非線形回帰

青木繁伸

2020 年 3 月 17 日

1　目的

各種関数に非線形回帰によりデータを当てはめる。
事前に用意されている関数以外の関数に当てはめる方法については，3.15 節に記述する。

2　使用法

```
import sys
sys.path.append("statlib")
from multi import nonlinear_fitting
```

2.1　引数

x	独立変数（リストでもよい）
y	従属変数（リストでもよい）
init	パラメータの初期値（リストでもよい）
model	表 1 に示すモデル名（デフォルトは "Asymptotic"）
method	計算手法。デフォルトでマルカート法（"marquardt"），他にシンプレックス法（"simplex"）を指定できる。
trace	True のとき収束過程を表示する（デフォルトは False）
maxit	収束計算上限回数（デフォルトは 1000）
epsilon	収束判定条件（デフォルトは 1e-7）
verbose	必要最小限のプリント出力をする（デフォルトは True）。

2.2　戻り値の名前

"p"	パラメータ
"predicted"	観察値に対する予測値
"resid"	予測誤差
"x"	独立変数
"y"	従属変数
"model"	モデル関数名

"function"　　モデル関数

表 1　model に指定するもの

model	function
"Power1"	$y = ax^b$
"Power2"	$y = ax^b + c$
"Exponential1"	$y = ab^x$
"Exponential2"	$y = a(1 - \exp(-bx))$
"Asymptotic"	$y = ab^x + c$
"Logistic1"	$y = a/(1 + b\exp(-cx))$
"Logistic2"	$y = a/(1 + b\exp(-cx)) + d$
"Logostic3"	$y = 1/(1 + a\exp(-bx))$
"Logistic4"	$y = a + \dfrac{b-a}{1+(c/x)^d}$
"Gomperts"	$y = ab^{\exp(-cx)}$
"Weibull"	$y = 1 - \exp(-x^a/b)$
"Sin"	$y = a\sin(bx + c) + dx + e$
"Hyperbola1"	$y = a + \dfrac{b}{x \dotplus d} c$
"Hyperbola2"	$y = \dfrac{}{bx^2 + cx + d}$
関数名	ユーザが定義する関数。使用例参照のこと。

3　使用例

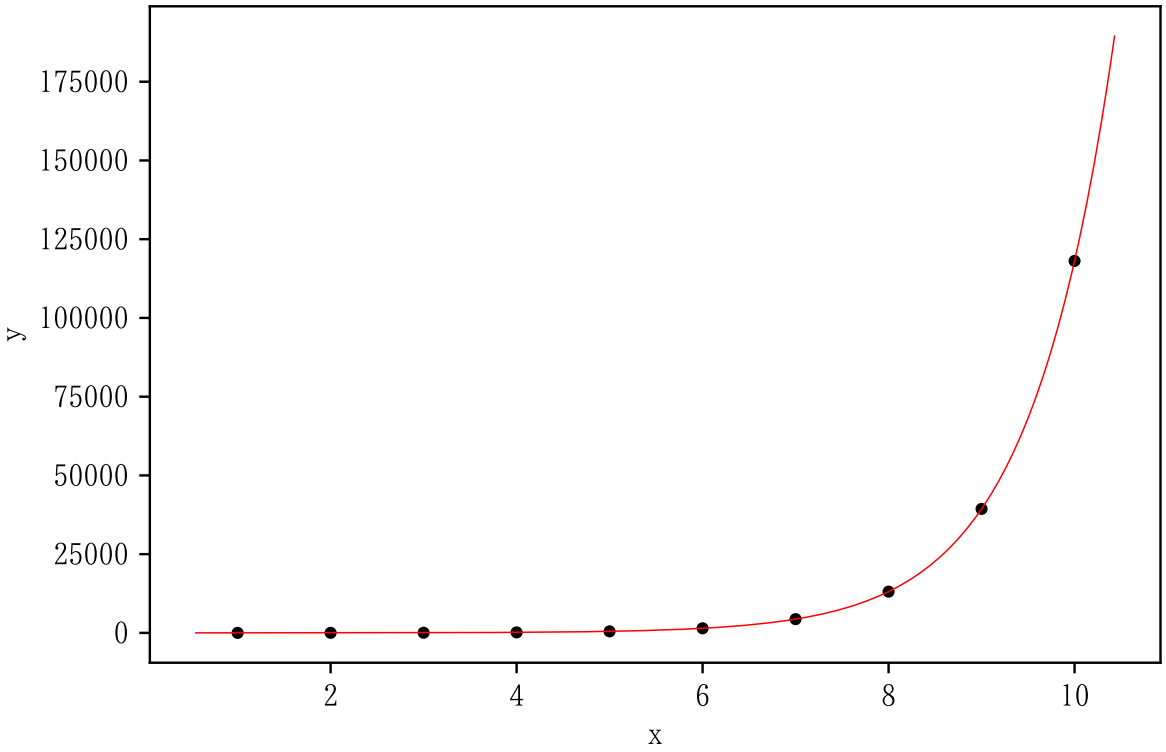
```
import sys
sys.path.append("statlib")
from multi import nonlinear_fitting

import matplotlib.pyplot as plt

def graph(x, y, a):
        x0 = np.min(x)
        x1 = np.max(x)
        delta = (x1-x0)*0.05
        x2 = np.arange(x0-delta, x1+delta, (x1-x0+2*delta)/500)
        y2 = a["function"](a["p"], x2)
        plt.scatter(x, y, c="black", s=9)
        plt.plot(x2, y2, linewidth=0.5, color="red")
        plt.xlabel("x")
        plt.ylabel("y")
        plt.show()
```

3.1 model = "Exponential1"

```
import numpy as np

x = np.arange(1, 11)
y = np.array([6, 18, 54, 162, 486, 1458, 4374, 13102, 39366, 118098])
```

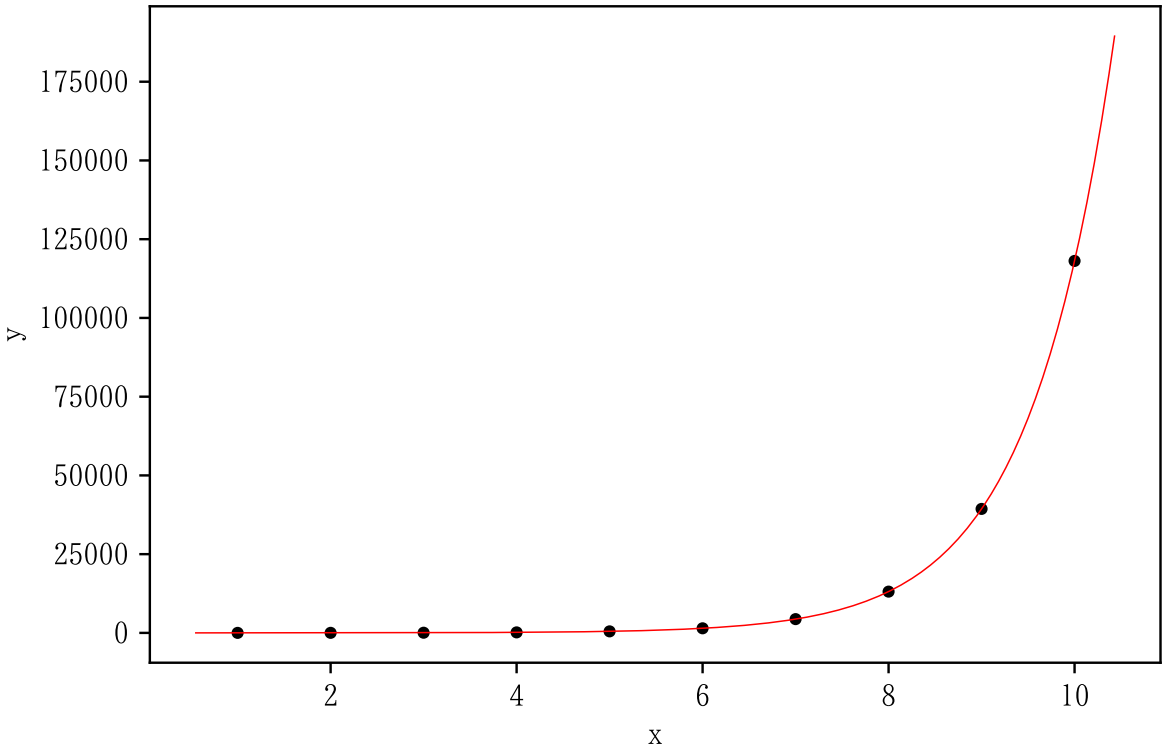
```
np.random.seed(1234) # 通常は指定しない
inival = [1, 1]
a = nonlinear_fitting(x, y, inival, model="Exponential1", method="
    simplex")
graph(x, y, a)
```

```
Exponential1      by Simplex method

                          estimates
a                          1.995568
b                          3.000669
residual sum of squares  285.884500

      x         y          pred.     resid.
0   1.0       6.0       5.988038   0.011962
1   2.0      18.0      17.968120   0.031880
2   3.0      54.0      53.916382   0.083618
3   4.0     162.0     161.785215   0.214785
4   5.0     486.0     485.463878   0.536122
5   6.0    1458.0    1456.716410   1.283590
6   7.0    4374.0    4371.123775   2.876225
7   8.0   13102.0   13116.295609 -14.295609
8   9.0   39366.0   39357.661635   8.338365
9  10.0  118098.0  118099.315200  -1.315200
```




```
b = nonlinear_fitting(x, y, a["p"], model="Exponential1")
graph(x, y, b)
```

```
Exponential1      by Marquardt method

                          estimates
a                          1.995567
b                          3.000669
residual sum of squares  285.884495

      x         y          pred.     resid.
0   1.0       6.0       5.988036   0.011964
1   2.0      18.0      17.968114   0.031886
2   3.0      54.0      53.916365   0.083635
3   4.0     162.0     161.785172   0.214828
4   5.0     486.0     485.463773   0.536227
5   6.0    1458.0    1456.716161   1.283839
6   7.0    4374.0    4371.123226   2.876774
7   8.0   13102.0   13116.294561 -14.294561
8   9.0   39366.0   39357.660290   8.339710
9  10.0  118098.0  118099.316561  -1.316561
```



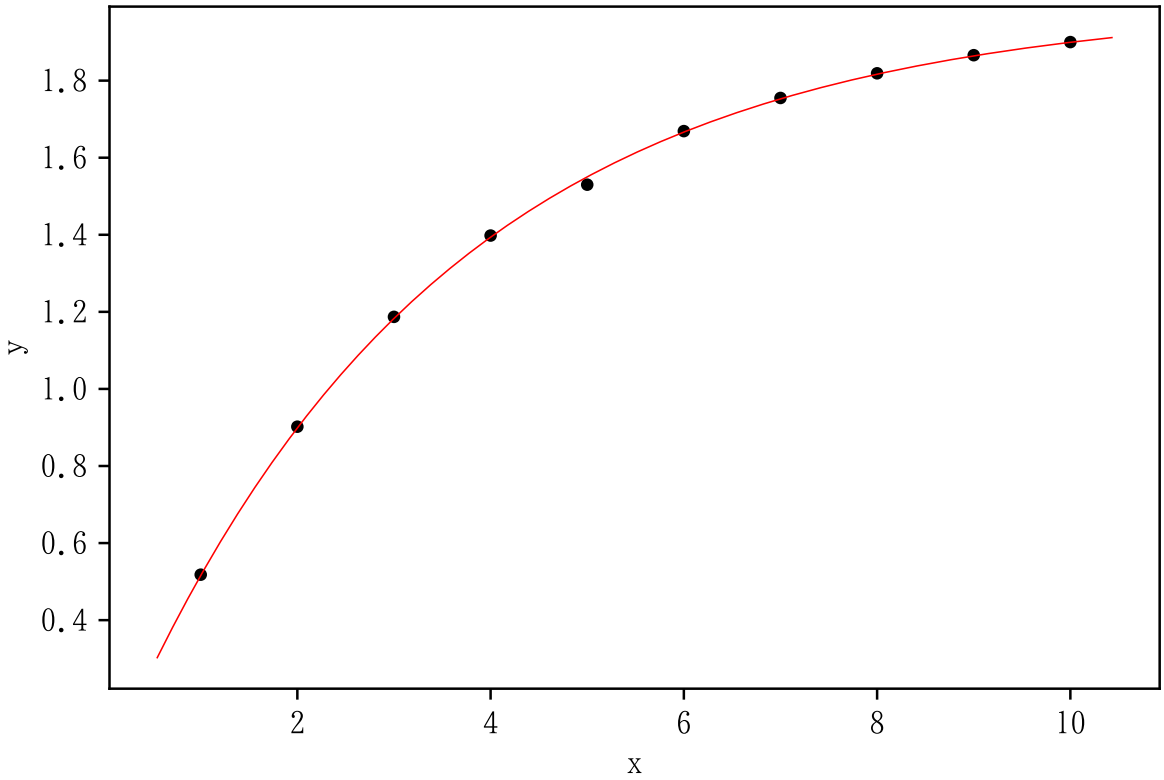
3.2 model = "Exponential2"

```
import numpy as np

x = np.arange(1, 11)
y = np.array([0.518, 0.902, 1.187, 1.398, 1.530, 1.669, 1.755, 1.819,
    1.866, 1.900])
```

```
np.random.seed(1234) # 通常は指定しない
inival = [1, 1]
a = nonlinear_fitting(x, y, inival, model="Exponential2", method="
    simplex")
graph(x, y, a)
```

```
Exponential2      by Simplex method

                          estimates
a                          2.000921
b                          0.298158
residual sum of squares    0.000483

      x      y     pred.    resid.
0   1.0  0.518  0.515869  0.002131
```

