指数回帰 $y = a * b^x$

青木繁伸

2020 年 3 月 17 日

1　目的

指数回帰曲線 $y = a * b^x$ への当てはめを行う。

両辺の対数をると $\log y = \log a + x \log b$ のようになるので，従属変数 y の対数を取ったもの $\log y$ を独立変数 x で予測する単回帰式を解けばよい。

2　使用法

```
import sys
sys.path.append("statlib")
from multi import exp1
```

2.1　引数

x	独立変数ベクトル（リストでもよい）
y	従属変数ベクトル（リストでもよい）
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

"b"	a
"b"	b
"predicted"	予測値
"resid"	残差（観察値 – 予測値）
"x"	独立変数 x
"y"	従属変数 y
"method"	分析手法名

3　使用例

```
import sys
```

```
sys.path.append("statlib")
from multi import exp1

import matplotlib.pyplot as plt

def graph(x, y, a, b):
    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 * b**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 使用例 1

```
import numpy as np

x = np.arange(1, 11)
y = np.array([3, 4.5, 6.75, 10.125, 15.188, 22.781, 34.172, 51.258,
    76.887, 115.330])
ans = exp1(x, y)
```

```
y = a * b**x
a = 2.00001, b = 1.5
```

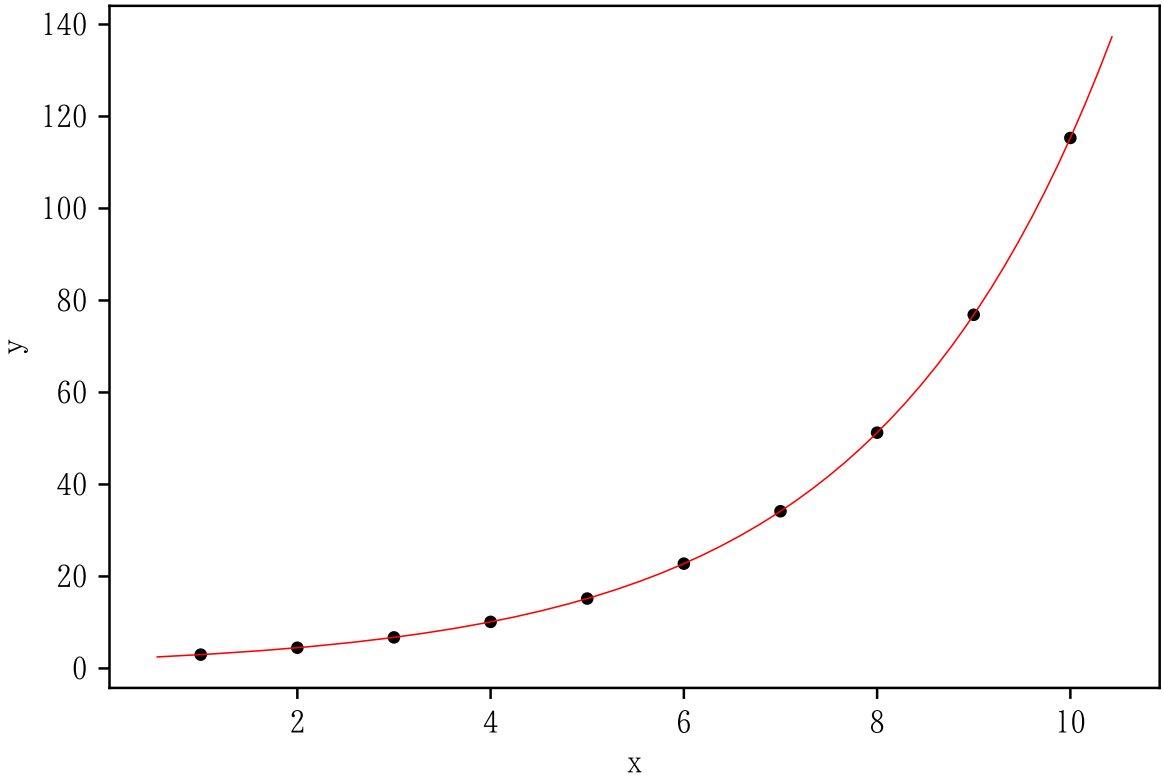
	x	y	pred.	resid.
0	1	3.000	3.000009	-0.000009
1	2	4.500	4.500014	-0.000014
2	3	6.750	6.750021	-0.000021
3	4	10.125	10.125032	-0.000032
4	5	15.188	15.187549	0.000451
5	6	22.781	22.781324	-0.000324
6	7	34.172	34.171987	0.000013
7	8	51.258	51.257982	0.000018
8	9	76.887	76.886975	0.000025
9	10	115.330	115.330465	-0.000465

```
print(ans)
```

```
{'a': 2.0000061235873563, 'b': 1.5000000443425876, 'predicted': array([  3.00000927,   4.500014
        15.18754875,  22.78132379,  34.1719867 ,  51.25798156,
        76.88697462, 115.33046534]), 'resid': array([-9.27406648e-06, -1.40441279e-05, -2.12657
```

