

# Gompertz 曲線を使った累積バグ数推定

---

- 1. 概要
  - 2. 数式
    - 2.1. Gompertz 曲線
    - 2.2. 線形変換
    - 2.3. 正規方程式
  - 3. プログラム
    - 3.1. Gompertz 方程式
    - 3.2. 正規方程式
    - 3.3. fit 関数
    - 3.4. データ
    - 3.5. main
  - 4. 結果
  - 5. コード
  - 6. 改善点
-

# 1. 概要

累積バグ数は Gompertz 曲線によく近似される。そこで実際の累積バグ数の時間推移データに最もよくあてはまる Gompertz 曲線のパラメータを重回帰分析で推定し、残存バグ数を予測する。

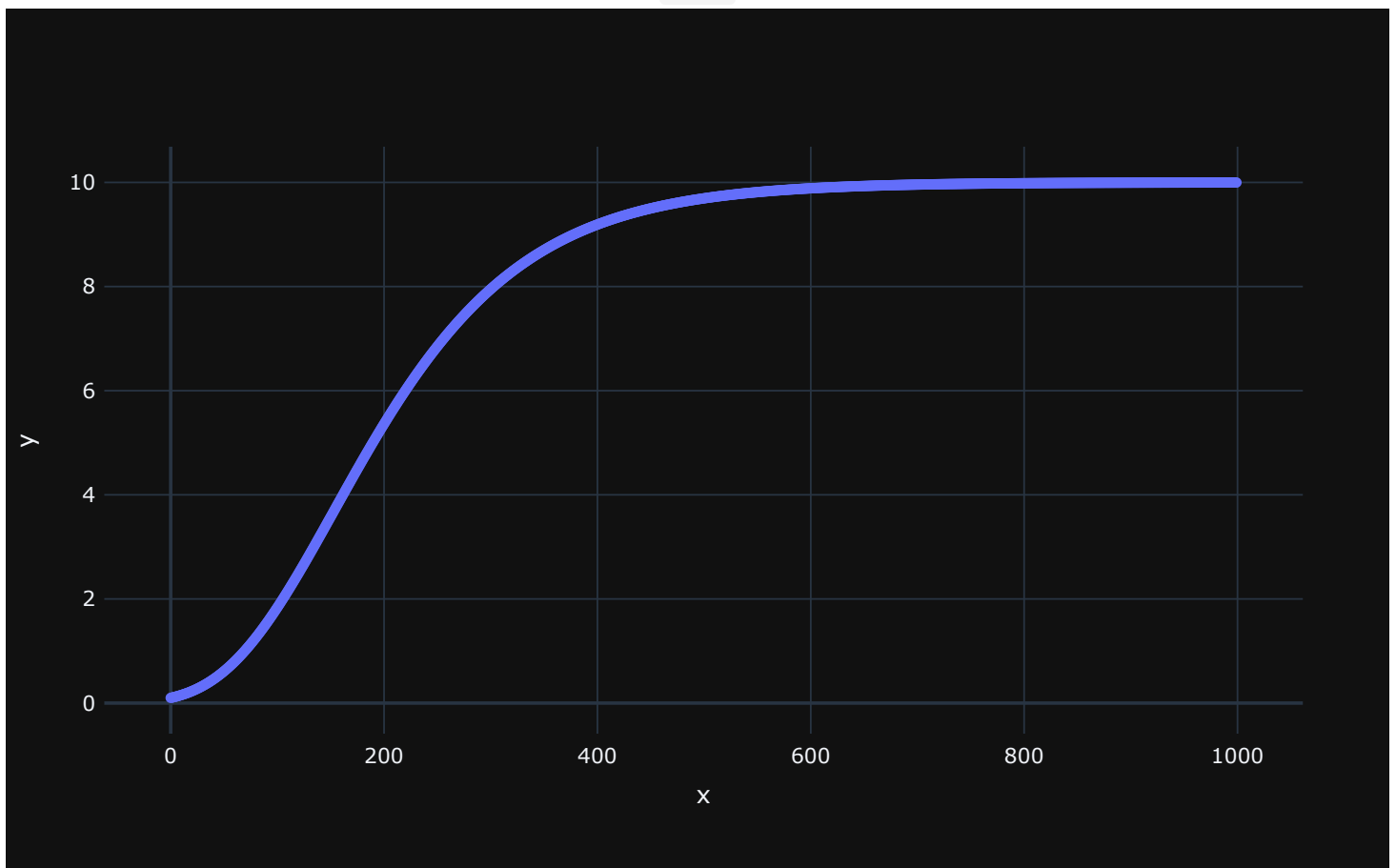
## 2. 数式

### 2.1. Gompertz 曲線

初期段階では指数関数的に増加し、その後成長率が減少する。最終的にある一定の値( $k$ )に収束する。テスト初期にはバグが発見され修正されていくが、テスト回数に応じて次第にシステムに潜むバグが減っていき、テスト当たりにバグを引く確率が下がっていく。

$$y = kb^{exp(-cx)}$$

k=10



Gompertz 方程式は非線形方程式なので線形に変換する。

## 2.2. 線形変換

両辺に自然対数をとる。

$$\ln y = \ln k + \exp(-cx) \ln b$$

ここで $\exp(-cx)$ は非線形なので逐次的に近似する手法をとる。

パラメータ $c$ の初期近似値を $c'$ としてその誤差を $\delta$ とすると、

$$\exp(-cx) \doteq \exp(-(c' + \delta)x) = \exp(-c'x) \exp(-\delta x) \quad (1)$$

$$\doteq \exp(-c'x)(1 - \delta x) \quad (2)$$

(2)ではマクローリン展開を使って一次の項までで近似した。

$$\exp(-\delta x) \doteq 1 + (-\delta x) + \frac{(-\delta x)^2}{2!} + \frac{(-\delta x)^3}{3!} + \dots$$