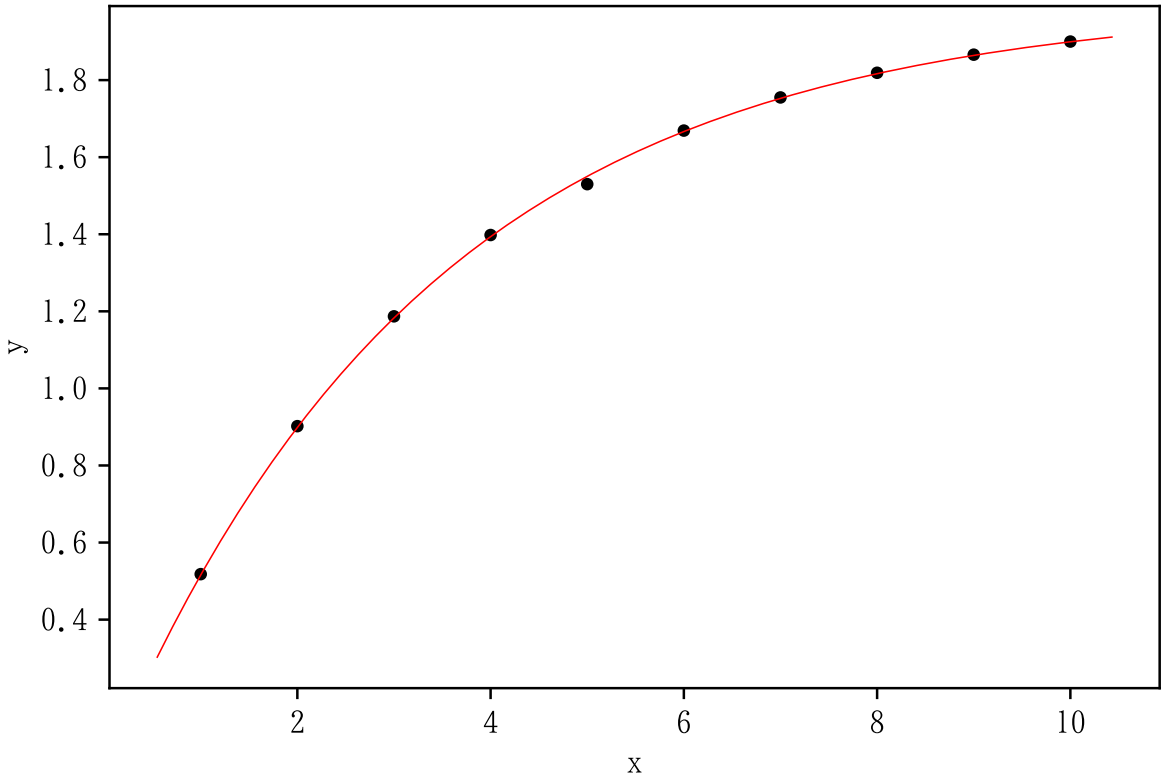
```
1   2.0  0.902  0.898739  0.003261
2   3.0  1.187  1.182899  0.004101
3   4.0  1.398  1.393798  0.004202
4   5.0  1.530  1.550324 -0.020324
5   6.0  1.669  1.666495  0.002505
6   7.0  1.755  1.752716  0.002284
7   8.0  1.819  1.816707  0.002293
8   9.0  1.866  1.864200  0.001800
9  10.0  1.900  1.899449  0.000551
```




```
b = nonlinear_fitting(x, y, a["p"], model="Exponential2")
graph(x, y, b)
```

```
Exponential2      by Marquardt method

                          estimates
a                          2.000916
b                          0.298159
residual sum of squares    0.000483

      x      y     pred.    resid.
0   1.0  0.518  0.515870  0.002130
1   2.0  0.902  0.898739  0.003261
```

```
2   3.0  1.187  1.182899  0.004101
3   4.0  1.398  1.393798  0.004202
4   5.0  1.530  1.550323 -0.020323
5   6.0  1.669  1.666493  0.002507
6   7.0  1.755  1.752713  0.002287
7   8.0  1.819  1.816704  0.002296
8   9.0  1.866  1.864197  0.001803
9  10.0  1.900  1.899446  0.000554
```

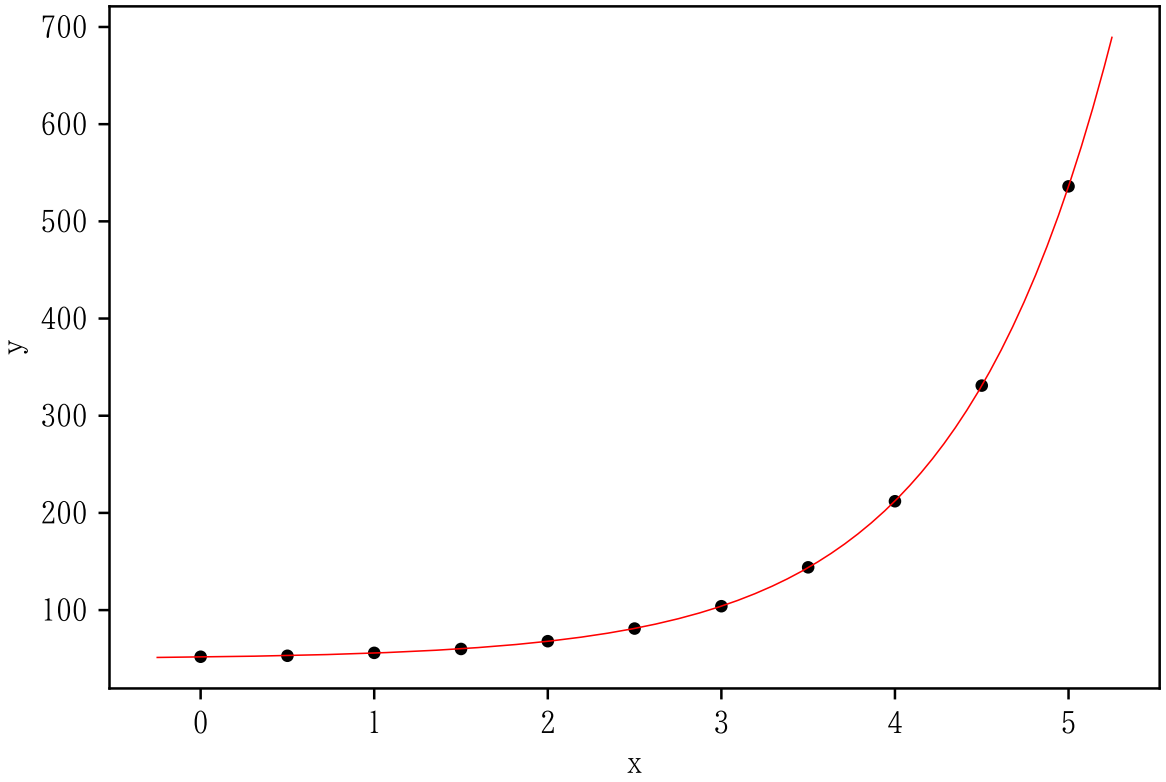


3.3 model = "Asymptotic"

```
import numpy as np

inival = [2, 2, 2]
x = np.arange(0, 11) / 2
y = np.array([52, 53, 56, 60, 68, 81, 104, 144, 212, 331, 536])
```

```
np.random.seed(1234) # 通常は指定しない
inival = [2, 2, 2]
a = nonlinear_fitting(x, y, inival, model="Asymptotic", method="
    simplex")
graph(x, y, a)
```

```
Asymptotic      by Simplex method

                         estimates
a                         2.028238
b                         2.991959
c                        49.735072
residual sum of squares   0.421581

      x      y       pred.    resid.
0   0.0   52.0   51.763310  0.236690
1   0.5   53.0   53.243372 -0.243372
2   1.0   56.0   55.803477  0.196523
3   1.5   60.0   60.231762 -0.231762
4   2.0   68.0   67.891491  0.108509
5   2.5   81.0   81.140738 -0.140738
6   3.0  104.0  104.058334 -0.058334
7   3.5  144.0  143.699541  0.300459
8   4.0  212.0  212.268049 -0.268049
9   4.5  331.0  330.872921  0.127079
10  5.0  536.0  536.027094 -0.027094
```

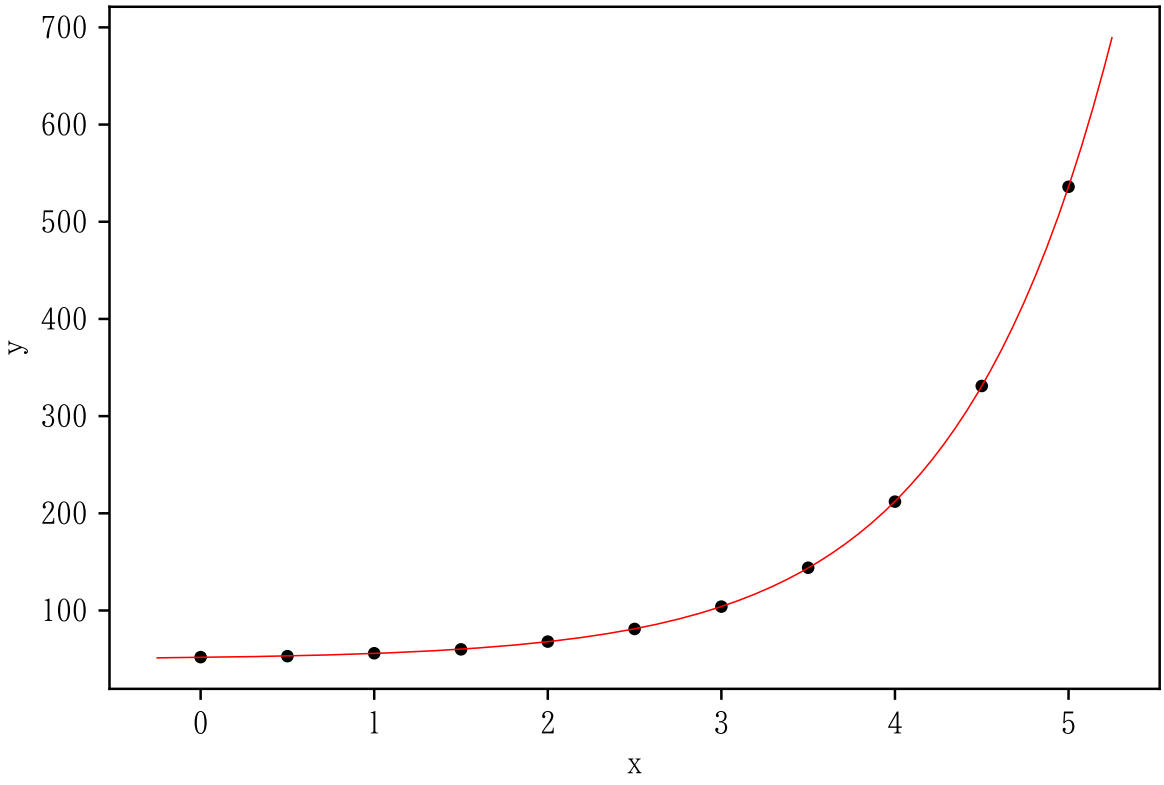



```
b = nonlinear_fitting(x, y, a["p"], model="Asymptotic")
```

```
graph(x, y, b)
```

```
Asymptotic      by Marquardt method

                          estimates
a                          2.028235
b                          2.991960
c                         49.735076
residual sum of squares    0.421581
```

	x	y	pred.	resid.
0	0.0	52.0	51.763311	0.236689
1	0.5	53.0	53.243372	-0.243372
2	1.0	56.0	55.803475	0.196525
3	1.5	60.0	60.231757	-0.231757
4	2.0	68.0	67.891481	0.108519
5	2.5	81.0	81.140723	-0.140723
6	3.0	104.0	104.058312	-0.058312
7	3.5	144.0	143.699511	0.300489
8	4.0	212.0	212.268015	-0.268015
9	4.5	331.0	330.872892	0.127108
10	5.0	536.0	536.027099	-0.027099

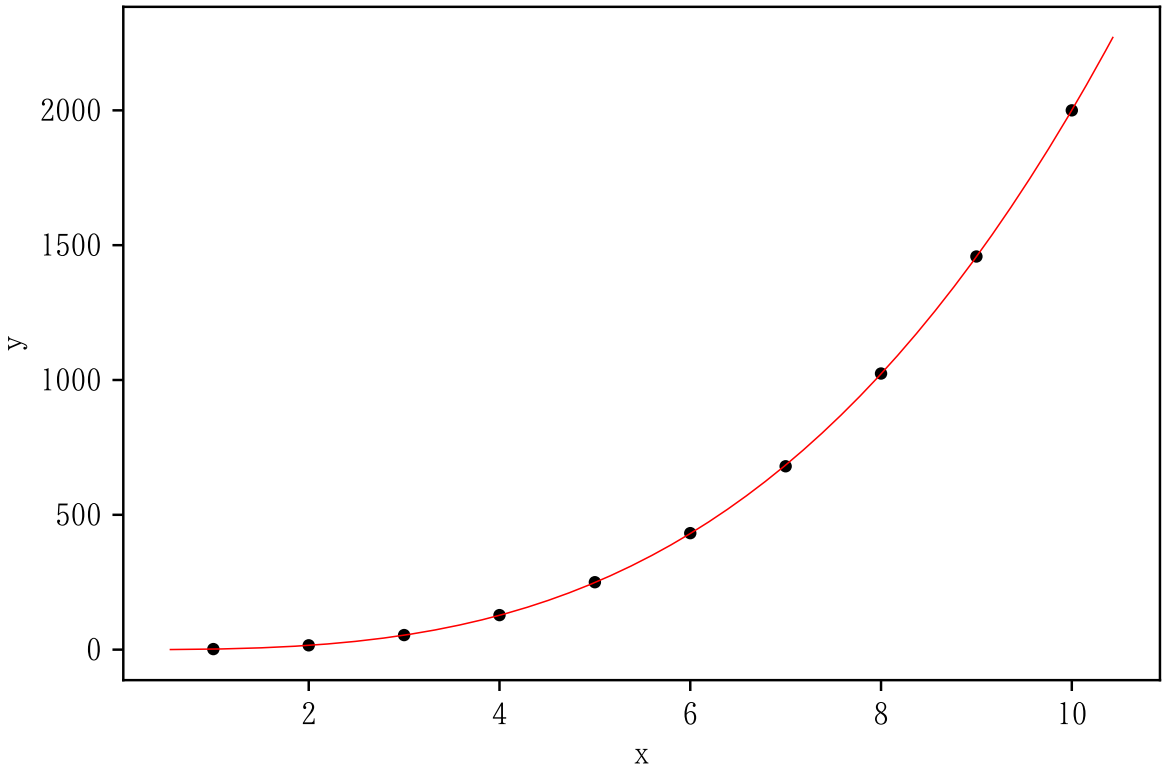


3.4 model = "Power1"

```
import numpy as np

x = np.arange(1, 11)
y = np.array([2, 16, 54, 128, 250, 432, 680, 1024, 1458, 2000])
```

