^{-\left(\frac{c}{m}\right)t} - \frac{mg}{c} \cdot \left(1 - e^{-\left(\frac{c}{m}\right)t}\right)$$

Optimization (최적화)

- 순간 변화율 $\frac{dz}{dt}$ 이 0이 되는 곳에서 포탄이 최고의 높이를 가진다

$$0 = \frac{dz}{dt} = v_0 \cdot e^{-\left(\frac{c}{m}\right)t} - \frac{mg}{c} \cdot \left(1 - e^{-\left(\frac{c}{m}\right)t}\right)$$

$$0 = v_0 \cdot e^{-\left(\frac{c}{m}\right)t} - \frac{mg}{c} \cdot \left(1 - e^{-\left(\frac{c}{m}\right)t}\right)$$

$$v_0 \cdot e^{-\left(\frac{c}{m}\right)t} = \frac{mg}{c} \cdot \left(1 - e^{-\left(\frac{c}{m}\right)t}\right)$$

$$v_0 \cdot e^{-\left(\frac{c}{m}\right)t} = \frac{mg}{c} - \frac{mg}{c} \cdot e^{-\left(\frac{c}{m}\right)t}$$

$$v_0 \cdot e^{-\left(\frac{c}{m}\right)t} + \frac{mg}{c} \cdot e^{-\left(\frac{c}{m}\right)t} = \frac{mg}{c}$$

$$\left(v_0 + \frac{mg}{c}\right) \cdot e^{-\left(\frac{c}{m}\right)t} = \frac{mg}{c} \qquad e^{-\left(\frac{c}{m}\right)t} = \frac{\frac{mg}{c}}{\left(v_0 + \frac{mg}{c}\right)}$$

Optimization (최적화)

- 순간 변화율 $\frac{dz}{dt}$ 이 0이 되는 곳에서 포탄이 최고의 높이를 가진다

$$e^{-\left(\frac{c}{m}\right)t} = \frac{\frac{mg}{c}}{\frac{v_0 c + mg}{c}}$$

$$e^{-\left(\frac{c}{m}\right)t} = \frac{mg}{v_0 c + mg}$$

$$\frac{1}{e^{\left(\frac{c}{m}\right)t}} = \frac{mg}{v_0 c + mg}$$

$$e^{\left(\frac{c}{m}\right)t} = \frac{v_0 c + mg}{mg}$$

$$e^{\left(\frac{c}{m}\right)t} = \frac{v_0 c}{mg} + 1 \qquad \ln e^{\left(\frac{c}{m}\right)t} = \ln\left(\frac{v_0 c}{mg} + 1\right)$$

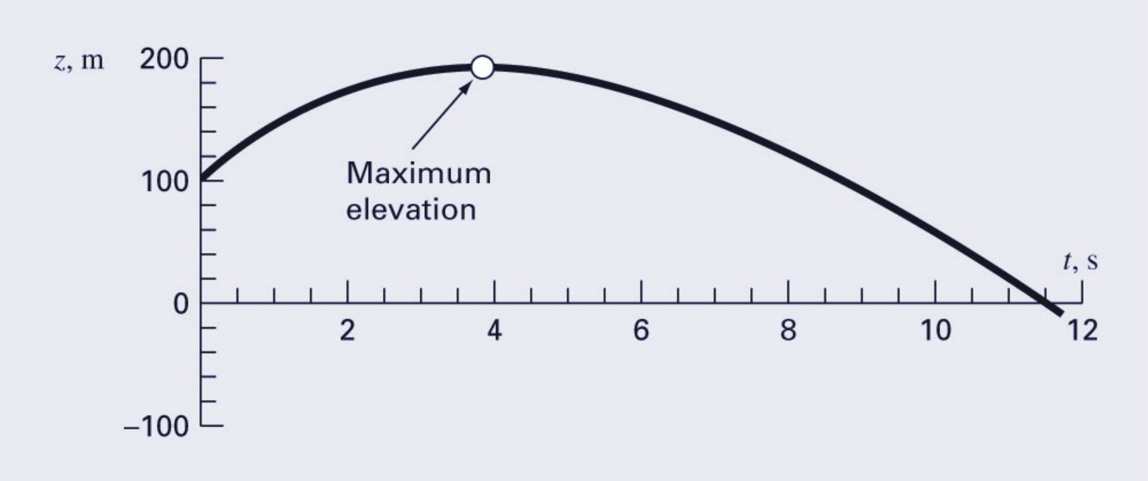
Optimization (최적화)

