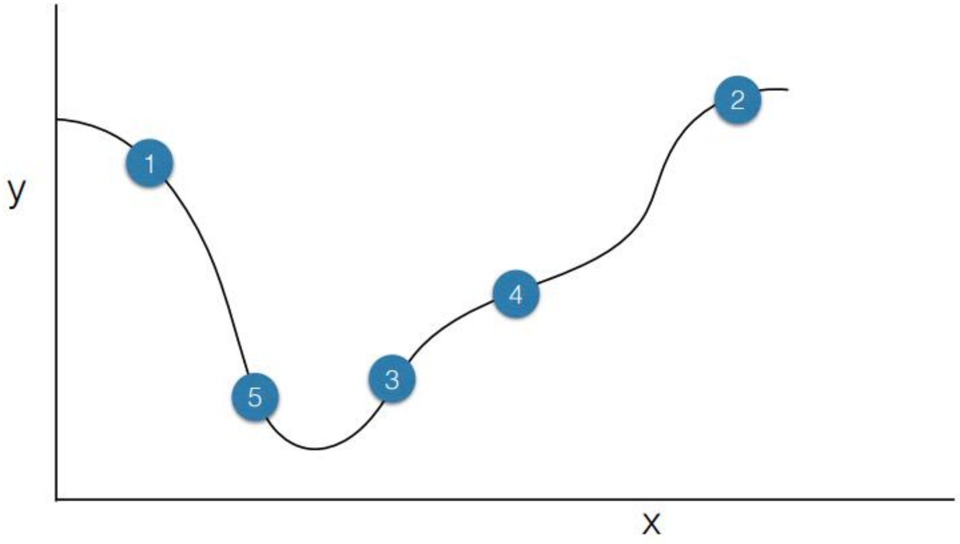
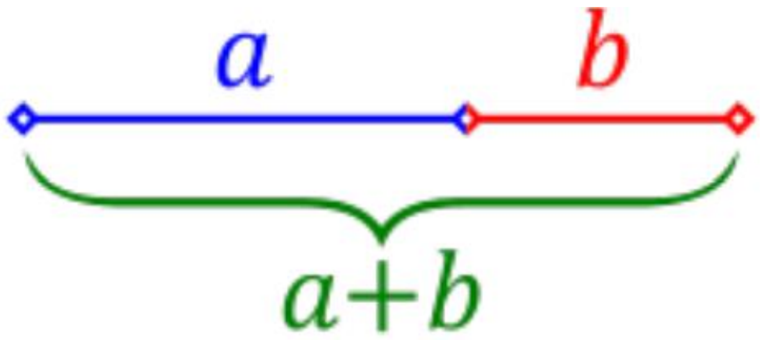
Minimization

2017135002/최성윤

기본원리

Line Minimization





a=0.618 b=0.382

1. 임의의 점 1, 2 선택
2. golden ratio로 점 내분 -> 4, 5
3. 이중 최소값을 가지는 점 선택 -> 5
4. 반복

실행

```python
def f1(x):
    y1=( np.sin(x)**2 ) * np.cos(x/4) + 4*(x**4) - 6*(x**3) - 2*x + 6
    return y1
def f2(x):
    y2=-( (np.cos(x/2)**2)/(x**2+5) )-2*np.exp( - ( ( (x-3)**2)/100))
    return y2
```

Line minimization을 실행할 함수 설정

```python
def  opt(x1,x2):
    if x1>x2:
        return [x2,x2*0.61803+x1*0.38197,x2*0.38197+x1*0.61803,x1]
    else:
        return [x1,x2*0.38197+x1*0.61803,x2*0.61803+x1*0.38197, x2]
```

설정한 2개의 점에 대하여
Golden ration로 내분한 점을
포함하는 list 반환

```python
def lm2(x1,x2):
    X=opt(x1,x2)
    if abs(x1-x2)<0.000001:
        return [x1,x2]
    if f2(X[1]) < f2(X[2]):
        return lm2(x1,X[2])
    else:
        return lm2(X[1],x2)
```