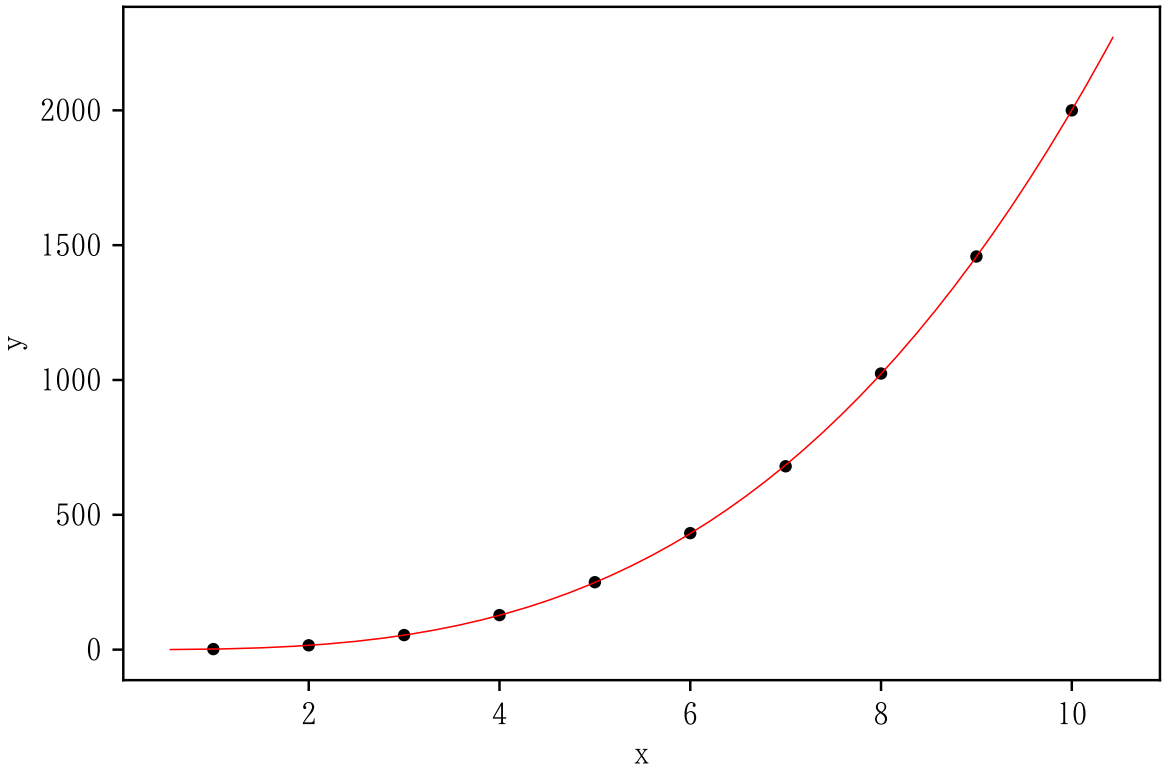
```
np.random.seed(1234) # 通常は指定しない
inival = [1, 1]
a = nonlinear_fitting(x, y, inival, model="Power1", method="simplex")
graph(x, y, a)
```

```
Power1      by Simplex method

                          estimates
a                          1.969082
b                          3.006834
residual sum of squares   26.620294

      x       y        pred.    resid.
0   1.0     2.0     1.969082  0.030918
1   2.0    16.0    15.827453  0.172547
2   3.0    54.0    53.565872  0.434128
3   4.0   128.0   127.220818  0.779182
4   5.0   250.0   248.857352  1.142648
5   6.0   432.0   430.561621  1.438379
6   7.0   680.0   684.436527 -4.436527
7   8.0  1024.0  1022.598896  1.401104
8   9.0  1458.0  1457.177471  0.822529
9  10.0  2000.0  2000.311414 -0.311414
```




```
b = nonlinear_fitting(x, y, a["p"], model="Power1")
graph(x, y, b)
```

```
Power1      by Marquardt method

                          estimates
a                          1.969074
b                          3.006836
residual sum of squares   26.620293

      x       y        pred.    resid.
0   1.0     2.0     1.969074  0.030926
1   2.0    16.0    15.827408  0.172592
2   3.0    54.0    53.565760  0.434240
3   4.0   128.0   127.220625  0.779375
4   5.0   250.0   248.857084  1.142916
5   6.0   432.0   430.561311  1.438689
6   7.0   680.0   684.436243 -4.436243
7   8.0  1024.0  1022.598741  1.401259
8   9.0  1458.0  1457.177587  0.822413
9  10.0  2000.0  2000.311988 -0.311988
```



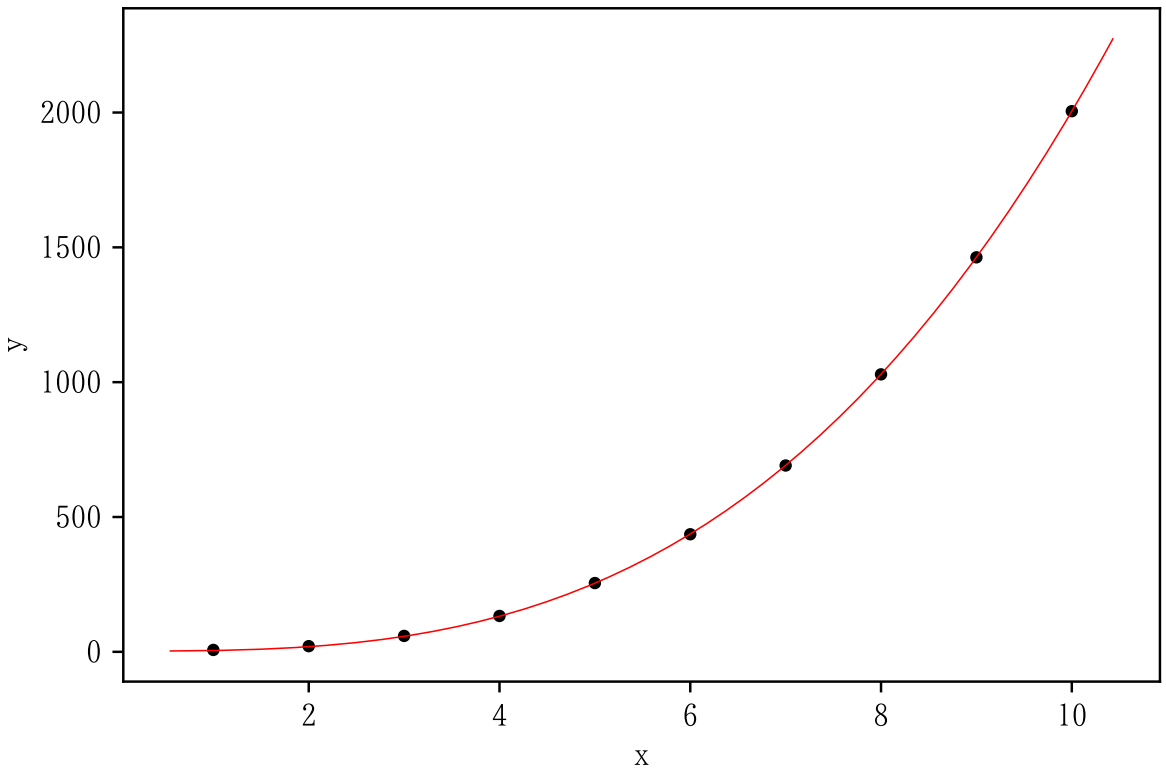
3.5 model = "Power2"

```
import numpy as np

x = np.arange(1, 11)
y = np.array([7, 21, 59, 133, 255, 436, 691, 1029, 1463, 2005])
```

```
np.random.seed(1234) # 通常は指定しない
inival = [1, 1, 1]
a = nonlinear_fitting(x, y, inival, model="Power2", method="simplex")
graph(x, y, a)
```

```
Power2      by Simplex method

                        estimates
a                        2.041872
b                        2.991345
c                        2.767676
residual sum of squares  14.965732

      x       y        pred.    resid.
0   1.0     7.0     4.809548  2.190452
```

1	2.0	21.0	19.004947	1.995053
2	3.0	59.0	57.376488	1.623512
3	4.0	133.0	131.888882	1.111118
4	5.0	255.0	254.470936	0.529064
5	6.0	436.0	437.025100	-1.025100
6	7.0	691.0	691.432951	-0.432951
7	8.0	1029.0	1029.558775	-0.558775
8	9.0	1463.0	1463.252096	-0.252096
9	10.0	2005.0	2004.349557	0.650443



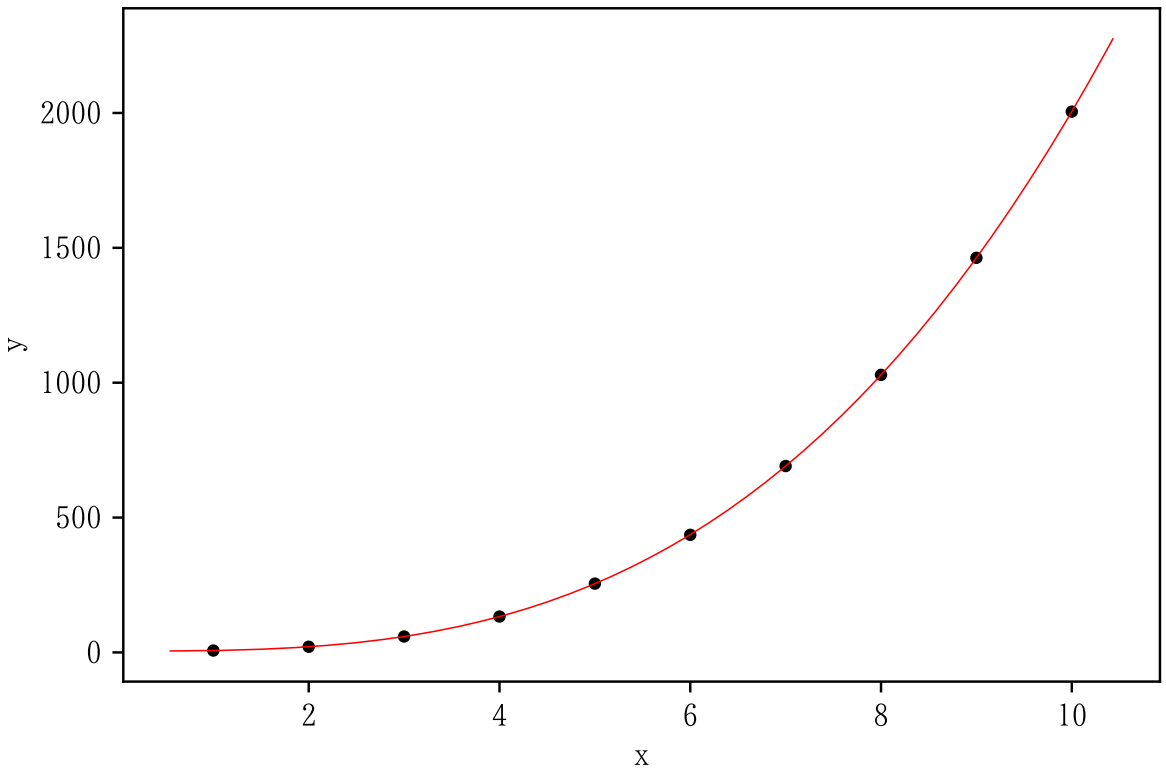

```
b = nonlinear_fitting(x, y, a["p"], model="Power2")
graph(x, y, b)
```

Power2　　by Marquardt method

	estimates
a	1.994464
b	3.001226
c	5.020082
residual sum of squares	0.771781

	x	y	pred.	resid.
0	1.0	7.0	7.014546	-0.014546

```
1   2.0    21.0    20.989361  0.010639
2   3.0    59.0    58.943198  0.056802
3   4.0   133.0   132.882932  0.117068
4   5.0   255.0   254.820547  0.179453
5   6.0   436.0   436.771792 -0.771792
6   7.0   691.0   690.755402  0.244598
7   8.0  1029.0  1028.792595  0.207405
8   9.0  1463.0  1462.906705  0.093295
9  10.0  2005.0  2005.122922 -0.122922
```

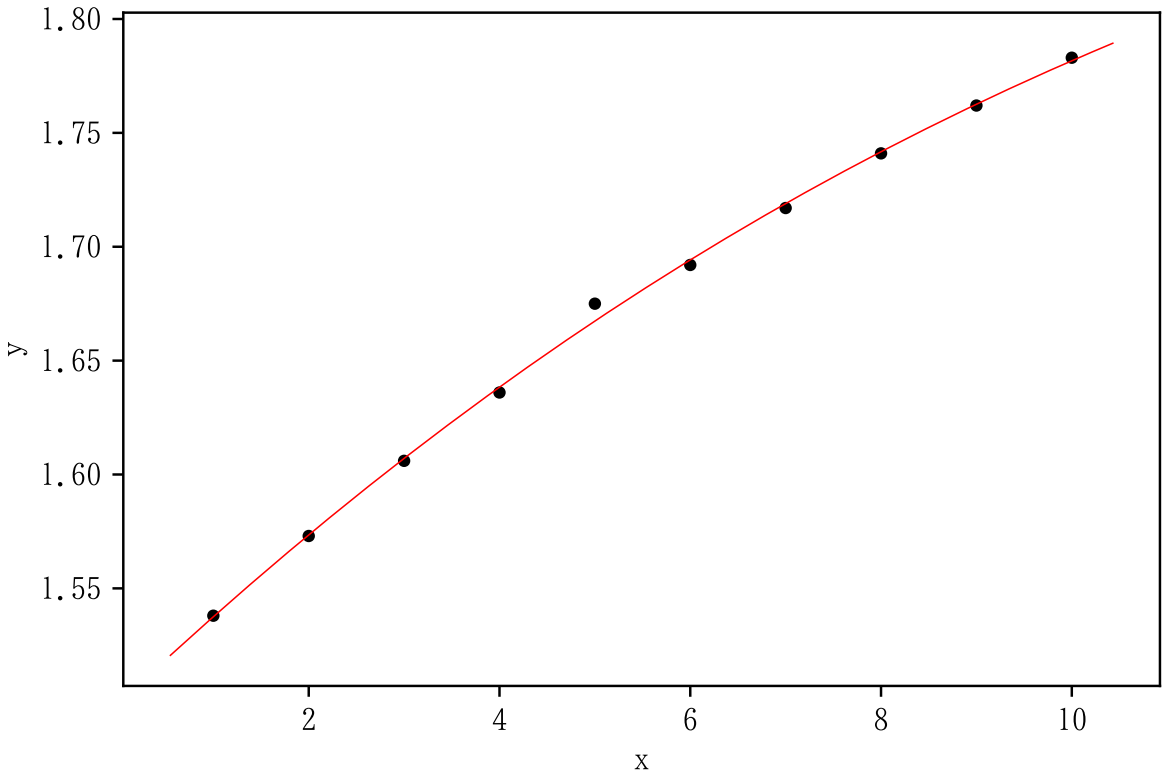


3.6 model = "Logistic1"

```
import numpy as np

x = np.arange(1, 11)
y =   np.array([1.538, 1.573, 1.606, 1.636, 1.675, 1.692, 1.717,
    1.741, 1.762, 1.783])
```

```
np.random.seed(1234) # 通常は指定しない
inival = [1, 1, 1]
a = nonlinear_fitting(x, y, inival, model="Logistic1", method="simplex
    ")
graph(x, y, a)
```

