삼분할법.
피보나치
황금분할 탐색법
일변량
이분법 (Bisec

04. 구간 분할법 (Bracketing)

경사법 (Local descent)

다변량

구간 분할법

- 황금 분할 탐색법 : 삼분할법

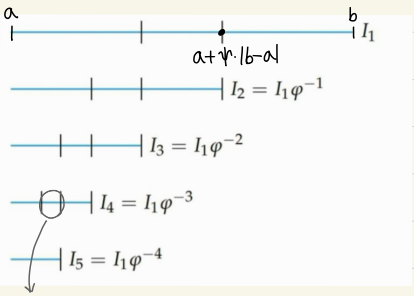
- $\lim_{n\to\infty}\frac{F_n}{F_{n-1}} = \psi$ 단, F_n은 n번째 피보나치 수열값.

  이미 증명된 것. 피보나치에서는 구간의 매 단계마다 $\frac{F_n}{F_{n-1}}$ 계산.
  두 피보나치 수열의 비율 값이 수렴된 값 (Golden ratio, ψ) 이용

- 피보나치는 매 단계마다 구간의 길이 비율 변화.

  삼분할 법 中 황금분할 탐색법은 ψ 고정

  → 매번 동일한 Golden Ratio에 의해 구간 분할



피보나치 탐색법에서 마지막 단계의 가까운 값으로 결정
황금 분할법에서 구간의 크기 좀 더 커짐.

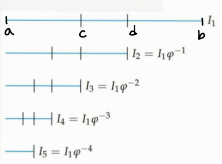
- 매번 비율 계산 안 해도 되어 계산량↓

- 알고리즘

convex-combination 처럼 표현.

반복 수 지정

```
function golden_section_search(f, a, b, n)
    ρ = φ-1 → -1승
    d = ρ * b + (1 - ρ)*a (b-a)ρ+a 변형
    yd = f(d)
    for i = 1 : n-1
        c = ρ*a + (1 - ρ)*b
        yc = f(c)
        if yc < yd update 시작!
            b, d, yd = d, c, yc
        else
            a, b = b, c
        end
    end
    return a < b ? (a, b) : (b, a)
end
```



- Golden Section Search Method

  : 최소값이 있다고 생각되는 구간 결정해야 함. (feasible set)

  bracket minimum Algorithm 이용.

황금분할 탐색법

→ 구현된 알고리즘 이용해 feasible set (a,b) 결정 이후 Golden Section search method 이용해 최소값 찾기.

```
def bracket_minimum(f,x,s=1E-2,k=2.0):
    a,ya=x,f(x)
    b,yb=a+s,f(a+s)

    print('init: (a:%.4f, b:%.4f) (ya:%.4f yb:%.4f)' %(a,b,ya,yb))

    if yb>ya:
        a,b=b,a
        ya,yb=yb,ya
        s=-s

    while True:
        c,yc=b+s, f(b+s)
        print('step: (a:%.4f, b:%.4f, c:%.4f) (ya:%.4f, yb:%.4f yc:%.4f)' %(a,b,c,ya,yb,yc))

        if yc>yb:
            return (a,c) if a<c else (c,a)

        else:
            a,ya,b,yb=b,yb,c,yc
            s*=k

def golden_section_search(f,x,epsilon=1E-6):
    a,b=bracket_minimum(f,x)
    print('init:(a:%.4f, b:%.4f)' %(a,b))

    distance=abs(a-b)

    psi=0.5*(1.+np.sqrt(5))
    rho = psi**(-1) # 구간을 결정해주는 convex combination하기 위해 rho 계산: 황금비율의 역수

    d=rho*b+(1.-rho)*a
    yd=f(d)

    i=1
    while distance>epsilon:

        c=rho*a+(1.-rho)*b #같은 방법의 convex combination이용
        yc=f(c)

        if yc<yd:
            b, d,yd =d, c,yc

        else:
            a,b=b,c

        pa,pb=(a,b) if a<b else(b,a)
        print('%d:(a:%.4f, b:%.4f)'%(i,pa,pb))

        distance=abs(a-b)

        i+=1

    a,b=(a,b) if a<b else(b,a)
    x=0.5*(a+b)
    y=f(x)
    return x,y
```

f: 목적함수, x: bracket_minimum에 의한 feasible set 찾기 위한 초기값

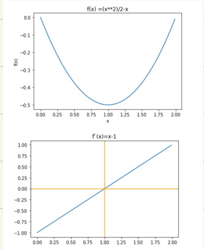
epsilon → 허용오차.