```
4.51254158e-04, -3.23792218e-04,  1.33014899e-05,  1.84369605e-05,
2.53825292e-05, -4.65335574e-04]), 'x': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10]),
51.258,  76.887, 115.33 ]), 'model': 'y = a * b**x'}
```

```
graph(x, y, ans["a"], ans["b"])
```



3.2 使用例 2

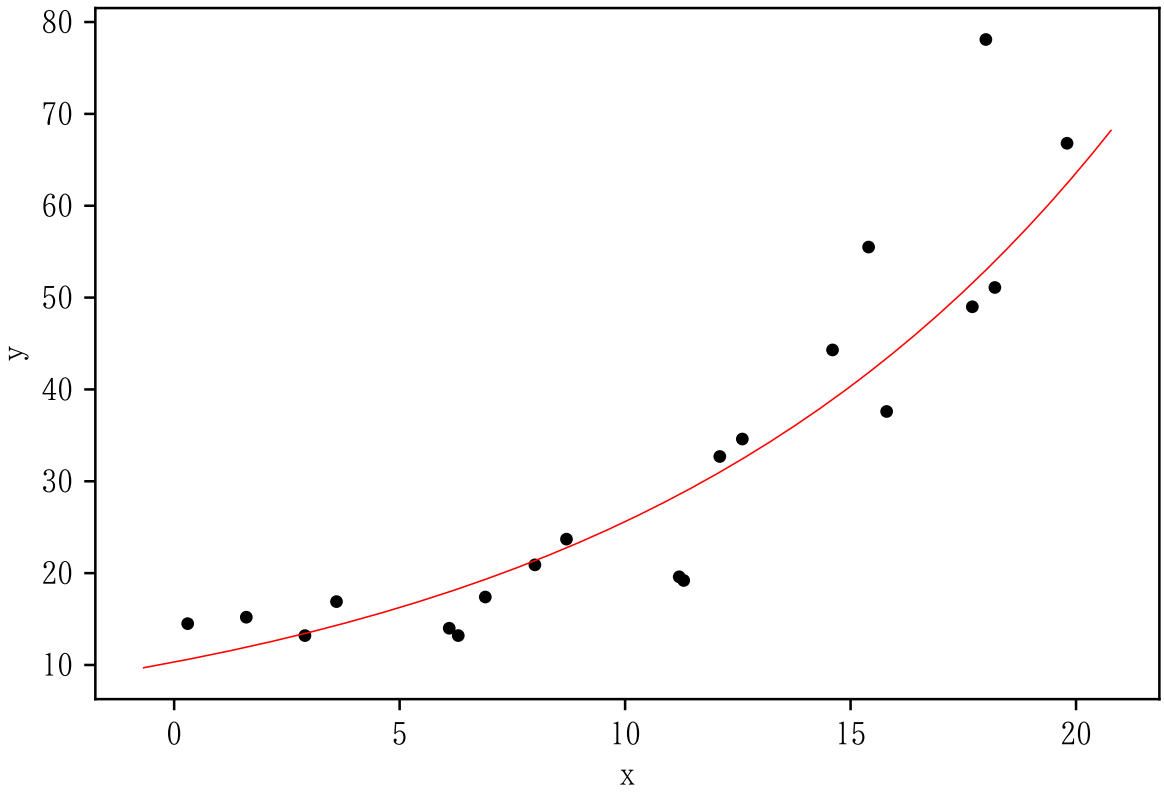
```
x2 = [0.3, 2.9, 3.6, 6.1, 6.9, 8, 11.3, 12.1, 15.8, 15.4, 18, 19.8,
      17.7, 8.7, 1.6, 6.3, 11.2, 12.6, 14.6, 18.2]
y2 = [14.5, 13.2, 16.9, 14, 17.4, 20.9, 19.2, 32.7, 37.6, 55.5, 78.1,
      66.8, 49, 23.7, 15.2, 13.2, 19.6, 34.6, 44.3, 51.1]

