03. 구간 분할법 (bracketing)

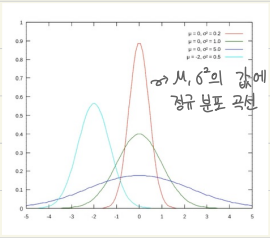
↓

최적점 포함 구간 점점 작게 나눔.

최적화 목적함수의 최소값, 최대값 찾기

반복계산 → minimizer 찾기

단봉성



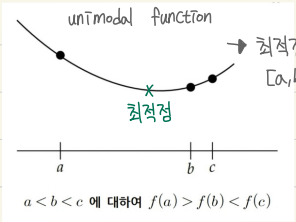
→ μ, σ^2의 값에 따른 정규 분포 곡선

- 구간 분할법에 의해 최적점 (최소점)을 찾기 위해 unimodality 가정 필요 ⇒ 단봉성
- 함수의 가장 높이 위치한 곳 → max 값 하나.
- 적절한 구간만 잘 선택해 구간 줄이기
  → 단봉성 만족시 최대값 찾을 수 O
  global min/max 존재

⇒ 단봉성 만족하는 함수는 최적점 하나만 존재

– 목적 함수 : 단봉형 함수 (unimodal function)

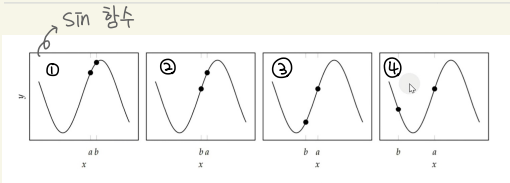
- 구간 분할법의 강한 가정 → 유일한 최적점 x^*
- 목적 함수 f : 단조 감소 ($x \le x^*$), 단조 증가 ($x \ge x^*$) (=미분 가능)
- 최적점 포함 구간 $[a, c]$
  ↳ 삼점 분할 : $a<b<c$ 에 대하여 $f(a) > f(b) > f(c)$



- Global minimum / local minimum
  → 함수 커브 모양 : 최소값 모양 ex) sin 함수 <단봉성 X>
  최적화 문제에서 x의 집합 (feasible set) 결정
  sin 함수에서 feasible set은 $[0, 2\pi]$, 전체구간 X
  관심있는 구간에 대한 local min. 단봉성으로 세 점 이용해 구하기.

all global?

초기 구간 탐색



① 차이가 작은 두 값 선택하기

② b : 목적 함수 값 적게 하는 b 값 선택

④ b : ③의 목적함수 값보다 커짐

→ ③과 ④ 구간 내 최적점 존재.

⇒ feasible set 내 minimum 하나만 존재.

최적점 구간 결정하고 구간 줄이기!

- 구간 탐색 알고리즘

```
function bracket_minimum(f, x=0; s=1e-2, k=2.0)
    a, ya = x, f(x)
    b, yb = a + s, f(a + s)
    if yb > ya
        a, b = b, a
        ya, yb = yb, ya
        s = -s
    end
    while true
        c, yc = b + s, f(b + s)
        if yc > yb
            return a < c ? (a, c) : (c, a)
        end
        a, ya, b, yb = b, yb, c, yc
        s *= k
    end
end
```

- bracket_minimum 모수

① f : unimodality 만족.

② x = 0 : 임의점

③ s = 1e-2 : a를 b로 이동시키는 값

④ k = 2.0 : 크게 이동

⇒ s, k : hyper-parameter (초모수, 조율 모수)

임의로 결정

구간 탐색 알고리즘

- Algorithm 3.1 : p.36

```
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 # 스왑핑
        ya,yb=yb,ya
        s=-s # b값 줄어나감

    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) # 구간의 하한 상한 결정
            # while 루프 빠져나옴
        else:
            a,ya,b,yb=b,yb,c,yc
            s*=k # 더 넓은 구간으로 이동

print('x=-1일때')
print(bracket_minimum(f,-1))
print('')
print('x=1일때')
print(bracket_minimum(f,1))
```

↗ 마지막에 a,b,C로 이루어진 구간 사이에 최적점 존재.