초기조건 (x1 x2 사이가 0.000001 이하)이
만족되면 종료되도록 재귀함수 설정

결과

```
print(lm1(-5,5))
print(lm2(-5,5))
```

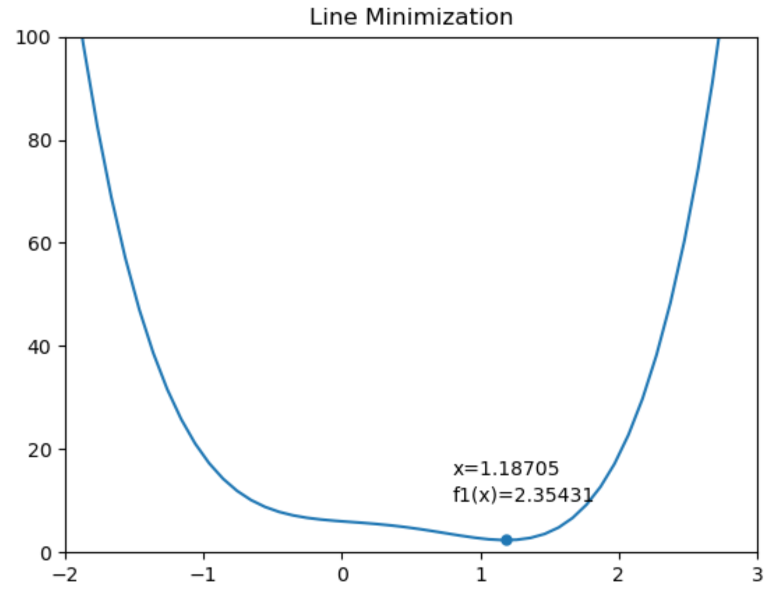
초기 양 끝점을 x=-5, +5로 설정후
Line minimization 진행

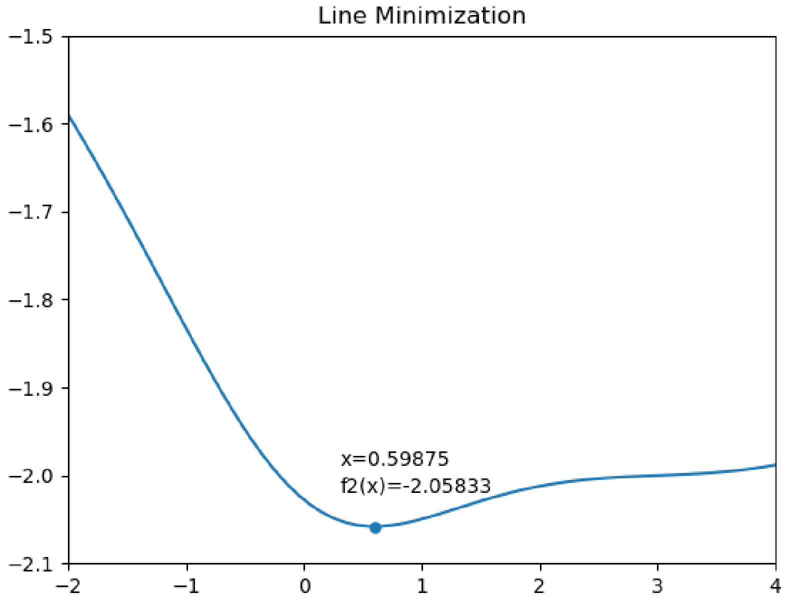
$$f(x) = \sin^2 x \cos\frac{x}{4} + 4x^4 - 6x^3 - 2x + 6$$

```
[1.187046612009283, 1.187047396025295]
```

$$f(x) = -\frac{\cos^2\frac{x}{2}}{x^2+5} - 2e^{-\frac{(x-3)^2}{100}}$$

```
[0.5987535466019636, 0.5987543306179754]
```