while distance>epsilon: 구간 길이보다 허용오차가 커지면 반복 멈춤.

```
import numpy as np
from IPython.display import Image
import matplotlib.pyplot as plt
#Image('golden_section.png',width=640)

f=lambda x:0.5*x**2-x
df=lambda x:x-1
x=np.arange(0.,2.,1E-2)

plt.title('f(x) =(x**2)/2-x')
plt.plot(x,f(x))
plt.xlabel('x')
plt.ylabel('f(x)')
plt.show()

# 도함수 : 직선
plt.title('f`(x)=x-1')
plt.plot(x,df(x))
plt.axhline(y=0,color='orange')
plt.axvline(x=1,color='orange')
plt.show()

print('golden_section_search algorithm')
print(golden_section_search(f,0))
```

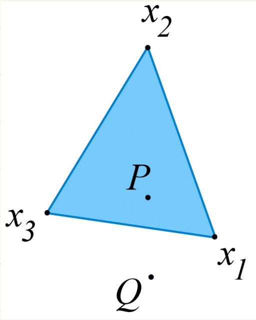
→ arange : 실수값
range : 정수

→ x=1 일때 최소값 존재.




```
golden_section_search algorithm
init: (a:0.0000, b:0.0100) (ya:0.0000 yb:-0.0100)
step: (a:0.0000, b:0.0100, c:0.0200) (ya:0.0000, yb:-0.0100 yc:-0.0198)
step: (a:0.0100, b:0.0200, c:0.0400) (ya:-0.0100, yb:-0.0198 yc:-0.0392)
step: (a:0.0200, b:0.0400, c:0.0800) (ya:-0.0198, yb:-0.0392 yc:-0.0768)
step: (a:0.0400, b:0.0800, c:0.1600) (ya:-0.0392, yb:-0.0768 yc:-0.1472)
step: (a:0.0800, b:0.1600, c:0.3200) (ya:-0.0768, yb:-0.1472 yc:-0.2688)
step: (a:0.1600, b:0.3200, c:0.6400) (ya:-0.1472, yb:-0.2688 yc:-0.4352)
step: (a:0.3200, b:0.6400, c:1.2800) (ya:-0.2688, yb:-0.4352 yc:-0.4608)
step: (a:0.6400, b:1.2800, c:2.5600) (ya:-0.4352, yb:-0.4608 yc:0.7168)
init:(a:0.6400, b:2.5600)
1:(a:0.6400, b:1.8266)
2:(a:0.6400, b:1.3734)
3:(a:0.6400, b:1.0933)
4:(a:0.8131, b:1.0933)
5:(a:0.9201, b:1.0933)
6:(a:0.9201, b:1.0271)
7:(a:0.9610, b:1.0271)
8:(a:0.9863, b:1.0271)
9:(a:0.9863, b:1.0115)
10:(a:0.9959, b:1.0115)
11:(a:0.9959, b:1.0055)
12:(a:0.9959, b:1.0019)
13:(a:0.9982, b:1.0019)
14:(a:0.9982, b:1.0005)
15:(a:0.9990, b:1.0005)
16:(a:0.9996, b:1.0005)
17:(a:0.9996, b:1.0001)
18:(a:0.9998, b:1.0001)
19:(a:0.9999, b:1.0001)
20:(a:0.9999, b:1.0000)
21:(a:1.0000, b:1.0000)
22:(a:1.0000, b:1.0000)
23:(a:1.0000, b:1.0000)
24:(a:1.0000, b:1.0000)
25:(a:1.0000, b:1.0000)
26:(a:1.0000, b:1.0000)
27:(a:1.0000, b:1.0000)
28:(a:1.0000, b:1.0000)
29:(a:1.0000, b:1.0000)
30:(a:1.0000, b:1.0000)
31:(a:1.0000, b:1.0000)
(0.9999999426899366, -0.49999999999999833)
```

⊕ Convex Combination



$$\begin{cases} P = a_1 x_1 + \cdots + a_n x_n \\ \Sigma a_n = 1, \quad a_n \geq 0 \end{cases}$$

→ Convex Combination은 위 식 만족하는 점 P들

즉, 주어진 지점 서로 연결한 도형 안에 존재하는 지점들

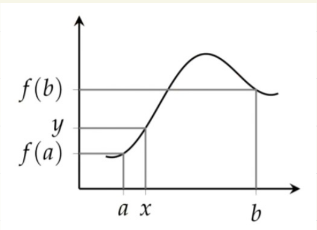
구간 탐색법

- 이분법

- 함수의 근 탐색 : $f(x)=0$
- 구간 $[a,b]$에 근이 존재한다면 $f(a)$와 $f(b)$의 부호는?
- $f(a)$, $f(b)$의 값을 비교하는 것이 아닌, 부호 비교.
- 근이 존재하는 구간 반복적으로 이분 (Bisection)
- 최적화 : $f'(x)=0$ → 탄젠트 라인이 수평선 되는 곳 찾기
- 삼분할법보다 직관적인 방법
- 수학적 증명 (중간값 정리, intermediate value theorom)