↗ 왼쪽으로 이동하며 최적점 포함구간 결정.

최소값 0, 점 3개로 구간 정하고 (a,b) (b,c) 파악

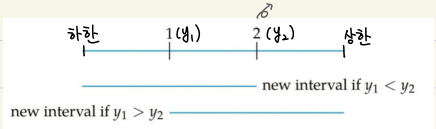
```
x=-1일때
init: (a:-1.0000, b:-0.9900) (ya:1.0000, yb:0.9801)
step: (a: -1.0000, b:-0.9900, c:-0.9800) (ya:1.0000, yb:0.9801, yc:0.9604)
step: (a: -0.9900, b:-0.9800, c:-0.9600) (ya:0.9801, yb:0.9604, yc:0.9216)
step: (a: -0.9800, b:-0.9600, c:-0.9200) (ya:0.9604, yb:0.9216, yc:0.8464)
step: (a: -0.9600, b:-0.9200, c:-0.8400) (ya:0.9216, yb:0.8464, yc:0.7056)
step: (a: -0.9200, b:-0.8400, c:-0.6800) (ya:0.8464, yb:0.7056, yc:0.4624)
step: (a: -0.8400, b:-0.6800, c:-0.3600) (ya:0.7056, yb:0.4624, yc:0.1296)
step: (a: -0.6800, b:-0.3600, c:0.2800) (ya:0.4624, yb:0.1296, yc:0.0784)
step: (a: -0.3600, b:0.2800, c:1.5600) (ya:0.1296, yb:0.0784, yc:2.4336)
(-0.35999999999999993, 1.56)

x=1일때
init: (a:1.0000, b:1.0100) (ya:1.0000, yb:1.0201)
step: (a: 1.0100, b:1.0000, c:0.9900) (ya:1.0201, yb:1.0000, yc:0.9801)
step: (a: 1.0000, b:0.9900, c:0.9700) (ya:1.0000, yb:0.9801, yc:0.9409)
step: (a: 0.9900, b:0.9700, c:0.9300) (ya:0.9801, yb:0.9409, yc:0.8649)
step: (a: 0.9700, b:0.9300, c:0.8500) (ya:0.9409, yb:0.8649, yc:0.7225)
step: (a: 0.9300, b:0.8500, c:0.6900) (ya:0.8649, yb:0.7225, yc:0.4761)
step: (a: 0.8500, b:0.6900, c:0.3700) (ya:0.7225, yb:0.4761, yc:0.1369)
step: (a: 0.6900, b:0.3700, c:-0.2700) (ya:0.4761, yb:0.1369, yc:0.0729)
step: (a: 0.3700, b:-0.2700, c:-1.5500) (ya:0.1369, yb:0.0729, yc:2.4025)
(-1.55, 0.36999999999999994)
```

구간 탐색법

- 삼분할 탐색법

찾은 구간 폭 줄이기.



- idea) 구간 2개로 나눠 나눠진 구간이 unimodality 만족하는지 비교
  → 계산 불안정 → 3개의 구간으로 나누자!
- 구간의 폭의 1/3 씩 줄어든다. 적절한 폭 가지면 계산 stop.

⇒구간 탐색법 아이디어는 대체로 삼분할 법 이용

폭 빨리 줄어드는 대신, 계산량 많아짐.

¥ 정리

: bracket - minimum 통해 unimodality 만족하는 구간 결정

구간 3등분으로 나눠 목적함수 값 결정해 비교후

오른쪽 / 왼쪽 구간 버림 → 반복!