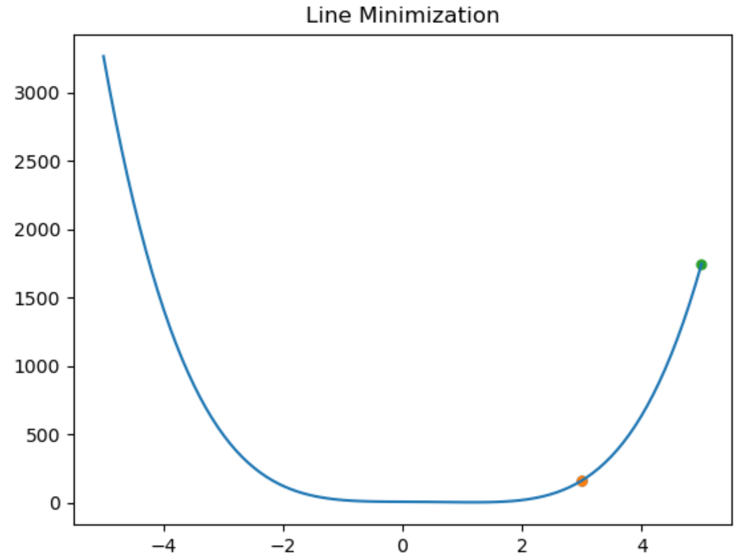
토의사항

처음 설정한 2개의 점 내부에 Global Miniminum이 존재하지 않는다면?

$$f(x) = \sin^2 x \cos\frac{x}{4} + 4x^4 - 6x^3 - 2x + 6$$

```
print(lm1(3,5))
```

```
[3, 3.0000006642418846]
```



Global Miniminum 지점으로 최소화가 되지 않는다

2개의 점을 Golden ration 외분한 점도 고려

```
def  exopt(x1,x2):
    if x1>x2:
        return [ x2*(1/0.61803)-(x1*0.38197)/0.61803,x2,x2*0.61803+x1*0.38197,
                 x2*0.38197+x1*0.61803,x1,x1*(1/0.61803)-(x2*0.38197)/0.61803]
    else:
        return [x1*(1/0.61803)-(x2*0.38197)/0.61803,x1,x2*0.38197+x1*0.61803,
                x2*0.61803+x1*0.38197, x2, x2*(1/0.61803)-(x1*0.38197)/0.61803]

def lm4(x1,x2):
    X=exopt(x1,x2)
    c=[f2(X[0]),f2(X[1]),f2(X[2]),f2(X[3]),f2(X[4]),f2(X[5])]
    if abs(x1-x2)<0.000001:
        return [x1,x2]
    if min(c)==f2(X[0]):
        return lm4(X[0],X[1])
    elif min(c)==f2(X[1]):
        return lm4(X[0],X[1])
    elif min(c)==f2(X[2]):
        return lm4(X[1],X[2])
    elif min(c)==f2(X[3]):
        return lm4(X[3],X[4])
    elif min(c)==f2(X[4]):
        return lm4(X[4],X[5])
    elif min(c)==f2(X[5]):
        return lm4(X[4],X[5])
```

토의사항

처음 설정한 2개의 점 내부에 Global Miniminum이 존재하지 않는다면?