→ 함수 $f:[a,b]$에서 연속

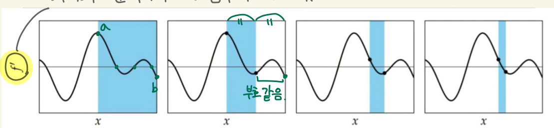
→ $y \in [f(a), f(b)]$에 대해 $f(x)=y$인 $x \in [a,b]$



↓

x가 존재하지 않으면 f함수는 불연속, 이분 되는 점 존재 X

최적화 문제니까 도함수의 근 찾기.



$$\mathrm{Sign}(f'(x)) \neq \mathrm{Sign}(f'(b)) \Leftrightarrow f'(a)\cdot f'(b) \leq 0$$

- 삼분할법에서 bracket_minimum에서 찾은 최소값 존재 구간

구간의 함수값 큰지 작은지 이용해 feasible set 찾음.

→ 이분법에서 구간 a,b를 찾으려고 하면 a,b에서의 도함수 값이 부호 일치 여부 이용해 탐색

- 이분법 알고리즘 (3분할법과 다름)

도함수

a와 b는 가까운 값으로 구간 결정 : 부호 같을 가능성 ↑

```
function bracket_sign_change(f′, a, b; k=2)
    if a > b; a,b = b,a; end # ensure a < b

    center, half_width = (b+a)/2, (b-a)/2
    while f′(a)*f′(b) > 0
        half_width *= k
        a = center - half_width
        b = center + half_width
    end

    return (a,b)
end
```

증가시키는 값의 길이

↳ a와 b를 계속 벌려나가 부호가 다르게 해주는 a,b값 → feasible set

목적함수(X). 도함수(O)

가까운 값

k : width 늘려나가는 비율.

하이퍼파라미터

```
def bracket_sign_change(df,a,b,k=2):

    if a>b:
        a,b=b,a

    center,half_width = 0.5*(b+a), 0.5*(b-a)

    while df(a)*df(b)>0:

        half_width*=k

        a=center-half_width
        b=center+half_width

    return (a,b)

def bisection(df,x,epsilon=1E-6):

    a,b=bracket_sign_change(df,x-epsilon,x+epsilon)
    print('init:(a:%.4f, b:%.4f)'%(a,b)) #feasible set지정

    ya,yb=df(a),df(b)

    if ya==0:
        b=a
    if yb==0:
        a=b

    i=1
    while b-a > epsilon: #b-a는 구간의 길이: b가 항상 a보다 크니까
        x=0.5*(a+b)
        y=df(x)

        if y==0:
            a,b=x,x
        elif y*ya>0: #두 도함수의 부호가 같다 -> a점을 x로 이동
            a=x
        else: #그렇지 않으면 b를 x로 이동
            b=x

        print('step %d -a:%.4f, b:%.4f, y:%.4f, ya:%.4f' %(i,a,b,y,ya))

        i+=1

        x=0.5*(a+b)
        y=df(x)

    return x,y

bisection(df,0)
```

목적함수(X). 도함수(O)

이분법에 의해 찾은 구간 길이 epsilon보다 작으면 멈춤.

비교적 가까운 x-epsilon, x+epsilon

└ 도함수의 근이 포함된 구간 = 목적함수 f의 최소값 포함되어 있는 feasible set

직관적이지만 확인해야 할 것 多

bisection이 최적화에 더 어려운 경우 존재.

```
init:(a:-1.0486, b:1.0486)
step 1 -a:0.0000, b:1.0486, y:-1.0000, ya:-2.0486
step 2 -a:0.5243, b:1.0486, y:-0.4757, ya:-2.0486
step 3 -a:0.7864, b:1.0486, y:-0.2136, ya:-2.0486
step 4 -a:0.9175, b:1.0486, y:-0.0825, ya:-2.0486
step 5 -a:0.9830, b:1.0486, y:-0.0170, ya:-2.0486
step 6 -a:0.9830, b:1.0158, y:0.0158, ya:-2.0486
step 7 -a:0.9994, b:1.0158, y:-0.0006, ya:-2.0486
step 8 -a:0.9994, b:1.0076, y:0.0076, ya:-2.0486
step 9 -a:0.9994, b:1.0035, y:0.0035, ya:-2.0486
step 10 -a:0.9994, b:1.0015, y:0.0015, ya:-2.0486
step 11 -a:0.9994, b:1.0004, y:0.0004, ya:-2.0486
step 12 -a:0.9999, b:1.0004, y:-0.0001, ya:-2.0486
step 13 -a:0.9999, b:1.0002, y:0.0002, ya:-2.0486
step 14 -a:0.9999, b:1.0001, y:0.0001, ya:-2.0486
step 15 -a:1.0000, b:1.0000, y:0.0000, ya:-2.0486

Out[9]: (1.0, 0.0)
```