삼분할 탐색법

- 최적점 구하기

```
def trifold_search(f,x,epsilon=1E-6): #epsilon은 적절한 구간 폭
    a,b =bracket_minimum(f,x)
    print('init: (a:%.4f, b:%.4f)' %(a,b))

    distance=abs(a-b)

    i=1
    while distance > epsilon:
        x1=a+(1.0/3.0)*distance #다른 방법으로도 계산 가능!
        x2=a+(2.0/3.0)*distance

        y1,y2 =f(x1),f(x2)

        if y1>y2:
            a,b=x1,b # 구간 폭 1/3만큼 버림
        else:
            a,b =a,x2


        distance =abs(a-b)

        print('%d: (a:%.4f, b:%.4f)' %(i,a,b))

        i+=1
        # whiile loop 끝나면 a,b의 구간의 폭은 epsilon보다 작을 것

    x=0.5*abs(a-b) #가운데 값을 중앙값 계산하듯이 최적점의 근사값이라고 하자
    y=f(x) #목적함수의 최소값

    return x,y

print('x=-1일때')
print(trifold_search(f,-1))
print(' ')
print('x=1일때')
print(trifold_search(f,1))
```

↗ 엄청 작은값. 0에 가까운 최적점·목적함수.

```
x=-1일때
init: (a:-1.0000, b:-0.9900) (ya:1.0000, yb:0.9801)
step: (a:-1.0000, b:-0.9900, c:-0.9800) (ya:1.0000, yb:0.9801, yc:0.9604)
step: (a:-0.9900, b:-0.9800, c:-0.9600) (ya:0.9801, yb:0.9604, yc:0.9216)
step: (a:-0.9800, b:-0.9600, c:-0.9200) (ya:0.9604, yb:0.9216, yc:0.8464)
step: (a:-0.9600, b:-0.9200, c:-0.8400) (ya:0.9216, yb:0.8464, yc:0.7056)
step: (a:-0.9200, b:-0.8400, c:-0.6800) (ya:0.8464, yb:0.7056, yc:0.4624)
step: (a:-0.8400, b:-0.6800, c:-0.3600) (ya:0.7056, yb:0.4624, yc:0.1296)
step: (a:-0.6800, b:-0.3600, c:0.2800) (ya:0.4624, yb:0.1296, yc:0.0784)
step: (a:-0.3600, b:0.2800, c:1.5600) (ya:0.1296, yb:0.0784, yc:2.4336)
init: (a:-0.3600, b:1.5600)
1: (a:-0.3600, b:0.9200)
2: (a:-0.3600, b:0.4933)
3: (a:-0.3600, b:0.2089)
4: (a:-0.1704, b:0.2089)
5: (a:-0.1704, b:0.0825)
6: (a:-0.0861, b:0.0825)
7: (a:-0.0299, b:0.0825)
8: (a:-0.0299, b:0.0450)
9: (a:-0.0299, b:0.0200)
10: (a:-0.0133, b:0.0200)
11: (a:-0.0133, b:0.0089)
12: (a:-0.0059, b:0.0089)
13: (a:-0.0059, b:0.0040)
14: (a:-0.0026, b:0.0040)
15: (a:-0.0026, b:0.0018)
16: (a:-0.0011, b:0.0018)
17: (a:-0.0011, b:0.0008)
18: (a:-0.0005, b:0.0008)
19: (a:-0.0005, b:0.0004)
20: (a:-0.0002, b:0.0004)
21: (a:-0.0002, b:0.0002)
22: (a:-0.0002, b:0.0001)
23: (a:-0.0001, b:0.0001)
24: (a:-0.0001, b:0.0000)
25: (a:-0.0000, b:0.0000)
26: (a:-0.0000, b:0.0000)
27: (a:-0.0000, b:0.0000)
28: (a:-0.0000, b:0.0000)