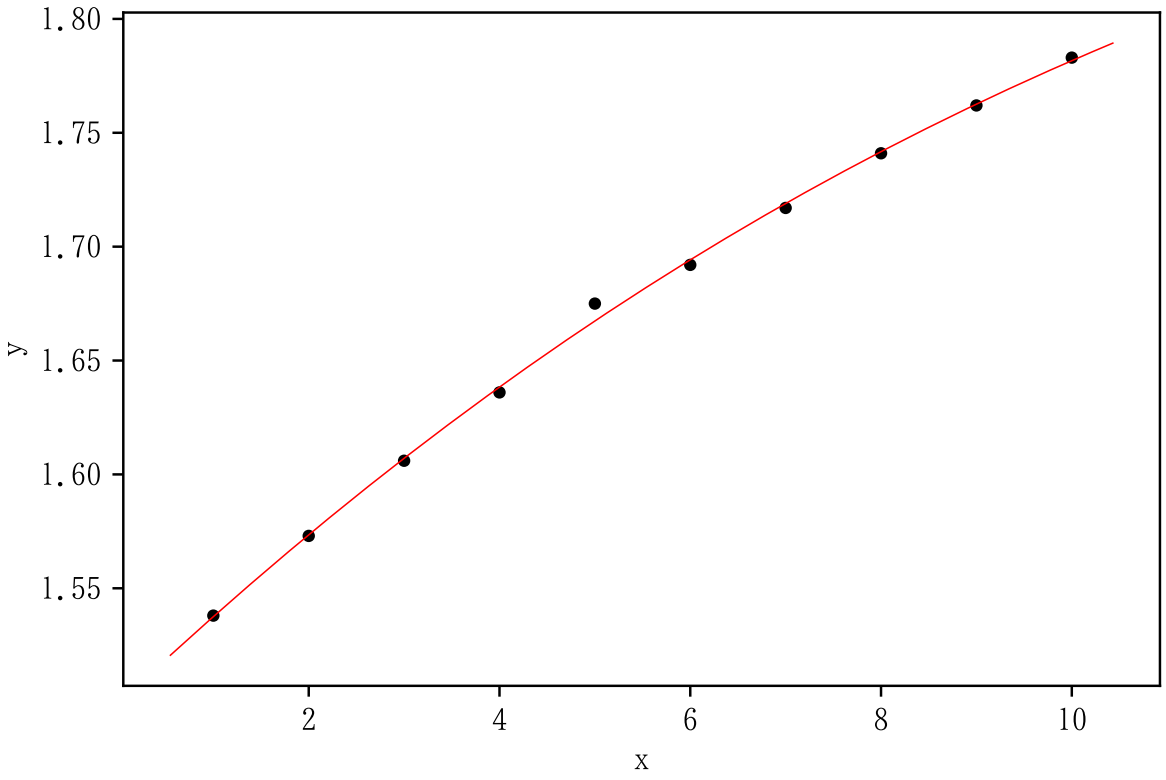
```
Logistic1      by Simplex method

                          estimates
a                          1.961317
b                          0.308199
c                          0.111728
residual sum of squares    0.000077

      x      y     pred.    resid.
0   1.0  1.538  1.537542  0.000458
1   2.0  1.573  1.573482 -0.000482
2   3.0  1.606  1.607076 -0.001076
3   4.0  1.636  1.638357 -0.002357
4   5.0  1.675  1.667381  0.007619
5   6.0  1.692  1.694223 -0.002223
6   7.0  1.717  1.718969 -0.001969
7   8.0  1.741  1.741720 -0.000720
8   9.0  1.762  1.762582 -0.000582
9  10.0  1.783  1.781666  0.001334
```




```
b = nonlinear_fitting(x, y, a["p"], model="Logistic1")
```

```
graph(x, y, b)
```

```
Logistic1      by Marquardt method

                          estimates
a                          1.961388
b                          0.308241
c                          0.111705
residual sum of squares    0.000077

      x      y     pred.    resid.
0   1.0  1.538  1.537544  0.000456
1   2.0  1.573  1.573482 -0.000482
2   3.0  1.606  1.607074 -0.001074
3   4.0  1.636  1.638355 -0.002355
4   5.0  1.675  1.667379  0.007621
5   6.0  1.692  1.694221 -0.002221
6   7.0  1.717  1.718968 -0.001968
7   8.0  1.741  1.741721 -0.000721
8   9.0  1.762  1.762585 -0.000585
9  10.0  1.783  1.781671  0.001329
```

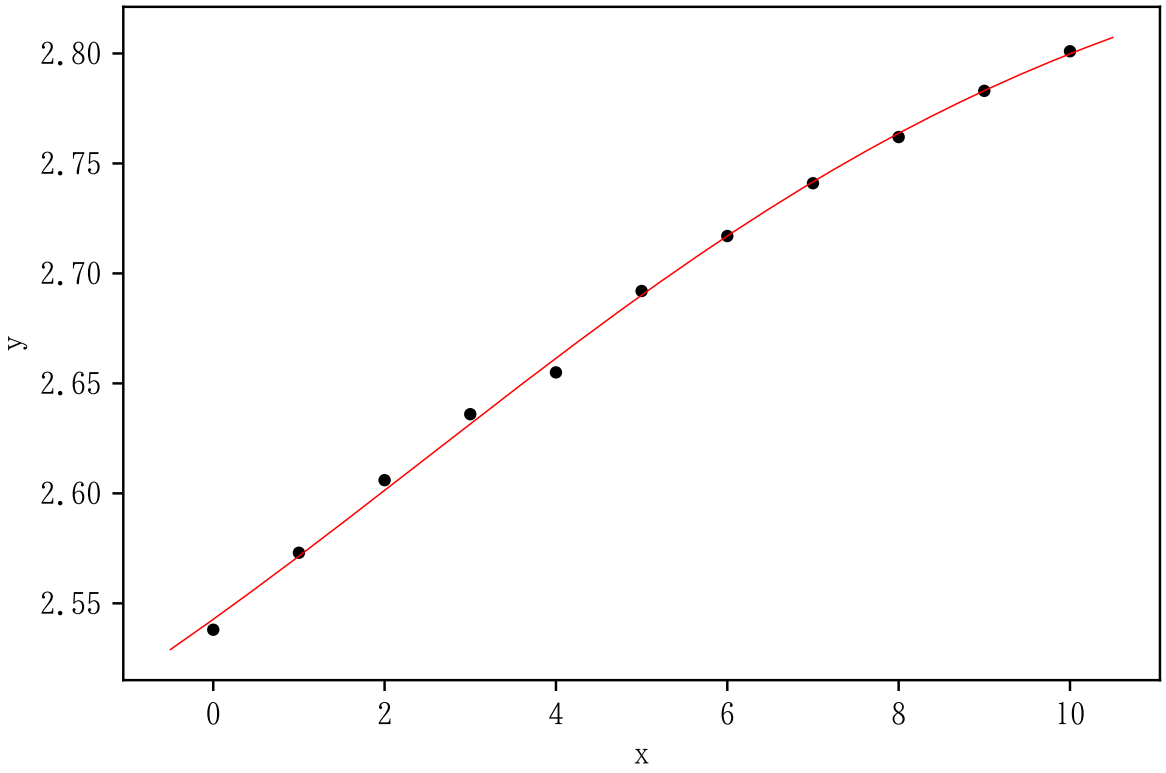


3.7 model = "Logistic2"

```
import numpy as np

x = np.arange(0, 11)
y =   np.array([2.538, 2.573, 2.606, 2.636, 2.655, 2.692, 2.717,
   2.741, 2.762, 2.783, 2.801])
```

```
np.random.seed(1234) # 通常は指定しない
inival = [1, 1, 1, 1]
a = nonlinear_fitting(x, y, inival, model="Logistic2", method="simplex
   ")
graph(x, y, a)
```

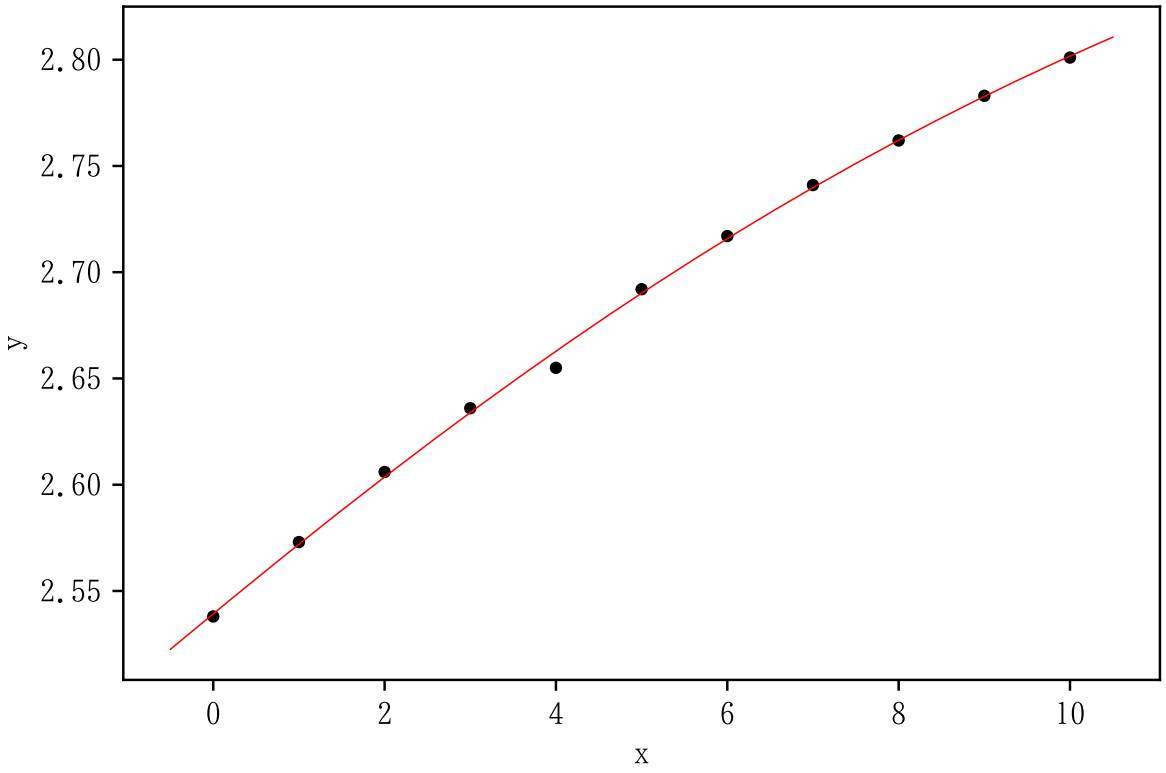
```
Logistic2      by Simplex method

                          estimates
a                          0.525845
b                          1.772696
c                          0.230427
d                          2.353074
residual sum of squares    0.000116

       x      y     pred.    resid.
0    0.0  2.538  2.542725 -0.004725
1    1.0  2.573  2.571460  0.001540
2    2.0  2.606  2.601334  0.004666
3    3.0  2.636  2.631593  0.004407
4    4.0  2.655  2.661442 -0.006442
5    5.0  2.692  2.690131  0.001869
6    6.0  2.717  2.717023 -0.000023
7    7.0  2.741  2.741644 -0.000644
8    8.0  2.762  2.763706 -0.001706
9    9.0  2.783  2.783096 -0.000096
10  10.0  2.801  2.799852  0.001148
```




```
b = nonlinear_fitting(x, y, a["p"], model="Logistic2")
graph(x, y, b)
```

```
Logistic2      by Marquardt method

                          estimates
a                          1.103042
b                          0.644541
c                          0.126422
d                          1.868542
residual sum of squares    0.000081

       x      y     pred.    resid.
0    0.0  2.538  2.539271 -0.001271
1    1.0  2.573  2.572014  0.000986
2    2.0  2.606  2.603637  0.002363
3    3.0  2.636  2.633958  0.002042
4    4.0  2.655  2.662831 -0.007831
5    5.0  2.692  2.690142  0.001858
6    6.0  2.717  2.715815  0.001185
7    7.0  2.741  2.739807  0.001193
8    8.0  2.762  2.762105 -0.000105
```

```
9    9.0  2.783  2.782723  0.000277
10  10.0  2.801  2.801697 -0.000697
```



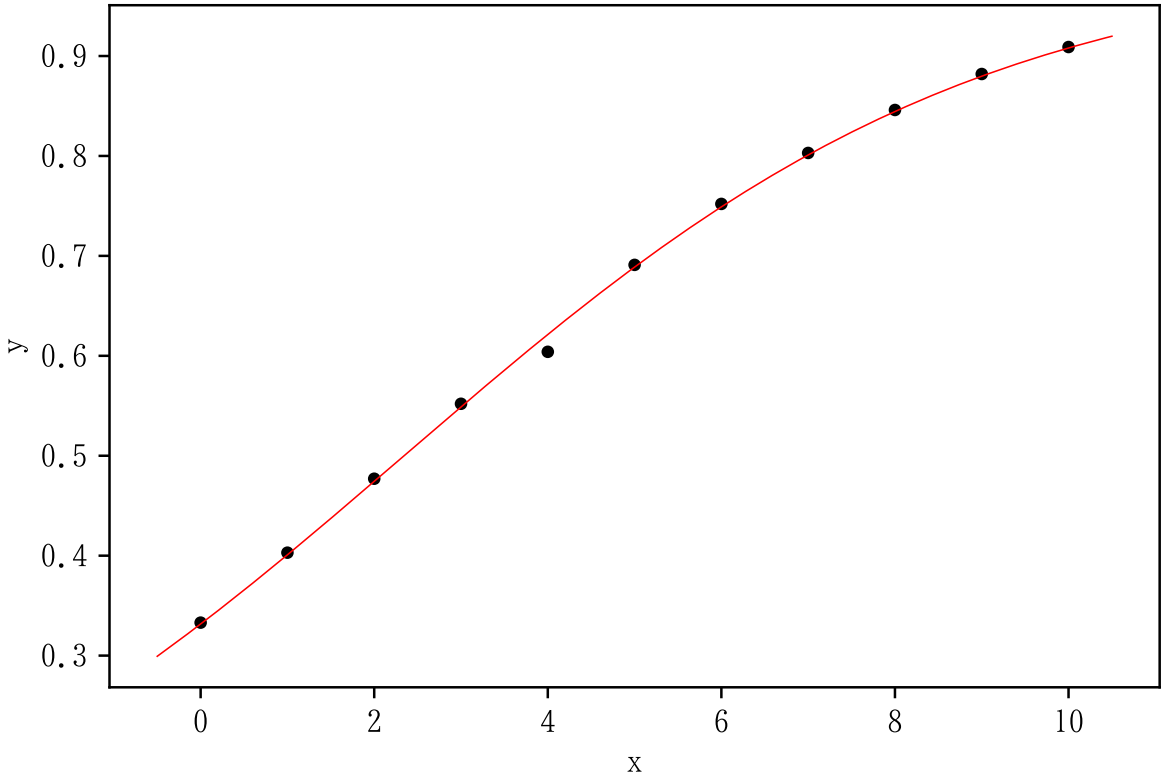
3.8 model = "Logistic3"

```
import numpy as np

x = np.arange(0, 11)
y =   np.array([0.333, 0.403, 0.477, 0.552, 0.604, 0.691, 0.752,
   0.803, 0.846, 0.882, 0.909])
```

```
np.random.seed(1234) # 通常は指定しない
inival = [1, 1, 1]
a = nonlinear_fitting(x, y, inival, model="Logistic3", method="simplex
   ")
graph(x, y, a)
```

```
Logistic3      by Simplex method

                        estimates
a                        2.015847
b                        0.299125
c                        0.755249
```