- 순간 변화율 $\boldsymbol{\frac{dz}{dt}}$ 이 0이 되는 곳에서 포탄이 최고의 높이를 가진다

$$\ln e^{\left(\frac{c}{m}\right)t} = \ln\left(\frac{v_0 c}{mg} + 1\right)$$

$$\left(\frac{c}{m}\right)t = \ln\left(1 + \frac{v_0 c}{mg}\right)$$

$$\boldsymbol{t = \frac{m}{c} \cdot \ln\left(1 + \frac{v_0 c}{mg}\right)}$$



$$t = \frac{80kg}{15} \cdot \ln\left(1 + \frac{\frac{55m}{s} \cdot 15}{80 \cdot 9.81}\right)$$

```
t=80/15*np.log(1+55*15/(80*9.81))
t
3.8316603648452263
```

$$t = 3.8317$$

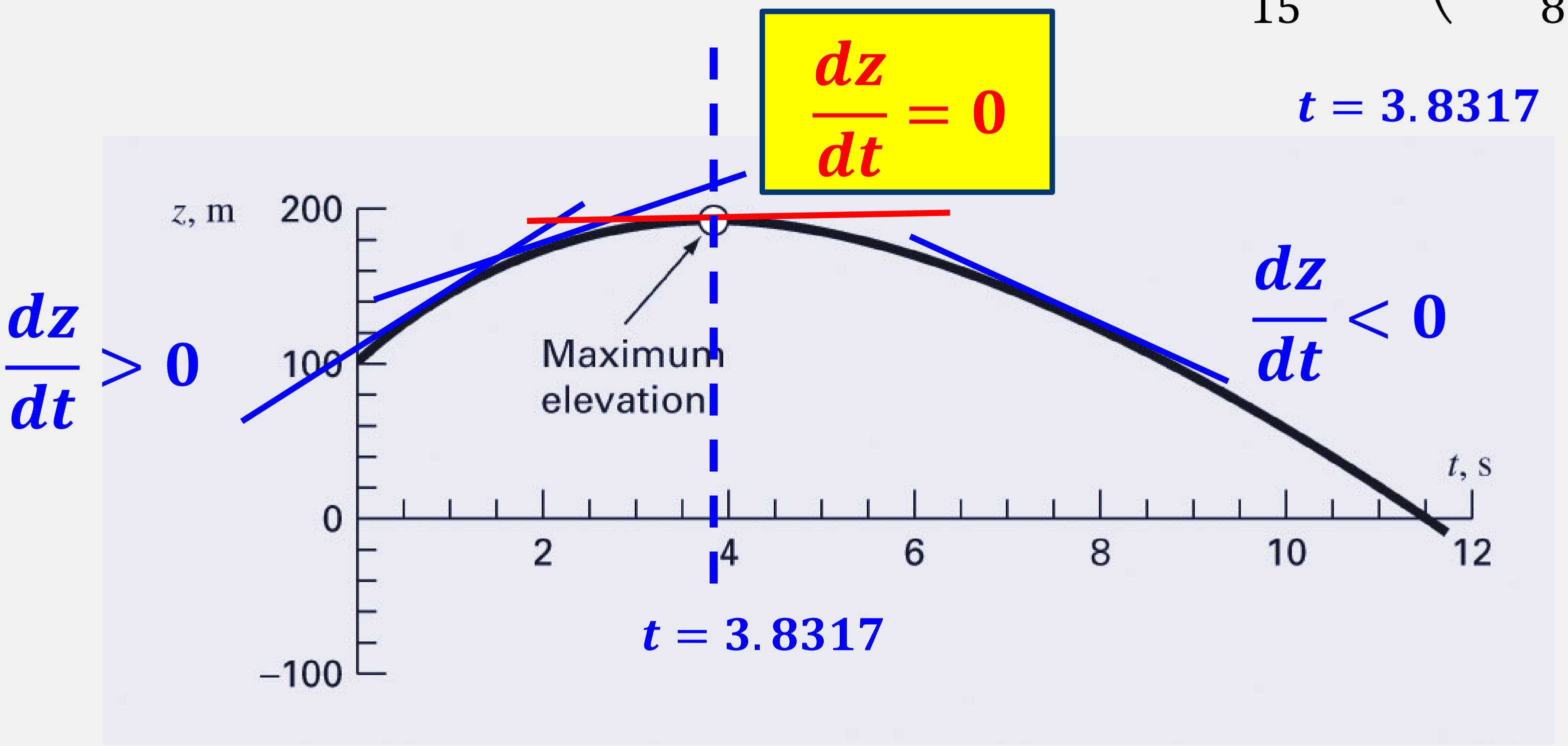
Optimization (최적화)

- 순간 변화율 $\frac{dz}{dt}$ 이 0이 되는 곳에서 포탄이 최고의 높이를 가진다

$$t = \frac{m}{c} \cdot \ln\left(1 + \frac{v_0 c}{mg}\right)$$

$$t = \frac{80kg}{15} \cdot \ln\left(1 + \frac{\frac{55m}{s} \cdot 15}{80 \cdot 9.81}\right)$$

$t = 3.8317$



Root and Optimization (근과 최적화)

$f(x) = \mathbf{0}$: Root 또는 근사해를 구하는 과정

$f'(x) = \mathbf{0}$: Optimization은 최대, 최소를 구하는 과정