29: (a:-0.0000, b:0.0000)
30: (a:-0.0000, b:0.0000)
31: (a:-0.0000, b:0.0000)
32: (a:-0.0000, b:0.0000)
33: (a:-0.0000, b:0.0000)
34: (a:-0.0000, b:0.0000)
35: (a:-0.0000, b:0.0000)
36: (a:-0.0000, b:0.0000)
(4.395273524334858e-07, 1.9318429353719158e-13)
```

```
x=1일때
init: (a:1.0000, b:1.0100) (ya:1.0000, yb:1.0201)
step: (a:1.0100, b:1.0000, c:0.9900) (ya:1.0201, yb:1.0000, yc:0.9801)
step: (a:1.0000, b:0.9900, c:0.9700) (ya:1.0000, yb:0.9801, yc:0.9409)
step: (a:0.9900, b:0.9700, c:0.9300) (ya:0.9801, yb:0.9409, yc:0.8649)
step: (a:0.9700, b:0.9300, c:0.8500) (ya:0.9409, yb:0.8649, yc:0.7225)
step: (a:0.9300, b:0.8500, c:0.6900) (ya:0.8649, yb:0.7225, yc:0.4761)
step: (a:0.8500, b:0.6900, c:0.3700) (ya:0.7225, yb:0.4761, yc:0.1369)
step: (a:0.6900, b:0.3700, c:-0.2700) (ya:0.4761, yb:0.1369, yc:0.0729)
step: (a:0.3700, b:-0.2700, c:-1.5500) (ya:0.1369, yb:0.0729, yc:2.4025)
init: (a:-1.5500, b:0.3700)
1: (a:-0.9100, b:0.3700)
2: (a:-0.4833, b:0.3700)
3: (a:-0.1989, b:0.3700)
4: (a:-0.1989, b:0.1804)
5: (a:-0.0725, b:0.1804)
6: (a:-0.0725, b:0.0961)
7: (a:-0.0725, b:0.0399)
8: (a:-0.0350, b:0.0399)
9: (a:-0.0350, b:0.0149)
10: (a:-0.0184, b:0.0149)
11: (a:-0.0073, b:0.0149)
12: (a:-0.0073, b:0.0075)
13: (a:-0.0073, b:0.0026)
14: (a:-0.0040, b:0.0026)
15: (a:-0.0018, b:0.0026)
16: (a:-0.0018, b:0.0011)
17: (a:-0.0008, b:0.0011)
18: (a:-0.0008, b:0.0005)
19: (a:-0.0004, b:0.0005)
20: (a:-0.0004, b:0.0002)
21: (a:-0.0002, b:0.0002)
22: (a:-0.0002, b:0.0001)
23: (a:-0.0001, b:0.0001)
24: (a:-0.0000, b:0.0001)
25: (a:-0.0000, b:0.0000)
26: (a:-0.0000, b:0.0000)
27: (a:-0.0000, b:0.0000)
28: (a:-0.0000, b:0.0000)
29: (a:-0.0000, b:0.0000)
30: (a:-0.0000, b:0.0000)
31: (a:-0.0000, b:0.0000)
32: (a:-0.0000, b:0.0000)
33: (a:-0.0000, b:0.0000)
34: (a:-0.0000, b:0.0000)
35: (a:-0.0000, b:0.0000)
36: (a:-0.0000, b:0.0000)
(4.395273524334884e-07, 1.9318429353719196e-13)
```

- 피보나치 수열

$\therefore F_n = \begin{cases} 1 & (\text{if } n \le 2) \\ F_{n-1} + F_{n-2} & (\text{otherwise}) \end{cases}$

- 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 ···· (구간 폭 커짐, 수렴값 황금비율)
- a와 b 사이 값 피보나치 수열 이용해 1/3 보다 안쪽 버리면 구간 폭 더 빠른 속도로 줄어들 것. (그 때의 구간은 unimodality 만족)