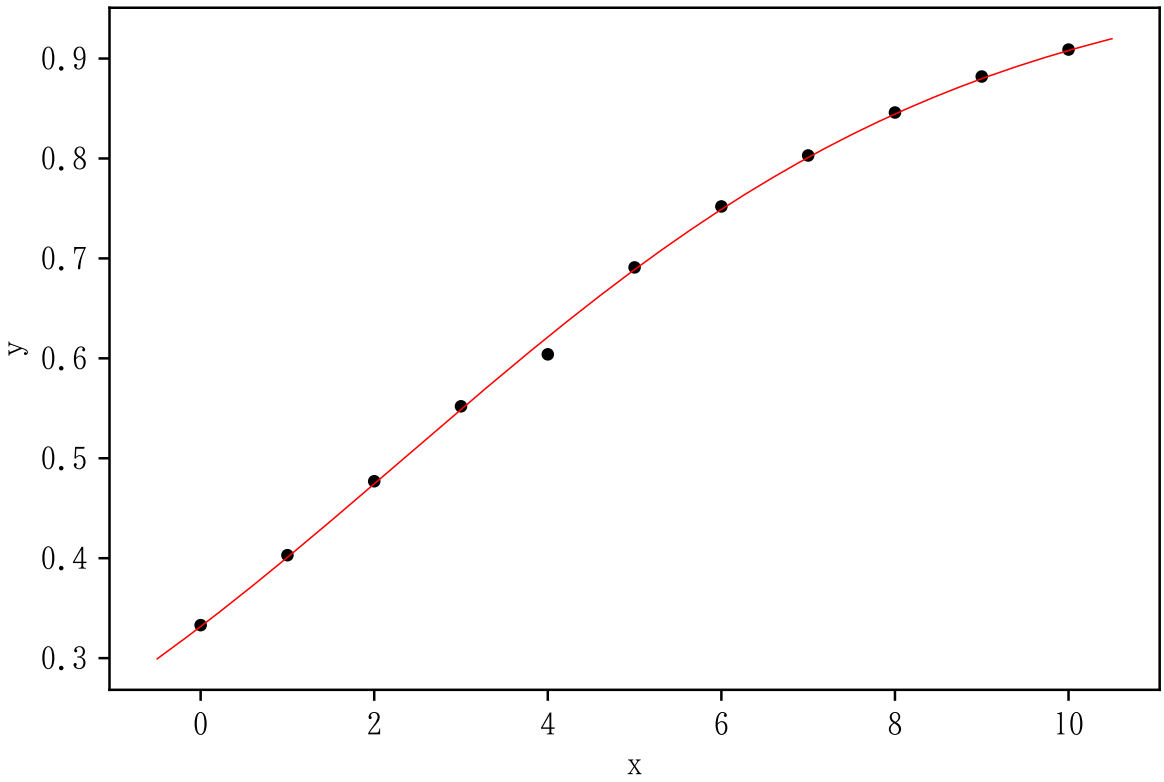
```
residual sum of squares   0.000351

       x      y     pred.    resid.
0    0.0  0.333  0.331582  0.001418
1    1.0  0.403  0.400852  0.002148
2    2.0  0.477  0.474325  0.002675
3    3.0  0.552  0.548927  0.003073
4    4.0  0.604  0.621392 -0.017392
5    5.0  0.691  0.688815  0.002185
6    6.0  0.752  0.749079  0.002921
7    7.0  0.803  0.801044  0.001956
8    8.0  0.846  0.844482  0.001518
9    9.0  0.882  0.879858  0.002142
10  10.0  0.909  0.908063  0.000937
```




```
b = nonlinear_fitting(x, y, a["p"], model="Logistic3")
graph(x, y, b)
```

```
Logistic3      by Marquardt method

                          estimates
a                          2.015847
b                          0.299125
```

```
c                          0.755249
residual sum of squares    0.000351

       x      y     pred.    resid.
0    0.0  0.333  0.331582  0.001418
1    1.0  0.403  0.400852  0.002148
2    2.0  0.477  0.474325  0.002675
3    3.0  0.552  0.548927  0.003073
4    4.0  0.604  0.621392 -0.017392
5    5.0  0.691  0.688815  0.002185
6    6.0  0.752  0.749079  0.002921
7    7.0  0.803  0.801044  0.001956
8    8.0  0.846  0.844482  0.001518
9    9.0  0.882  0.879858  0.002142
10  10.0  0.909  0.908063  0.000937
```



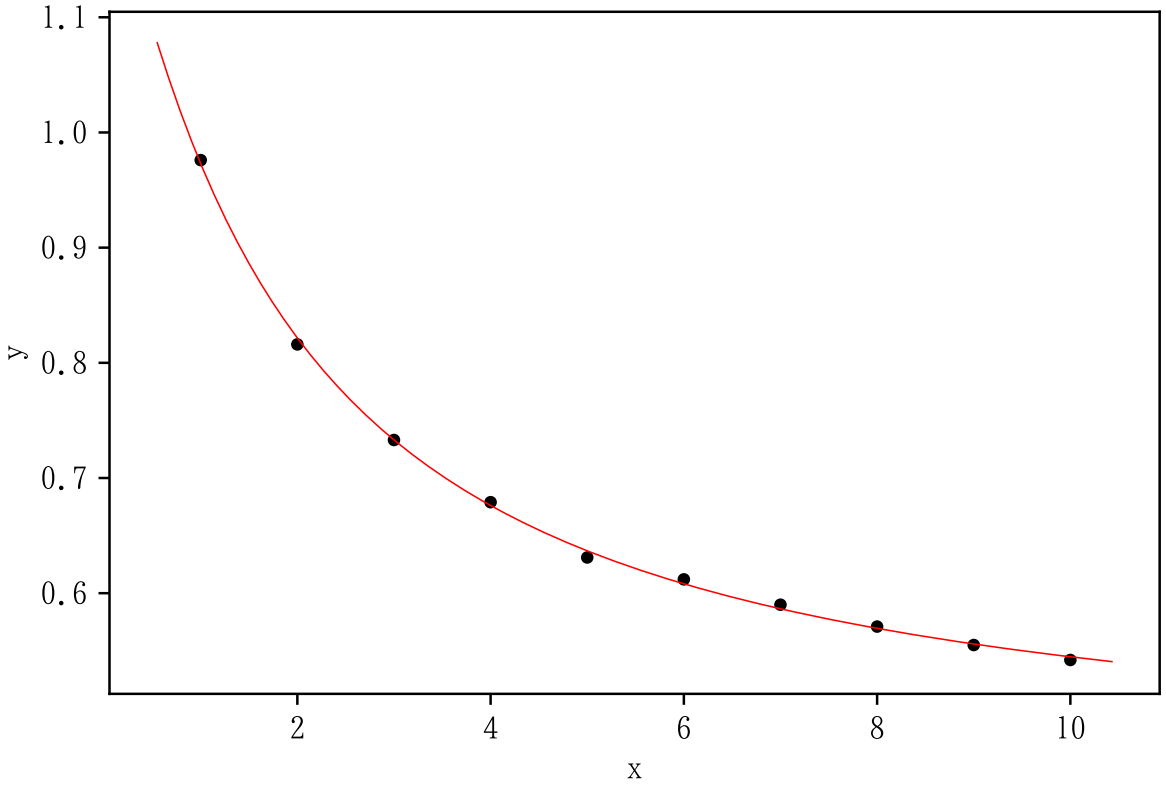
3.9 model = "Logistic4"

```
import numpy as np

x = np.arange(1, 11)
y =   np.array([0.976, 0.816, 0.733, 0.679, 0.631, 0.612, 0.59, 0.571,
```

```
    0.555, 0.542])
```

```
np.random.seed(1234) # 通常は指定しない
inival = [1, 1, 1, 1]
a = nonlinear_fitting(x, y, inival, model="Logistic4", method="simplex
    ")
graph(x, y, a)
```

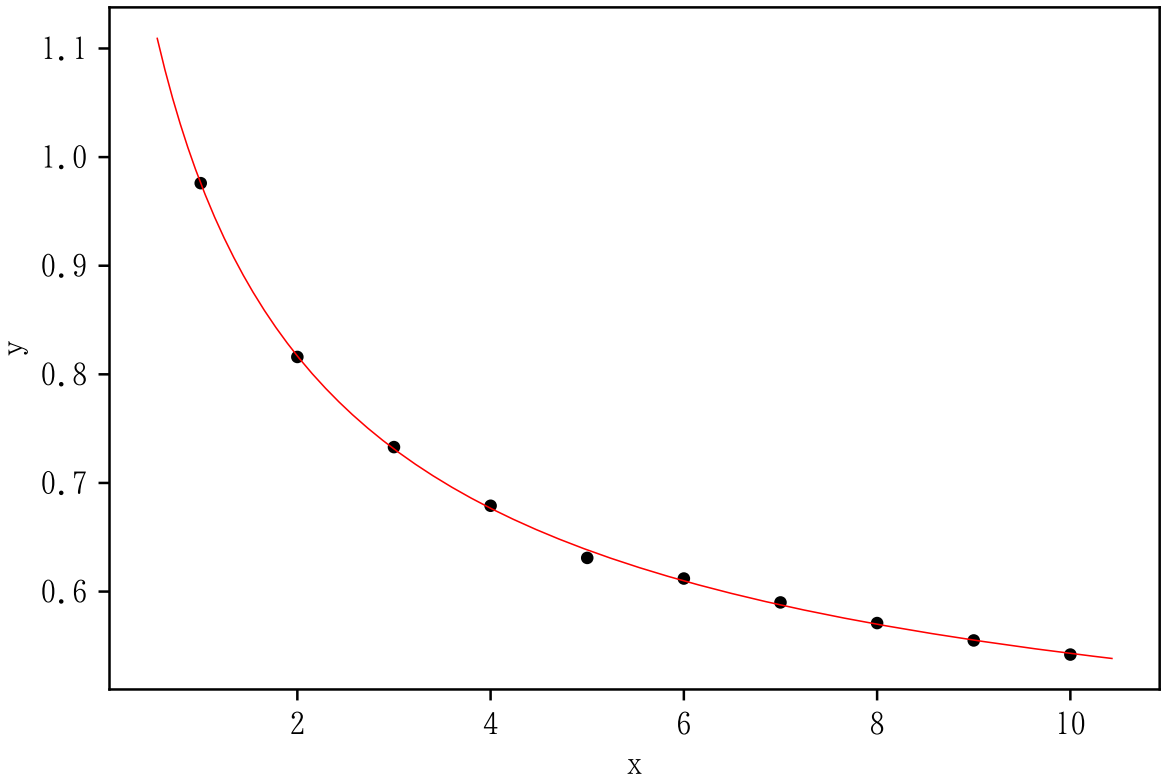
```
Logistic4      by Simplex method

                          estimates
a                          1.243296
b                          0.441712
c                          1.822641
d                          1.124166
residual sum of squares    0.000122

      x      y     pred.    resid.
0   1.0  0.976  0.972827  0.003173
1   2.0  0.816  0.821603 -0.005603
2   3.0  0.733  0.733088 -0.000088
3   4.0  0.679  0.676120  0.002880
4   5.0  0.631  0.636769 -0.005769
5   6.0  0.612  0.608126  0.003874
6   7.0  0.590  0.586428  0.003572
7   8.0  0.571  0.569472  0.001528
8   9.0  0.555  0.555884 -0.000884
9  10.0  0.542  0.544771 -0.002771
```




```
b = nonlinear_fitting(x, y, a["p"], model="Logistic4")
graph(x, y, b)
```

```
Logistic4      by Marquardt method

                          estimates
a                          1.490517
b                          0.387936
c                          1.171216
d                          0.843693
residual sum of squares    0.000076

      x      y     pred.    resid.
0   1.0  0.976  0.975926  0.000074
1   2.0  0.816  0.816854 -0.000854
2   3.0  0.733  0.731287  0.001713
3   4.0  0.679  0.676670  0.002330
4   5.0  0.631  0.638376 -0.007376
5   6.0  0.612  0.609856  0.002144
6   7.0  0.590  0.587695  0.002305
7   8.0  0.571  0.569925  0.001075
8   9.0  0.555  0.555323 -0.000323
```

```
9  10.0  0.542  0.543089 -0.001089
```



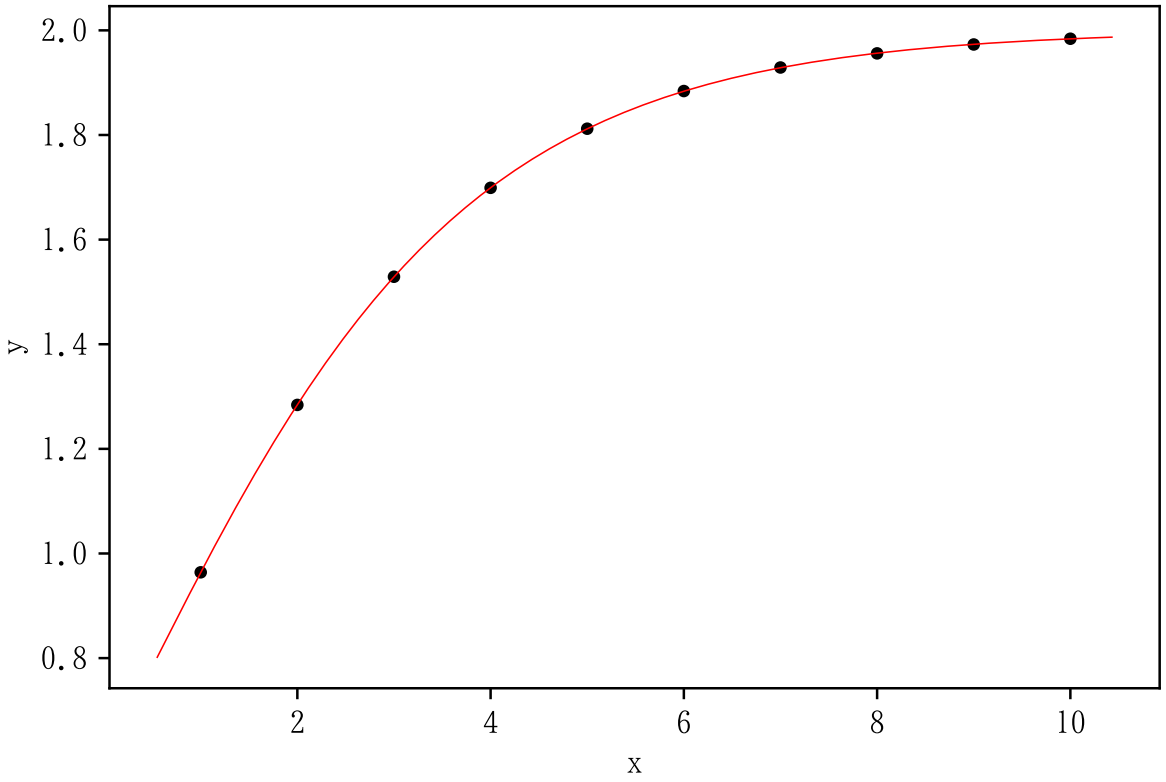
3.10 model = "Gomperts"

```
import numpy as np

x = np.arange(1, 11)
y =   np.array([0.964, 1.284, 1.529, 1.699, 1.812, 1.884, 1.929,
    1.956, 1.973, 1.984])
```

```
np.random.seed(1234) # 通常は指定しない
inival = [1.1, 1.1, 1.1]
a = nonlinear_fitting(x, y, inival, model="Gomperts", method="simplex
    ")
graph(x, y, a)
```

```
Gomperts      by Simplex method

                          estimates
a                          2.000028
b                          0.300125
c                          0.499889
residual sum of squares    0.000001
```