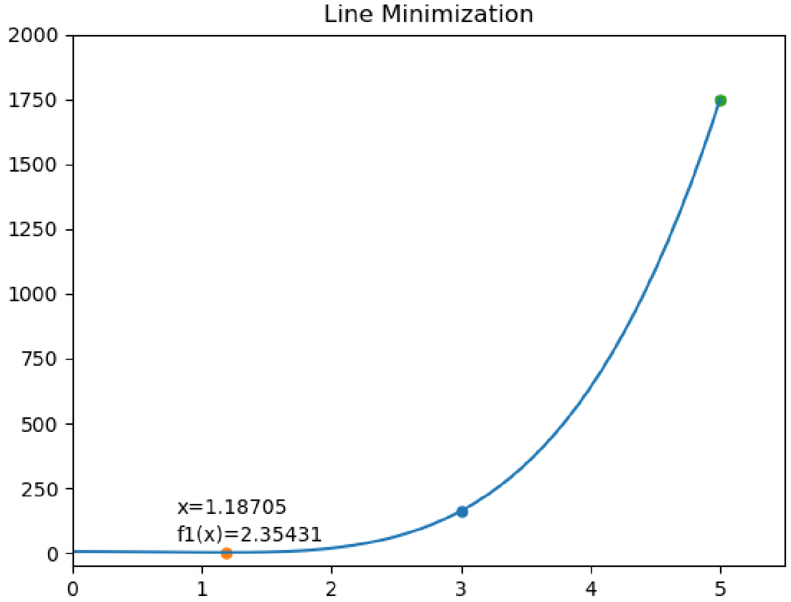
```
print(lm3(3,5))
print(lm4(3,5))
```

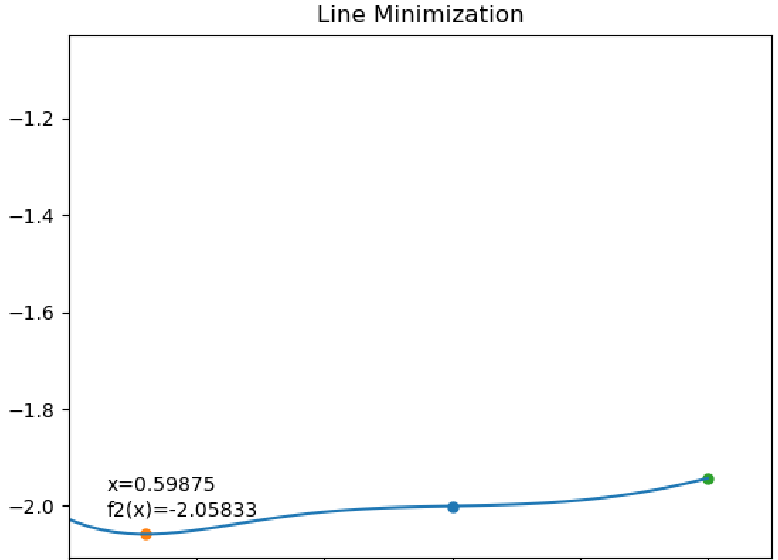
$$f(x) = \sin^2 x \cos \frac{x}{4} + 4x^4 - 6x^3 - 2x + 6$$

```
[1.1870466498621761, 1.1870473144608962]
```

$$f(x) = -\frac{\cos^2 \frac{x}{2}}{x^2 + 5} - 2e^{-\frac{(x-3)^2}{100}}$$

```
[0.5987535132925724, 0.5987539240632936]
```





토의사항

Local Minimum이 여러 곳인 함수? (1차 미분값이 0인 지점이 여러 곳)

임의의 4차함수 생성후 Local Minimization 실행

```
def f3(x):
    return (x-2)*(3*x+1)*(x-4)*(x+3)
```

```
print(lm5(-3,4))
```