- Binet's formula ← n번째 피보나치 수열값 계산 가능

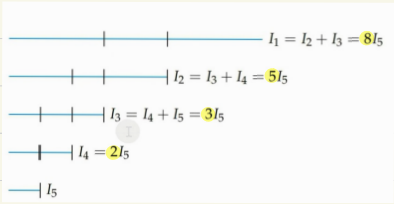
$\therefore F_n = \frac{\varphi^n - (1-\varphi)^n}{\sqrt{5}}$, $\varphi = \frac{1+\sqrt{5}}{2}$ = Golden ratio ≈ 1.4236

- 피보나치 수열의 비율

$\frac{F_n}{F_{n-1}} = \varphi \frac{1-s^{n+1}}{1-s^n}$

$-s = \frac{1-\sqrt{5}}{1+\sqrt{5}} \approx -0.382$ → 두 수열의 비율 만큼 구간을 왼/오 이동

- 피보나치 탐색법 : 일변량 탐색법



↳ 1, 2, 3, 5, 8 로 피보나치로 구간 줄여나가면서 최적점이 포함되어 있다고 생각되는 구간 결정

unimodality 만족하는 구간 결정. bracket_minimum 가지고도 가능.

ex) $f(x) = \exp(x-2) - x$, $[a,b] = [-2, 6]$

⊕ 초기 bracketing interval 길이에 따라 $\frac{F_7}{F_8}$, $1-\frac{F_5}{F_6}$ 만들어짐.

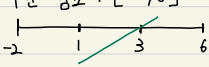
a에서 오른쪽으로 이동하는 양. ($F_5 = 5$, $F_6 = 8$)

① $f(x^{(1)}) = f(a + (b-a)(1 - \frac{F_5}{F_6})) = f(1) = -0.632$

$f(x^{(2)}) = f(a + (b-a)(\frac{F_5}{F_6})) = f(3) = -0.282$

↳ b값에서 이동하는 양

구간 감소 : $[-2, 6] \rightarrow [-2, 3]$



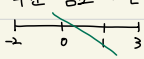
→ unimodality 만족

: f(3) 값이 더 크니까 상한 6→3 감소

② $x_{왼쪽} = a + (b-a)(1 - \frac{F_4}{F_5}) = 0$ → $f(0) = 0.135$

$x_{오른쪽} = a + (b-a) \cdot \frac{F_4}{F_5} = 1$ → $f(1) = -0.632$

구간 감소 : $[-2, 3] \rightarrow [0, 3]$

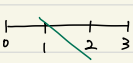


: f(0) 값이 크니까 하한 감소 (-2→0)

③ $x_{왼쪽} : a + (b-a)(1 - \frac{F_3}{F_4}) = 1$ → $f(1) = -0.632$

$x_{오른쪽} : a + (b-a) \cdot \frac{F_3}{F_4} = 2$ → $f(2) = -1$

구간 감소 : $[0, 3] \rightarrow [1, 3]$



: f(1) 값이 크니까 하한 감소 (0→1)

- 피보나치 탐색법 알고리즘

제일 마지막 구간 결정

```
function fibonacci_search(f, a, b, n; ϵ=0.01)
    s = (1-√5)/(1+√5)
    ρ = 1 / (φ*(1-s^(n+1))/(1-s^n))
    d = ρ*b + (1-ρ)*a
    yd = f(d)
    for i in 1 : n-1
        if i == n-1
            c = ϵ*a + (1-ϵ)*d
        else
            c = ρ*a + (1-ρ)*b
        end
        yc = f(c)
        if yc < yd
            b, d, yd = d, c, yc
        else
            a, b = b, c
        end
        ρ = 1 / (φ*(1-s^(n-i+1))/(1-s^(n-i)))
    end
    return a < b ? (a, b) : (b, a)
end
```

결정경계 ← → 피보나치 수열비 : $\frac{F_n}{F_{n-1}}$, $\frac{F_{n-1}}{F_n}$ 로 계산 후 역수 취함.