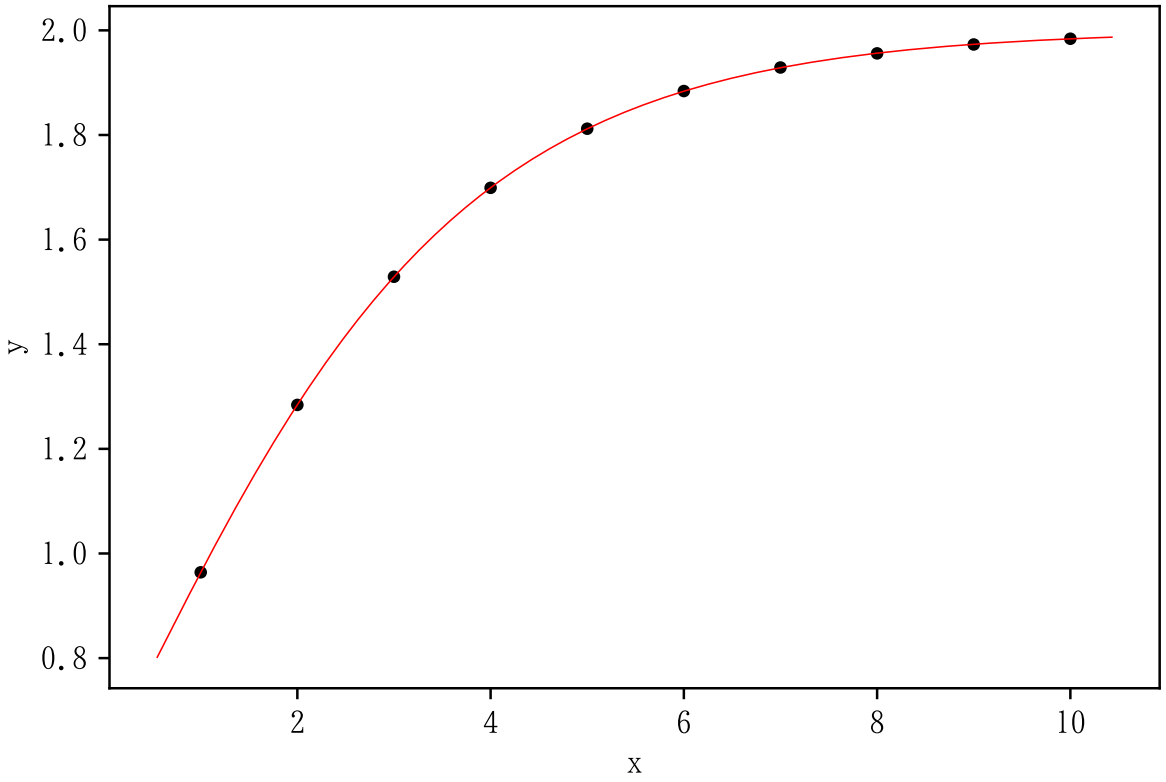
	x	y	pred.	resid.
0	1.0	0.964	0.963760	0.000240
1	2.0	1.284	1.284410	-0.000410
2	3.0	1.529	1.528859	0.000141
3	4.0	1.699	1.699281	-0.000281
4	5.0	1.812	1.811784	0.000216
5	6.0	1.884	1.883628	0.000372
6	7.0	1.929	1.928589	0.000411
7	8.0	1.956	1.956383	-0.000383
8	9.0	1.973	1.973438	-0.000438
9	10.0	1.984	1.983856	0.000144




```
b = nonlinear_fitting(x, y, a["p"], model="Gomperts")
graph(x, y, b)
```

```
Gomperts      by Marquardt method

                          estimates
a                          2.000028
b                          0.300125
c                          0.499889
residual sum of squares    0.000001
```

```
      x      y     pred.    resid.
0   1.0  0.964  0.963760  0.000240
1   2.0  1.284  1.284410 -0.000410
2   3.0  1.529  1.528859  0.000141
3   4.0  1.699  1.699281 -0.000281
4   5.0  1.812  1.811784  0.000216
5   6.0  1.884  1.883628  0.000372
6   7.0  1.929  1.928589  0.000411
7   8.0  1.956  1.956383 -0.000383
8   9.0  1.973  1.973438 -0.000438
9  10.0  1.984  1.983856  0.000144
```



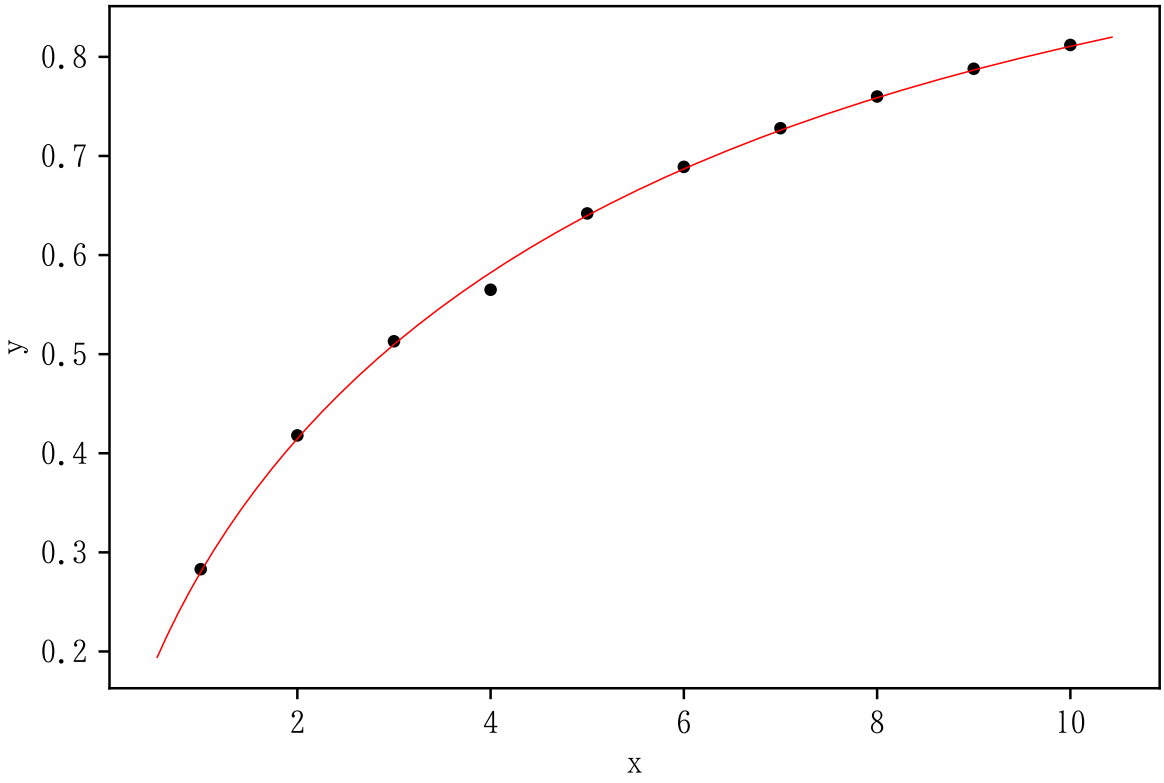
3.11 model = "Weibull"

```
import numpy as np

x = np.arange(1, 11)
y =   np.array([0.283, 0.418, 0.513, 0.565, 0.642, 0.689, 0.728, 0.76,
    0.788, 0.812])
```

```
np.random.seed(1234) # 通常は指定しない
```

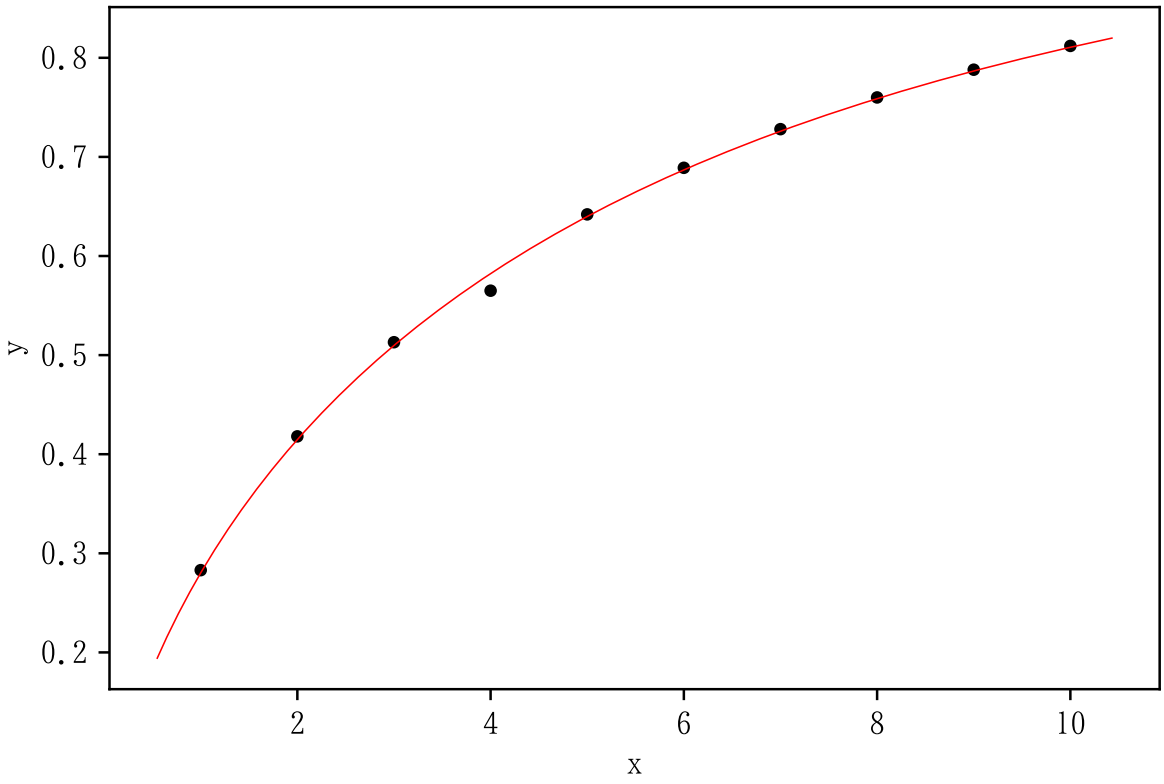
```
inival = [1, 1]
a = nonlinear_fitting(x, y, inival, model="Weibull", method="simplex")
graph(x, y, a)
```

```
Weibull      by Simplex method

                          estimates
a                          3.040486
b                          0.704106
residual sum of squares    0.000344

      x      y     pred.    resid.
0   1.0  0.283  0.280281  0.002719
1   2.0  0.418  0.414807  0.003193
2   3.0  0.513  0.509757  0.003243
3   4.0  0.565  0.582267 -0.017267
4   5.0  0.642  0.639918  0.002082
5   6.0  0.689  0.686933  0.002067
6   7.0  0.728  0.725960  0.002040
7   8.0  0.760  0.758792  0.001208
8   9.0  0.788  0.786701  0.001299
9  10.0  0.812  0.810627  0.001373
```



```
b = nonlinear_fitting(x, y, a["p"], model="Weibull")
graph(x, y, b)
```

```
Weibull      by Marquardt method

                          estimates
a                          3.040486
b                          0.704106
residual sum of squares    0.000344

      x      y     pred.    resid.
0   1.0  0.283  0.280281  0.002719
1   2.0  0.418  0.414807  0.003193
2   3.0  0.513  0.509757  0.003243
3   4.0  0.565  0.582267 -0.017267
4   5.0  0.642  0.639918  0.002082
5   6.0  0.689  0.686933  0.002067
6   7.0  0.728  0.725960  0.002040
7   8.0  0.760  0.758792  0.001208
8   9.0  0.788  0.786701  0.001299
9  10.0  0.812  0.810627  0.001373
```



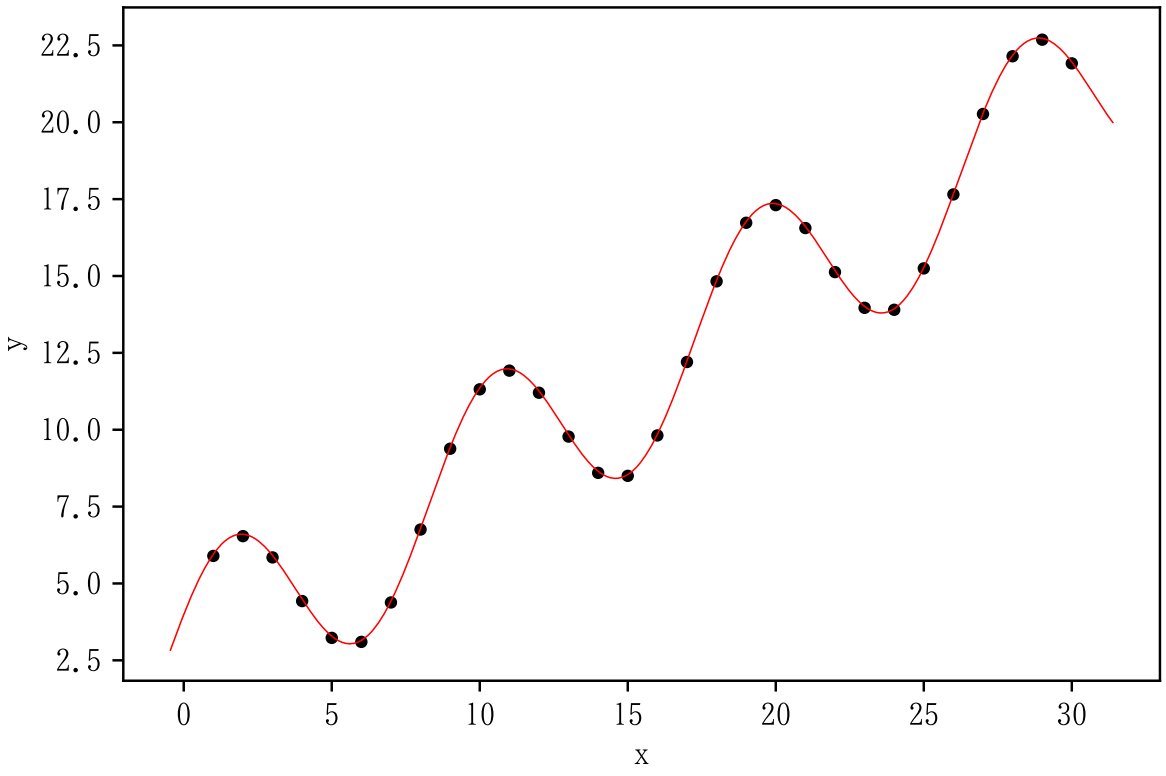
3.12 model = "Sin"

```
import numpy as np

x = np.arange(1, 31)
y =   np.array([5.896, 6.539, 5.847, 4.427, 3.23, 3.1, 4.382, 6.754,
   9.382,
11.314, 11.922, 11.203, 9.777, 8.597, 8.5, 9.814, 12.203, 14.826,
16.731, 17.305, 16.559, 15.127, 13.965, 13.901, 15.247, 17.653,
20.269, 22.147, 22.686, 21.915])
```

```
np.random.seed(1234) # 通常は指定しない
inival = [3.0911, 0.6935, 0.6307, 0.5922, 2.6575]
a = nonlinear_fitting(x, y, inival, model="Sin", method="simplex")
graph(x, y, a)
```

```
Sin      by Simplex method

                         estimates
a                         3.004147
b                         0.699925
c                         0.497174
d                         0.599190
e                         2.556804
residual sum of squares   0.064335

       x       y      pred.    resid.
0    1.0   5.896   5.952807 -0.056807
1    2.0   6.539   6.600886 -0.061886
2    3.0   5.847   5.910860 -0.063860
3    4.0   4.427   4.488945 -0.061945
4    5.0   3.230   3.285502 -0.055502
5    6.0   3.100   3.148164 -0.048164
6    7.0   4.382   4.423262 -0.041262
7    8.0   6.754   6.792969 -0.038969
8    9.0   9.382   9.424755 -0.042755
9   10.0  11.314  11.362853 -0.048853
10  11.0  11.922  11.977682 -0.055682
11  12.0  11.203  11.261889 -0.058889
12  13.0   9.777   9.833803 -0.056803
13  14.0   8.597   8.646690 -0.049690
14  15.0   8.500   8.540504 -0.040504
```

```
15  16.0   9.814   9.846925 -0.032925
16  17.0  12.203  12.233401 -0.030401
17  18.0  14.826  14.859514 -0.033514
18  19.0  16.731  16.772166 -0.041166
19  20.0  17.305  17.353741 -0.048741
20  21.0  16.559  16.612522 -0.053522
21  22.0  15.127  15.178795 -0.051795
22  23.0  13.965  14.008478 -0.043478
23  24.0  13.901  13.933626 -0.032626
24  25.0  15.247  15.271188 -0.024188
25  26.0  17.653  17.673965 -0.020965
26  27.0  20.269  20.293877 -0.024877
27  28.0  22.147  22.180741 -0.033741
28  29.0  22.686  22.729067 -0.043067
29  30.0  21.915  21.962772 -0.047772
```