점 x=-3, 4 에 대하여 실행

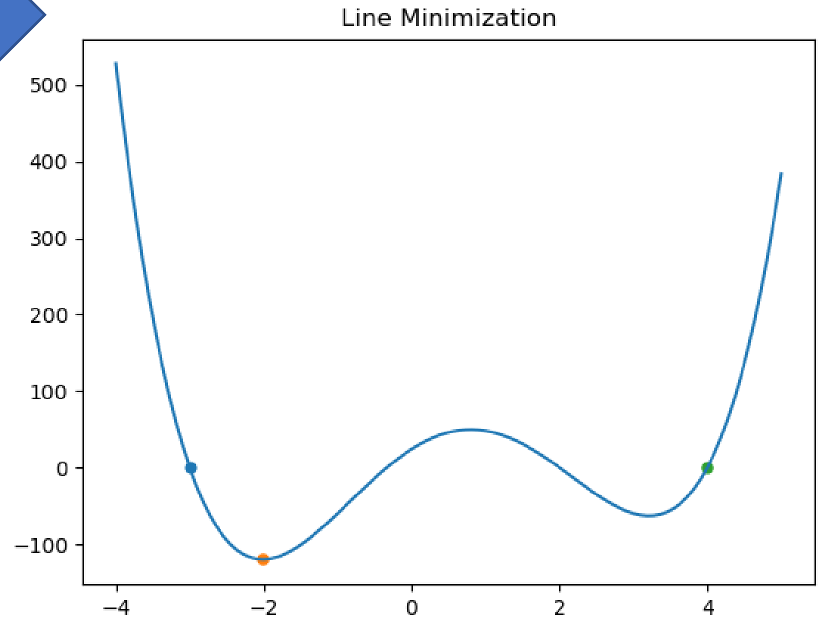
```
[-2.011422111223266, -2.011421640484014б]
```

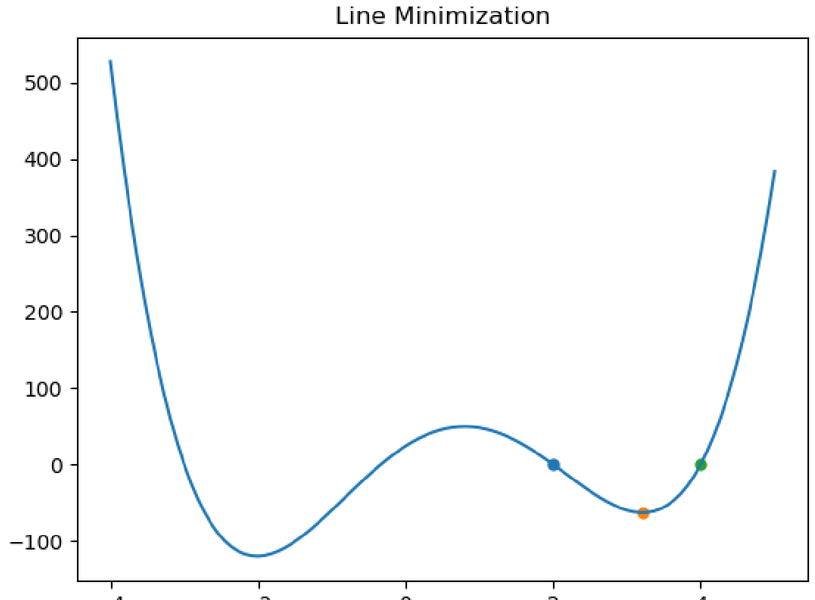
```
print(lm5(2,4))
```

점 x=2, 4 에 대하여 실행

```
[3.2116183724528886, 3.2116187831852447]
```

점 x=2 ,4 에 대해서는 그 사이에 있는 Local Minimum지점을 Global Minimum으로 착각하여 Minimization 된다





기본원리

Singular Isothermal Sphere Profile

$$\rho(r) = \frac{\sigma_v^2}{2\pi G r^2} = \frac{\rho_0}{r^2}$$

변수 : ρ_0

NFW Profile

$$\rho(r) = \frac{\rho_0}{r/r_s(1+r/r_s)^2}$$

변수 : ρ_0 , r_s

옆의 식에 대한
Chi-Square 값에 대하여
2가지 방법으로
Minimization 실행

1.Downhill

2.Direction Set Method

- Save your starting position as $\mathbf{P}_0$.
- For $i = 1, \ldots, N$, move $\mathbf{P}_{i-1}$ to the minimum along direction $\mathbf{u}_i$ and call this point $\mathbf{P}_i$.
- For $i = 1, \ldots, N-1$, set $\mathbf{u}_i \leftarrow \mathbf{u}_{i+1}$.
- Set $\mathbf{u}_N \leftarrow \mathbf{P}_N - \mathbf{P}_0$.
- Move $\mathbf{P}_N$ to the minimum along direction $\mathbf{u}_N$ and call this point $\mathbf{P}_0$.