피보나치 탐색법

- 구간을 더 빨리 줄이고 싶어. 계산은 복잡해짐
- 생각해낸 것은 황금비율!! --> 피보나치 수열

```
import numpy as np

def fibonacci_search(f,x,n,epsilon=1E-2):
    a,b=bracket_minimum(f,x) #구간의 상한 하한 결정하는 bracket_minimum 알고리즘 활용
    print('init: (a:%.4f, B:%.4f)' % (a,b))

    psi=0.5*(1.+np.sqrt(5))
    s=(1.-np.sqrt(5))/(1.+np.sqrt(5))

    rho=1./psi*((1.-s**(n+1))/(1.-s**n)) #n은 1,1,2,3,5,8에서 몇번째인지 결정
    d=rho*b+(1.-rho)*a

    yd=f(d)

    for i in range(1,n):
        if i==n-1:
            c=epsilon*a+(1.-epsilon)*d
        else:
            c=rho*a+(1.-rho)*d
        yc=f(c)

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

        else:
            a,b=b,c

        rho=1./psi*((1.-s**(n-i+1)))/(1.-s**(n-i))

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

    a,b=(a,b) if a<b else (b,a)

    x=0.5*abs(a-b)
    y=f(x)

    return x,y


print('n=30')
print(fibonacci_search(f,-1,30))
print('')
print('n=50')
print(fibonacci_search(f,-1,50))
```