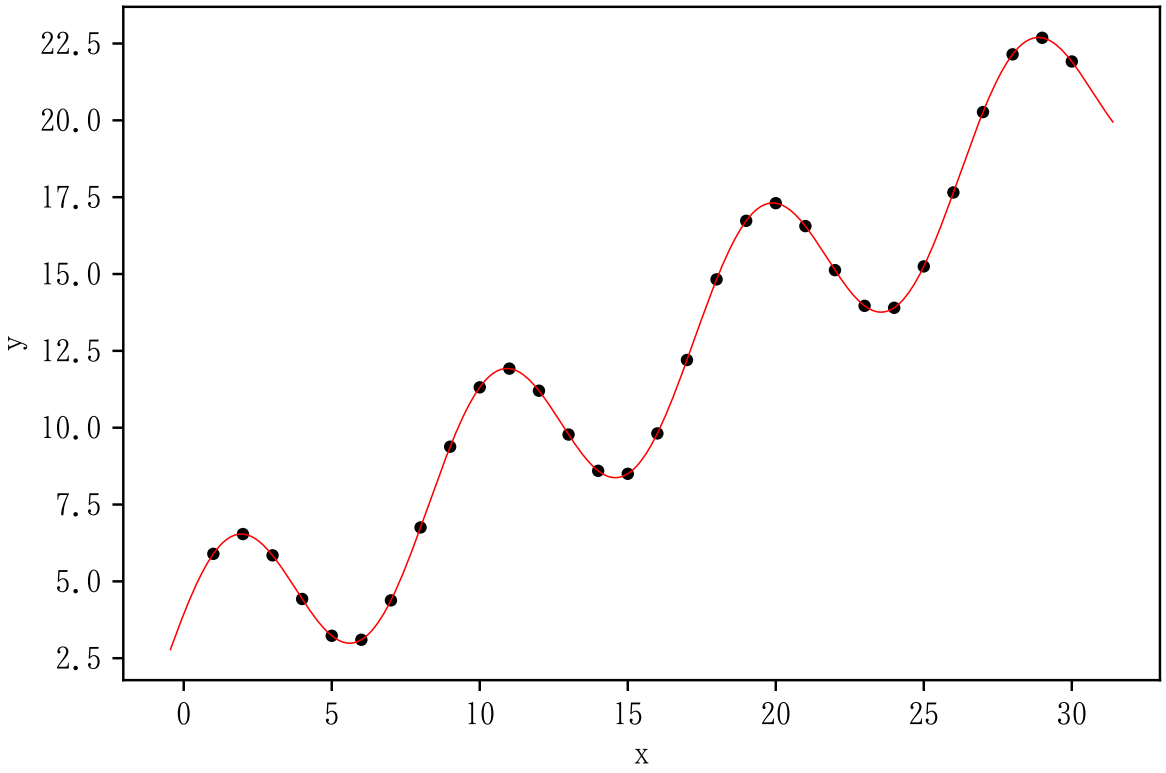



```
inival = [3.0911, 0.6935, 0.6307, 0.5922, 2.6575]
b = nonlinear_fitting(x, y, inival, model="Sin")
graph(x, y, b)
```

```
Sin      by Marquardt method


                        estimates
```

a	2.999913
b	0.700002
c	0.499962
d	0.599990
e	2.500157
residual sum of squares	0.000001

	x	y	pred.	resid.
0	1.0	5.896	5.896144	-0.000144
1	2.0	6.539	6.538987	0.000013
2	3.0	5.847	5.846669	0.000331
3	4.0	4.427	4.426984	0.000016
4	5.0	3.230	3.229821	0.000179
5	6.0	3.100	3.100415	-0.000415
6	7.0	4.382	4.381812	0.000188
7	8.0	6.754	6.753535	0.000465
8	9.0	9.382	9.382307	-0.000307
9	10.0	11.314	11.313954	0.000046
10	11.0	11.922	11.922175	-0.000175
11	12.0	11.203	11.203098	-0.000098
12	13.0	9.777	9.777105	-0.000105
13	14.0	8.597	8.597051	-0.000051
14	15.0	8.500	8.500123	-0.000123
15	16.0	9.814	9.814093	-0.000093
16	17.0	12.203	12.203165	-0.000165
17	18.0	14.826	14.825901	0.000099
18	19.0	16.731	16.730967	0.000033
19	20.0	17.305	17.304562	0.000438
20	21.0	16.559	16.559102	-0.000102
21	22.0	15.127	15.127374	-0.000374
22	23.0	13.965	13.964932	0.000068
23	24.0	13.901	13.900680	0.000320
24	25.0	15.247	15.247022	-0.000022
25	26.0	17.653	17.652936	0.000064
26	27.0	20.269	20.269063	-0.000063
27	28.0	22.147	22.147178	-0.000178
28	29.0	22.686	22.686156	-0.000156
29	30.0	21.915	21.914692	0.000308



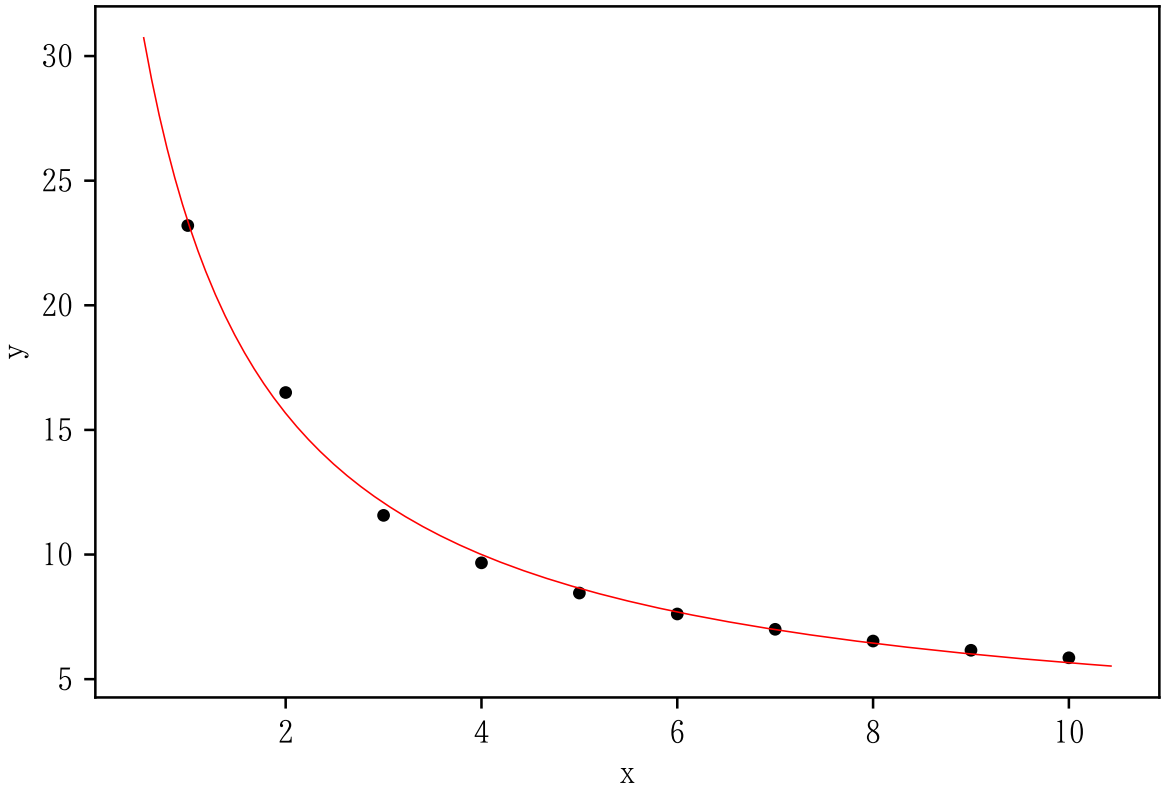
3.13 model = "Hyperbola1"

```
import numpy as np

x = np.arange(1, 11)
y =   np.array([23.2, 16.5, 11.571, 9.667, 8.455, 7.615, 7, 6.529,
    6.158, 5.857])
```

```
np.random.seed(1234) # 通常は指定しない
inival = [1, 1, 1]
a = nonlinear_fitting(x, y, inival, model="Hyperbola1", method="
    simplex")
graph(x, y, a)
```

```
Hyperbola1      by Simplex method

                          estimates
a                          2.226427
b                         36.867817
c                          0.743333
residual sum of squares    1.202826

      x       y      pred.    resid.
```

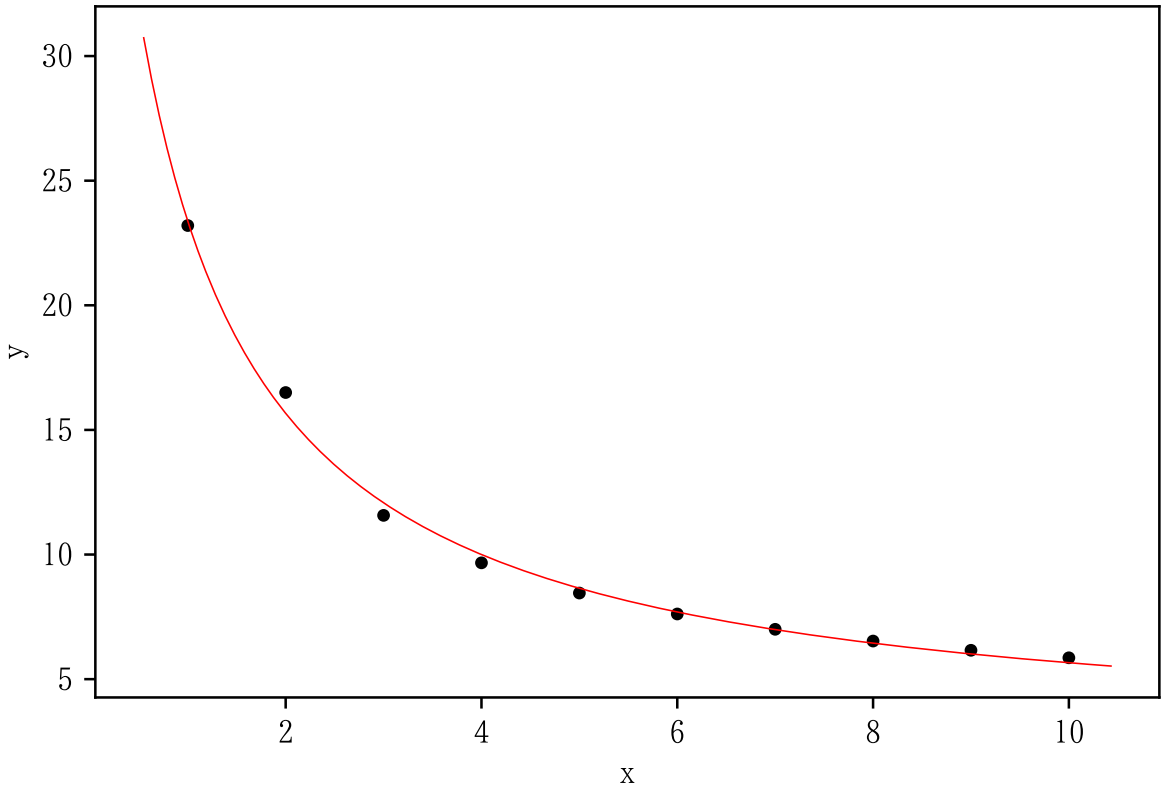
```
0   1.0  23.200  23.374322 -0.174322
1   2.0  16.500  15.665489  0.834511
2   3.0  11.571  12.075356 -0.504356
3   4.0   9.667   9.998983 -0.331983
4   5.0   8.455   8.645665 -0.190665
5   6.0   7.615   7.693727 -0.078727
6   7.0   7.000   6.987661  0.012339
7   8.0   6.529   6.443105  0.085895
8   9.0   6.158   6.010330  0.147670
9  10.0   5.857   5.658120  0.198880
```




```
b = nonlinear_fitting(x, y, a["p"], model="Hyperbola1")
graph(x, y, b)
```

```
Hyperbola1      by Marquardt method

                          estimates
a                          2.226828
b                         36.864234
c                          0.743211
residual sum of squares    1.202826

      x       y      pred.    resid.
```

```
0   1.0  23.200  23.374141 -0.174141
1   2.0  16.500  15.665178  0.834822
2   3.0  11.571  12.075119 -0.504119
3   4.0   9.667   9.998827 -0.331827
4   5.0   8.455   8.645577 -0.190577
5   6.0   7.615   7.693694 -0.078694
6   7.0   7.000   6.987674  0.012326
7   8.0   6.529   6.443154  0.085846
8   9.0   6.158   6.010409  0.147591
9  10.0   5.857   5.658226  0.198774
```