ans2 = exp1(x2, y2)
graph(x2, y2, ans2["a"], ans2["b"])
```

```
y = a * b**x
a = 10.3156, b = 1.09518

       x     y      pred.     resid.
0    0.3  14.5  10.600803   3.899197
1    2.9  13.2  13.427556  -0.227556
2    3.6  16.9  14.309871   2.590129
```

3	6.1	14.0	17.961621	-3.961621
4	6.9	17.4	19.316688	-1.916688
5	8.0	20.9	21.348383	-0.448383
6	11.3	19.2	28.817874	-9.617874
7	12.1	32.7	30.991963	1.708037
8	15.8	37.6	43.385000	-5.785000
9	15.4	55.5	41.835603	13.664397
10	18.0	78.1	52.991260	25.108740
11	19.8	66.8	62.413036	4.386964
12	17.7	49.0	51.565480	-2.565480
13	8.7	23.7	22.751170	0.948830
14	1.6	15.2	11.930753	3.269247
15	6.3	13.2	18.291205	-5.091205
16	11.2	19.6	28.557063	-8.957063
17	12.6	34.6	32.433293	2.166707
18	14.6	44.3	38.900832	5.399168
19	18.2	51.1	53.963614	-2.863614



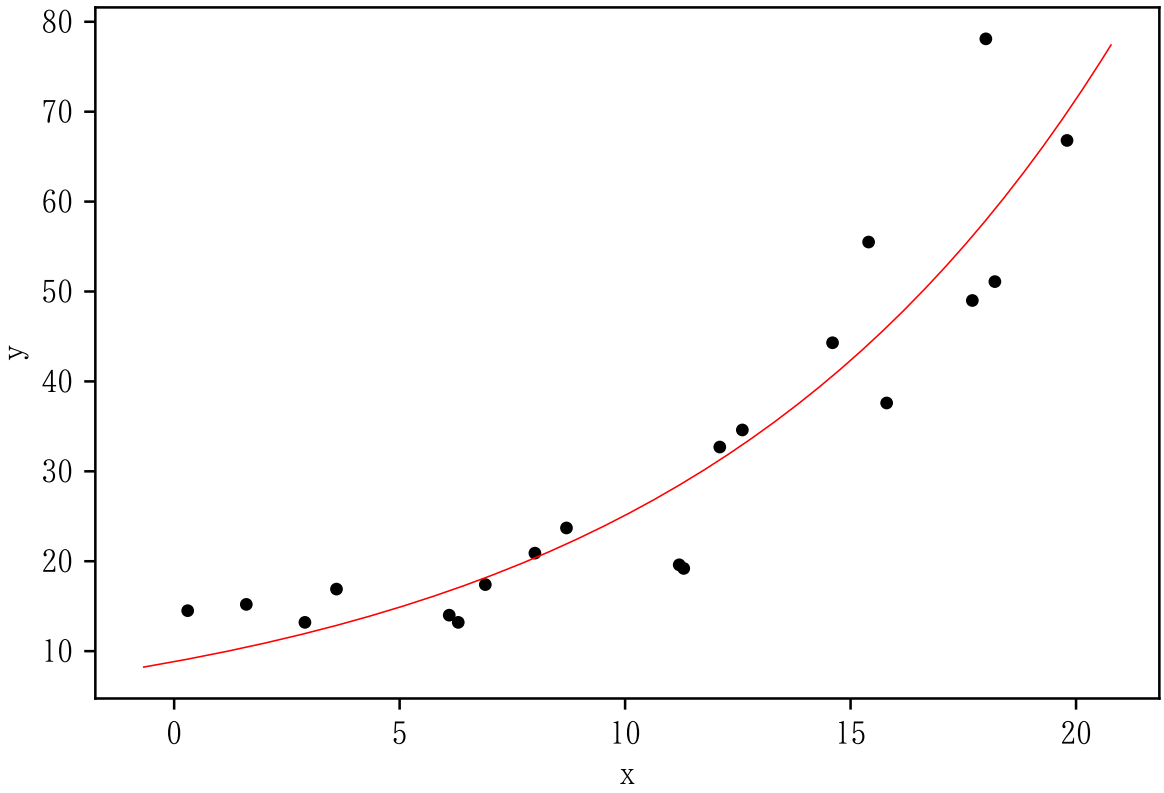
非線形回帰の結果は少し異なる。

```
from multi import nonlinear_fitting
ans3 = nonlinear_fitting(x2, y2, [1, 1], model="Exponential1")
graph(x2, y2, ans3["p"][0], ans3["p"][1])
```

Exponential1 by Marquardt method

	estimates
a	8.828283
b	1.110170
residual sum of squares	1016.594190

	x	y	pred.	resid.
0	0.3	14.5	9.109470	5.390530
1	2.9	13.2	11.953800	1.246200
2	3.6	16.9	12.861117	4.038883
3	6.1	14.0	16.701391	-2.701391
4	6.9	17.4	18.157845	-0.757845
5	8.0	20.9	20.370085	0.529915
6	11.3	19.2	28.759309	-9.559309
7	12.1	32.7	31.267281	1.432719
8	15.8	37.6	46.029001	-8.429001
9	15.4	55.5	44.144411	11.355589
10	18.0	78.1	57.928012	20.171988
11	19.8	66.8	69.918140	-3.118140
12	17.7	49.0	56.139915	-7.139915
13	8.7	23.7	21.916215	1.783785
14	1.6	15.2	10.435171	4.764829
15	6.3	13.2	17.054169	-3.854169
16	11.2	19.6	28.460301	-8.860301
17	12.6	34.6	32.944651	1.655349
18	14.6	44.3	40.603555	3.696445
19	18.2	51.1	59.151606	-8.051606


80
70
60
50
40
30
20
10
y
0
5
10
15
20
x