29번 반복

더 정확한 값. but 삼분할법보다 반복횟수 多

수학적으로, 삼분할법의 경우, Unimodality 만족한다 해도 최적점 못구하고 사이로 발생가능성
→피보나치 사용.

```
n=30
init: (a:-1.0000, b:-0.9900) (ya:1.0000, yb:0.9801)
step: (a: -1.0000, b:-0.9900, c:-0.9800) (ya:1.0000, yb:0.9801, yc:0.9604)
step: (a: -0.9900, b:-0.9800, c:-0.9600) (ya:0.9801, yb:0.9604, yc:0.9216)
step: (a: -0.9800, b:-0.9600, c:-0.9200) (ya:0.9604, yb:0.9216, yc:0.8464)
step: (a: -0.9600, b:-0.9200, c:-0.8400) (ya:0.9216, yb:0.8464, yc:0.7056)
step: (a: -0.9200, b:-0.8400, c:-0.6800) (ya:0.8464, yb:0.7056, yc:0.4624)
step: (a: -0.8400, b:-0.6800, c:-0.3600) (ya:0.7056, yb:0.4624, yc:0.1296)
step: (a: -0.6800, b:-0.3600, c:0.2800) (ya:0.4624, yb:0.1296, yc:0.0784)
step: (a: -0.3600, b:0.2800, c:1.5600) (ya:0.1296, yb:0.0784, yc:2.4336)
init: (a:-0.3600, B:1.5600)
1:(a:-0.3600, b:0.8266)
2:(a:-0.1869, b:0.8266)
3:(a:-0.1869, b:0.5465)
4:(a:-0.1869, b:0.0933)
5:(a:-0.1460, b:0.0933)
6:(a:-0.0799, b:0.0933)
7:(a:-0.0799, b:0.0680)
8:(a:-0.0390, b:0.0680)
9:(a:-0.0390, b:0.0524)
10:(a:-0.0390, b:0.0271)
11:(a:-0.0294, b:0.0271)
12:(a:-0.0137, b:0.0271)
13:(a:-0.0137, b:0.0212)
14:(a:-0.0137, b:0.0115)
15:(a:-0.0101, b:0.0115)
16:(a:-0.0101, b:0.0055)
17:(a:-0.0078, b:0.0055)
18:(a:-0.0041, b:0.0055)
19:(a:-0.0041, b:0.0041)
20:(a:-0.0041, b:0.0019)
21:(a:-0.0032, b:0.0019)
22:(a:-0.0018, b:0.0019)
23:(a:-0.0018, b:0.0013)
24:(a:-0.0009, b:0.0013)
25:(a:-0.0009, b:0.0010)
26:(a:-0.0009, b:0.0005)
27:(a:-0.0008, b:0.0005)
28:(a:-0.0004, b:0.0005)
29:(a:-0.0004, b:0.0001)
(0.0002478044132501504, 6.140702722625132e-08)
```

```
n=50
init: (a:-1.0000, b:-0.9900) (ya:1.0000, yb:0.9801)
step: (a: -1.0000, b:-0.9900, c:-0.9800) (ya:1.0000, yb:0.9801, yc:0.9604)
step: (a: -0.9900, b:-0.9800, c:-0.9600) (ya:0.9801, yb:0.9604, yc:0.9216)
step: (a: -0.9800, b:-0.9600, c:-0.9200) (ya:0.9604, yb:0.9216, yc:0.8464)
step: (a: -0.9600, b:-0.9200, c:-0.8400) (ya:0.9216, yb:0.8464, yc:0.7056)
step: (a: -0.9200, b:-0.8400, c:-0.6800) (ya:0.8464, yb:0.7056, yc:0.4624)
step: (a: -0.8400, b:-0.6800, c:-0.3600) (ya:0.7056, yb:0.4624, yc:0.1296)
step: (a: -0.6800, b:-0.3600, c:0.2800) (ya:0.4624, yb:0.1296, yc:0.0784)
step: (a: -0.3600, b:0.2800, c:1.5600) (ya:0.1296, yb:0.0784, yc:2.4336)
init: (a:-0.3600, B:1.5600)
1:(a:-0.3600, b:0.8266)
2:(a:-0.1869, b:0.8266)
3:(a:-0.1869, b:0.5465)
4:(a:-0.1869, b:0.0933)
5:(a:-0.1460, b:0.0933)
6:(a:-0.0799, b:0.0933)
7:(a:-0.0799, b:0.0680)
8:(a:-0.0390, b:0.0680)
9:(a:-0.0390, b:0.0524)
10:(a:-0.0390, b:0.0271)
11:(a:-0.0294, b:0.0271)
12:(a:-0.0137, b:0.0271)
13:(a:-0.0137, b:0.0212)
14:(a:-0.0137, b:0.0115)
15:(a:-0.0101, b:0.0115)
16:(a:-0.0101, b:0.0055)
17:(a:-0.0078, b:0.0055)
18:(a:-0.0041, b:0.0055)
19:(a:-0.0041, b:0.0041)
20:(a:-0.0041, b:0.0019)
21:(a:-0.0032, b:0.0019)
22:(a:-0.0018, b:0.0019)
23:(a:-0.0018, b:0.0013)
24:(a:-0.0010, b:0.0013)
25:(a:-0.0010, b:0.0010)
26:(a:-0.0010, b:0.0005)
27:(a:-0.0007, b:0.0005)
28:(a:-0.0004, b:0.0005)
29:(a:-0.0004, b:0.0003)
30:(a:-0.0002, b:0.0003)
31:(a:-0.0002, b:0.0003)
32:(a:-0.0002, b:0.0001)
33:(a:-0.0002, b:0.0001)
34:(a:-0.0001, b:0.0001)
35:(a:-0.0001, b:0.0001)
36:(a:-0.0001, b:0.0000)
37:(a:-0.0001, b:0.0000)
38:(a:-0.0000, b:0.0000)
39:(a:-0.0000, b:0.0000)
40:(a:-0.0000, b:0.0000)
41:(a:-0.0000, b:0.0000)
42:(a:-0.0000, b:0.0000)
43:(a:-0.0000, b:0.0000)
44:(a:-0.0000, b:0.0000)
45:(a:-0.0000, b:0.0000)
46:(a:-0.0000, b:0.0000)
47:(a:-0.0000, b:0.0000)
48:(a:-0.0000, b:0.0000)
49:(a:-0.0000, b:0.0000)
(2.0148032101375695e-06, 4.059431975581161 6e-12)
```