실행

```
sisdir=minimize(sis,d[0],method='Powell')
sisdown=minimize(sis,d[0],method='Nelder-Mead')
nfxdir=minimize(nfx,[d[0],r[0]],method='Powell')
nfxdown=minimize(nfx,[d[0],r[0]],method='Nelder-Mead')
```

결과

SIS profile (Direction Set Method)

```
  direc: array([[1.0081065e-07]])
    fun: 257.7903385321215
message: 'Optimization terminated successfully.'
   nfev: 68
    nit: 2
 status: 0
success: True
      x: array([37.92951826])
```

SIS profile (Downhill)

```
final_simplex: (array([[37.92950863],
       [37.92957643]]), array([257.79033853, 257.79033854]))
          fun: 257.79033853224615
      message: 'Optimization terminated successfully.'
         nfev: 42
          nit: 21
       status: 0
      success: True
            x: array([37.92950863])
```

NFW profile(Direction Set Method)

```
  direc: array([[-0.43665869,  1.48244308],
       [-0.19469277,  1.0403165 ]])
    fun: 119.90190394060187
message: 'Optimization terminated successfully.'
   nfev: 588
    nit: 19
 status: 0
success: True
      x: array([ 1.10815929, 14.92080317])
```

NFW profile(Downhill)

```
final_simplex: (array([[ 1.10739587, 14.92835456],
       [ 1.10740897, 14.928279  ],
       [ 1.1073882 , 14.92838298]]), array([119.9012560
          fun: 119.90125601862249
      message: 'Optimization terminated successfully.'
         nfev: 161
          nit: 86
       status: 0
      success: True
            x: array([ 1.10739587, 14.92835456])
```

토의사항

SIS profile (Direction Set Method)의 ρ_0

37.92951826

소요시간 t `0.001995086669921875`

SIS profile (Downhill)의 ρ_0

37.92950863

소요시간 t `0.0019948482513427734`

NFW profile(Direction Set Method)의 ρ_0 , r_s

1.10815929 / 14.92080317

소요시간 t `0.02098536491394043`

NFW profile(Downhill)의 ρ_0 , r_s

1.10739587 / 14.92835456

소요시간 t `0.005984067916870117`

동일한 Profile 내에서 2가지 방법에 대한 결과의 차이는 거의 없다.
하지만 소요시간은 매번 다르지만 대략 위와 같은 시간이 걸리는 것을 알 수 있다.
NFW에서 SIS에 비해 소요시간이 더 길었는데 이는 변수 2개를 사용해서 발생한 걸로 추측할수 있다,

SIS profile의 경우 2개의 방법 다 수행속도가 비슷 하지만
NFW profile의 경우 Downhill을 사용 하였을때 수행속도가 더 빨랐다.

```
nfev: 588
 nit: 19
```

```
nfev: 161
 nit: 86
```

위에는 Direction, 밑에는 Downhill의 경우에
nfev, nit 결과값을 나타내었다. 여기서
nfev는 Number of evaluations of the objective
nit는 Number of iterations performed by the optimizer
를 나타내는데 이 횟수가 수행속도에 영향을 미쳤음을 추측할수 있다.

Direction에서 nfev+ nfit이
크기 때문에 소요시간이 더 길다

토의사항

각각 Profile에 대하여 2개의 방법으로 Minimization 결과의 평균을 바탕으로
Uncertainty 와 Reduced Chi- square을 구하였다.