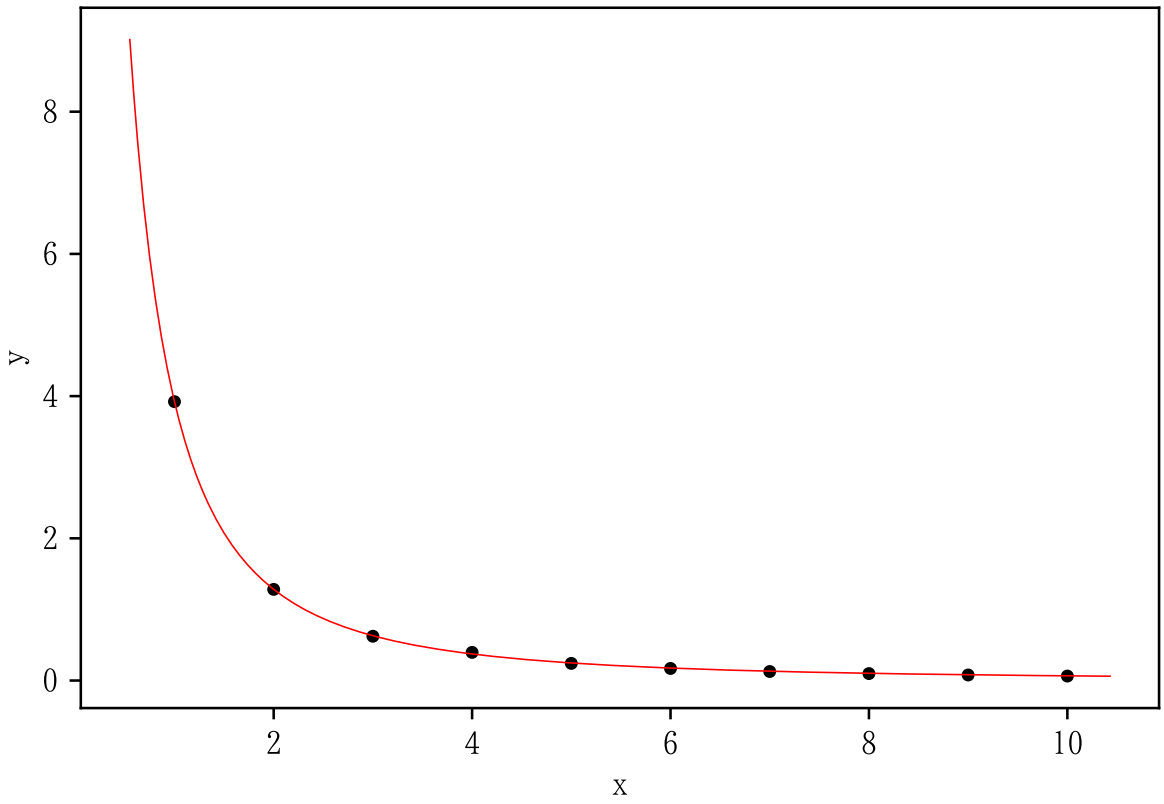
3.14 model = "Hyperbola2"

```
import numpy as np

x = np.arange(1, 11)
y =   np.array([3.922, 1.282, 0.623, 0.396, 0.241, 0.17, 0.127, 0.098,
    0.078, 0.063])
```

```
np.random.seed(1234) # 通常は指定しない
inival = [1, 1, 1, 1]
a = nonlinear_fitting(x, y, inival, model="Hyperbola2", method="
```

```
    simplex")
graph(x, y, a)
```

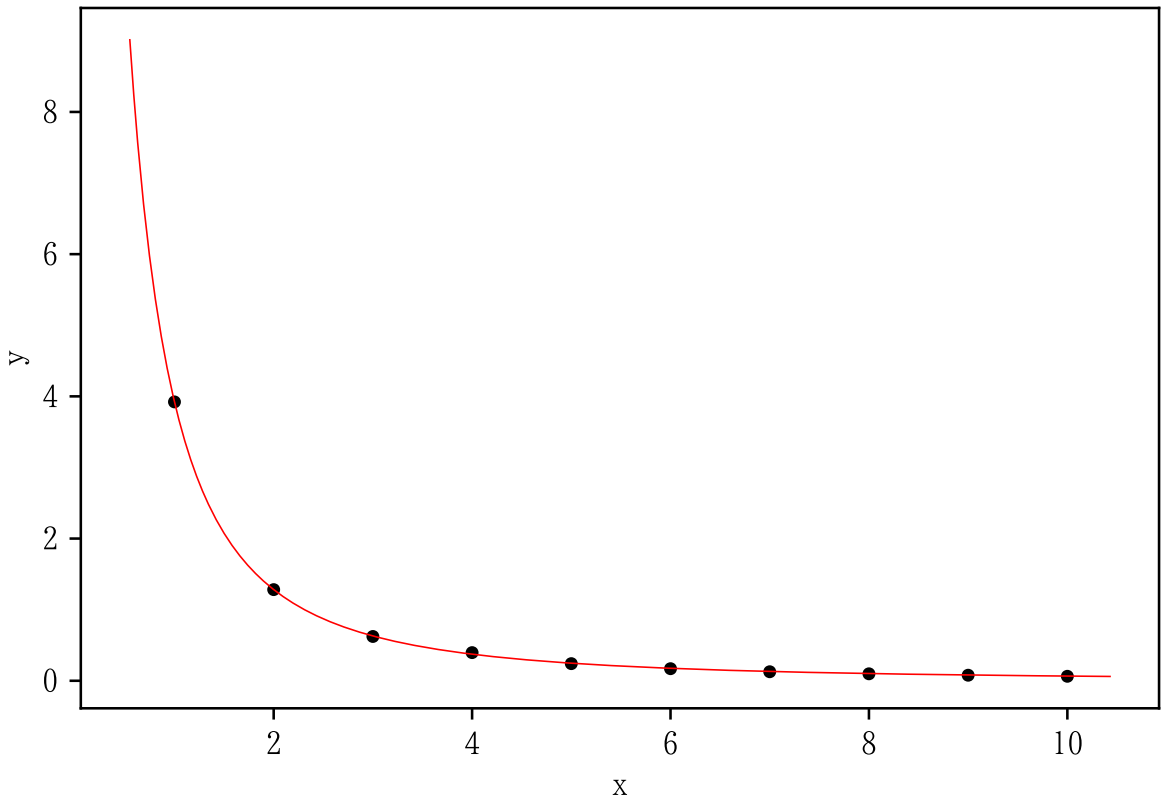
```
Hyperbola2      by Simplex method

                          estimates
a                          1.146401
b                          0.161888
c                          0.116019
d                          0.014397
residual sum of squares    0.000661

      x      y     pred.    resid.
0   1.0  3.922  3.921950  0.000050
1   2.0  1.282  1.282346 -0.000346
2   3.0  0.623  0.630082 -0.007082
3   4.0  0.396  0.373581  0.022419
4   5.0  0.241  0.246979 -0.005979
5   6.0  0.170  0.175331 -0.005331
6   7.0  0.127  0.130882 -0.003882
7   8.0  0.098  0.101421 -0.003421
8   9.0  0.078  0.080895 -0.002895
9  10.0  0.063  0.066024 -0.003024
```




```
b = nonlinear_fitting(x, y, a["p"], model="Hyperbola2")
graph(x, y, b)
```

```
Hyperbola2      by Marquardt method

                          estimates
a                          1.146400
b                          0.161882
c                          0.116038
d                          0.014384
residual sum of squares    0.000661

      x      y     pred.    resid.
0   1.0  3.922  3.921941  0.000059
1   2.0  1.282  1.282343 -0.000343
2   3.0  0.623  0.630085 -0.007085
3   4.0  0.396  0.373584  0.022416
4   5.0  0.241  0.246982 -0.005982
5   6.0  0.170  0.175334 -0.005334
6   7.0  0.127  0.130884 -0.003884
7   8.0  0.098  0.101423 -0.003423
8   9.0  0.078  0.080897 -0.002897
```

```
 9  10.0  0.063  0.066026 -0.003026
```



3.15 model = ユーザが定義する関数名

ユーザが定義する関数への当てはめを行える。

ここでは，関数が $f(a,b,c,d,x) = \dfrac{c}{ax+b} + d$ の場合を借りて説明する。

関数は a, b, c, d の 4 つのパラメータを持つ関数である。4 つのパラメータそれぞれについて偏微分した式を求めておく。

以下のような **Python** の関数を定義する。

関数名は任意のものでよい。以下の例では `user_defined` である。

引数は パラメータベクトル：`p`, 独立変数ベクトル：`x_value`, 偏微分式：`fj=None` の 3 個で，この通り記述する（引数の名前は対応さえ着けておけば任意の名前にしてよい）。

Python の関数内ではパラメータは `p[0]`, `p[1]`, `p[2]`, `...` として使う。パラメータの順序は任意である。以下では，アルファベット順を採用した。

ユーザが定義する必要があるのは `fj[:n, 0]`, `fj[:n, 1]`, `fj[:n, 2]`, `...` の右辺の式である。

`fj[:n, 0]` は元の関数 $f(a,b,c,d,x)$ を a で偏微分した式である。すなわち，$-\dfrac{cx}{(ax+b)^2}$ の a, b, c を `p[0]`, `p[1]`, `p[2]` を使って書いている。

`fj[:n, 1]` は元の関数 $f(a,b,c,d,x)$ を b で偏微分した式... などとなる。

最後の `return` の次の式は $f(a,b,c,d,x)$ そのものである。

```
def user_defined(p, x_value, fj=None):  # y = c / (a * x + b) + d
    n = len(x_value)
```

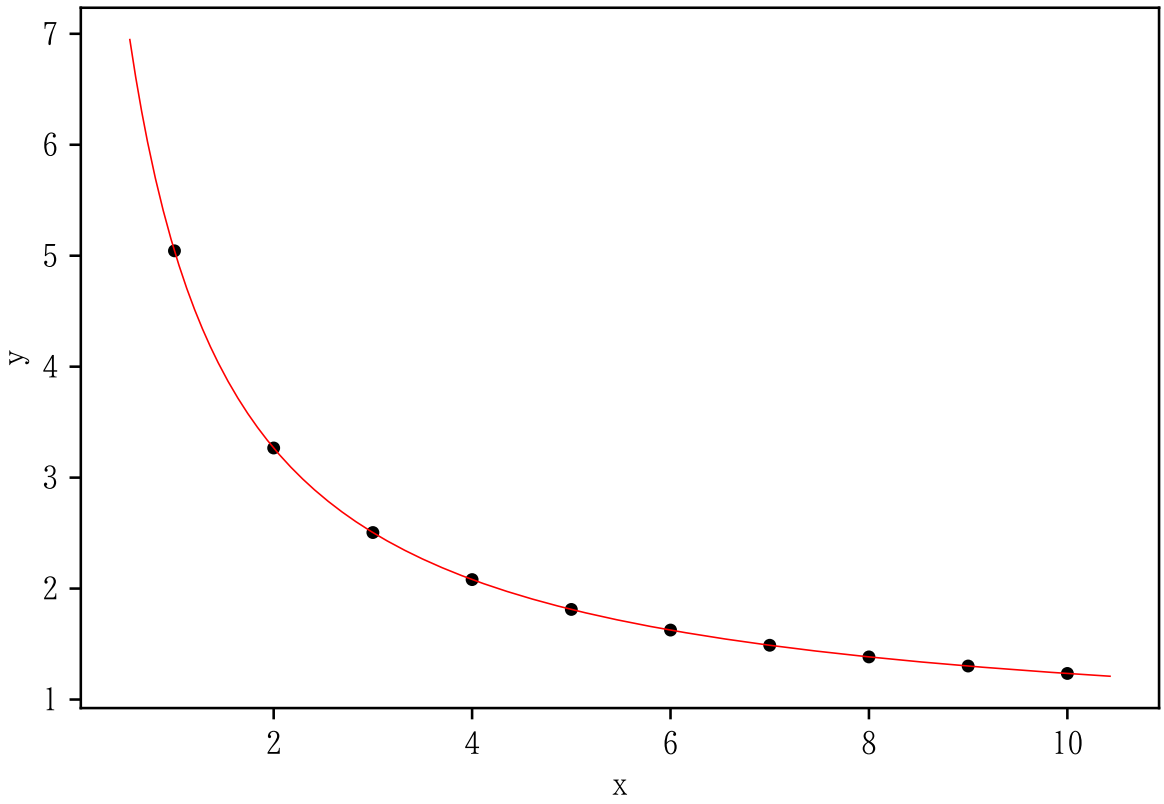
```
    if fj is not None:
        fj[:n, 0] = - p[2] * x_value / (p[0] * x_value + p[1])**2
        fj[:n, 1] = -p[2] / (p[0] * x_value + p[1])**2
        fj[:n, 2] = 1 / (p[0] * x_value + p[1])
        fj[:n, 3] = 1
    return p[2] / (p[0] * x_value + p[1]) + p[3]
```

nonlinear_fitting() を使用するとき，model に関数名を指定する。文字列で指定するのではない。上の例であれば，関数名は user_defined なので，model=user_defined とする。

```
import numpy as np

x = np.arange(1, 11)
y = np.array([5.044, 3.267, 2.505, 2.081, 1.812, 1.626, 1.489, 1.384,
    1.302, 1.235])
```

```
inival = [1, 1, 1, 1]
a = nonlinear_fitting(x, y, inival, model=user_defined, method="
    simplex")
graph(x, y, a)
```

```
user defined function      by Simplex method

                             estimates
a                         3.976140e-01
b                         1.990073e-01
c                         2.651524e+00
d                         5.998305e-01
residual sum of squares   7.412983e-07

      x      y     pred.    resid.
0   1.0  5.044  5.044063 -0.000063
1   2.0  3.267  3.266728  0.000272
2   3.0  2.505  2.504867  0.000133
3   4.0  2.081  2.081573 -0.000573
4   5.0  1.812  1.812190 -0.000190
5   6.0  1.626  1.625687  0.000313
6   7.0  1.489  1.488916  0.000084
7   8.0  1.384  1.384324 -0.000324
8   9.0  1.302  1.301750  0.000250
9  10.0  1.235  1.234904  0.000096
```




```
b = nonlinear_fitting(x, y, a["p"], model=user_defined)
graph(x, y, b)
```

```
user defined function      by Marquardt method

                             estimates
a                         3.976140e-01
b                         1.990073e-01
c                         2.651524e+00
d                         5.998305e-01
residual sum of squares   7.412983e-07

      x      y     pred.    resid.
0   1.0  5.044  5.044063 -0.000063
1   2.0  3.267  3.266728  0.000272
2   3.0  2.505  2.504867  0.000133
3   4.0  2.081  2.081573 -0.000573
4   5.0  1.812  1.812190 -0.000190
5   6.0  1.626  1.625687  0.000313
6   7.0  1.489  1.488916  0.000084
7   8.0  1.384  1.384324 -0.000324
8   9.0  1.302  1.301750  0.000250
```

```
9   10.0   1.235   1.234904   0.000096
```