SIS profile의 Reduced Chi- square

```
2.6039428134560803
```

NFW profile의 Reduced Chi- square

```
1.2234888157204273
```

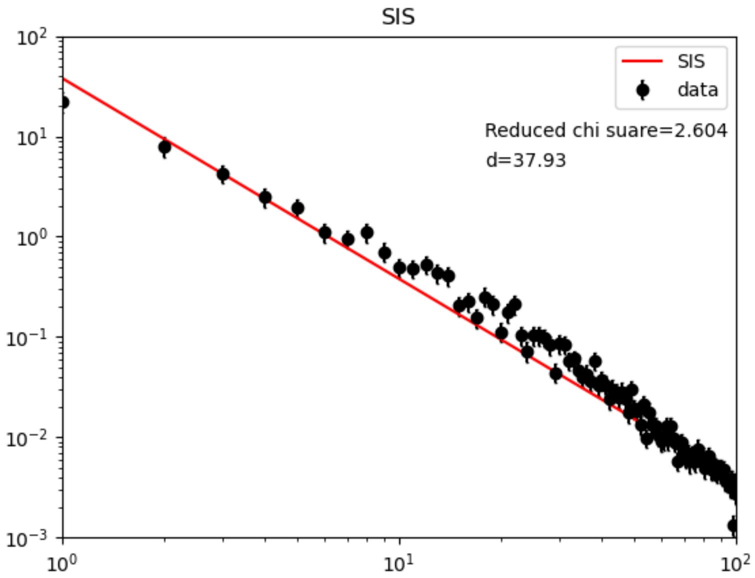
NFW profile의 Reduced Chi-square가 더 1에 근접하므로
더 Data에 적합한 모델이다.

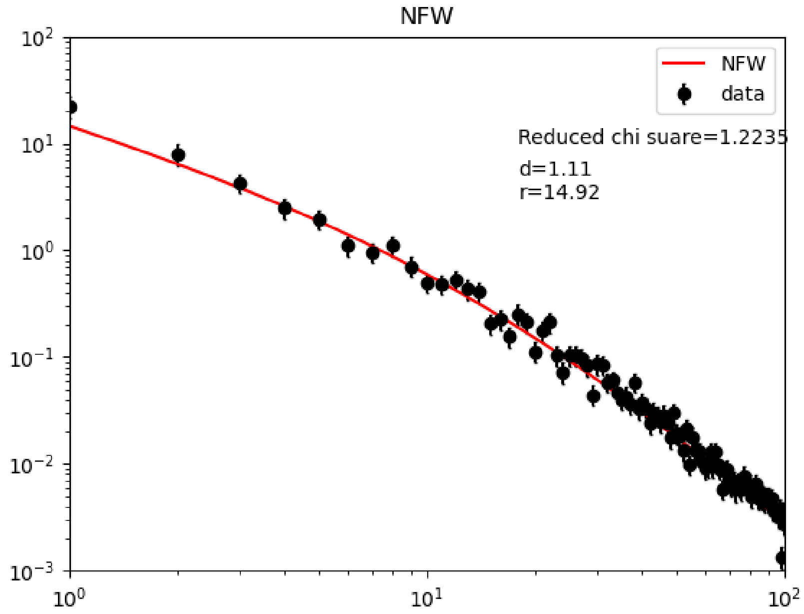
SIS profile의 ρ_0 Uncertainty

```
0.5991862857142857
```

NFW profile의 ρ_0, r_s Uncertainty

```
0.017062003006405518
0.09476384781491361
```





Reference

1.Scipy.optimize.minimize function to determine multiple variables

https://stackoverflow.com/questions/48038562/scipy-optimize-minimize-function-to-determine-multiple-variables

2.scipy.optimize.OptimizeResult

https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.optimize.OptimizeResult.html

3.Understanding the output of scipy.optimize.basinhopping

https://stackoverflow.com/questions/27728483/understanding-the-output-of-scipy-optimize-basinhopping