(2)式から

$$\ln y = \ln k + \exp(-cx) \ln b \quad (3)$$

$$= \ln k + \exp(-c'x)(1 - \delta x) \ln b \quad (4)$$

$$= \ln k + \exp(-c'x) \ln b - \delta x \exp(-c'x) \ln b \quad (5)$$

従って以下の重回帰式が得られる。

$$Y = \alpha + \beta X_1 + \gamma X_2$$

$$Y = \ln y$$

$$\alpha = \ln k$$

$$\beta = \ln b$$

$$X_1 = \exp(-c'x)$$

$$X_2 = x \exp(-c'x)$$

$$\gamma = -\delta \beta$$

$c$ の近似誤差 $\delta = -\gamma/\beta$ が十分小さくなるまで重回帰式を繰り返し計算する。 $c$ は以下の差分方程式で近似値を更新する。

$$c_{t+1} = c_t + \delta$$

## 2.3. 正規方程式

最小二乗法

$$L(\theta) = \text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

$y_i$  : 真値  
 $\hat{y}_i$  : 予測値

以下の重回帰式で予測する。

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{W} + \boldsymbol{\epsilon}$$
$$\hat{\mathbf{y}} = \begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_i \end{pmatrix}$$
$$\mathbf{X} = \begin{pmatrix} 1 & X_{10} & X_{20} \\ 1 & X_{11} & X_{21} \\ \vdots & \vdots & \vdots \\ 1 & X_{1i} & X_{2i} \end{pmatrix}$$
$$\mathbf{W} = \begin{pmatrix} \theta_1 = \alpha \\ \theta_2 = \beta \\ \theta_3 = \gamma \end{pmatrix}$$

$\boldsymbol{\epsilon}$  : 誤差項

誤差関数は以下ようになる。

$$L(\theta) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \tag{6}$$

$$= \frac{1}{n} \left( (\mathbf{y} - \mathbf{X}\mathbf{W})^\top (\mathbf{y} - \mathbf{X}\mathbf{W}) \right) \tag{7}$$

$$= \frac{1}{n} \left( \mathbf{y}^\top \mathbf{y} - \mathbf{y}^\top \mathbf{X}\mathbf{W} - \mathbf{W}^\top \mathbf{X}^\top \mathbf{y} + \mathbf{W}^\top \mathbf{X}^\top \mathbf{X}\mathbf{W} \right) \tag{8}$$

$$= \frac{1}{n} \left( \mathbf{y}^\top \mathbf{y} - 2\mathbf{W}^\top \mathbf{X}^\top \mathbf{y} + \mathbf{W}^\top \mathbf{X}^\top \mathbf{X}\mathbf{W} \right) \tag{9}$$

両辺を $\mathbf{W}$ で偏微分する。

$$\frac{\partial L(\theta)}{\partial W} = \frac{1}{n} \frac{\partial}{\partial W} \left( \mathbf{y}^\top \mathbf{y} - 2\mathbf{W}^\top \mathbf{X}^\top \mathbf{y} + \mathbf{W}^\top \mathbf{X}^\top \mathbf{X} \mathbf{W} \right) \quad (10)$$

$$= \frac{1}{n} \left( -2\mathbf{X}^\top \mathbf{y} + 2\mathbf{X}^\top \mathbf{X} \mathbf{W} \right) \quad (11)$$

$\frac{\partial L(\theta)}{\partial W} = 0$  のとき、 $L(\theta)$  は最小になるので以下のように正規方程式を導ける。

$$\frac{\partial L(\theta)}{\partial W} = \frac{1}{n} \left( -2\mathbf{X}^\top \mathbf{y} + 2\mathbf{X}^\top \mathbf{X} \mathbf{W} \right) = 0$$

$$\therefore \mathbf{X}^\top \mathbf{X} \mathbf{W} = \mathbf{X}^\top \mathbf{y} \quad (12)$$

$$\mathbf{W} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} \quad (13)$$

補足

$$(8) \because \mathbf{y}^\top \mathbf{X} \mathbf{W} = (\mathbf{W}^\top \mathbf{X}^\top \mathbf{y})^\top = \mathbf{W}^\top \mathbf{X}^\top \mathbf{y}$$

$$(10) \because \frac{\partial}{\partial W} \mathbf{W}^\top \mathbf{X}^\top \mathbf{y} = \mathbf{X}^\top \mathbf{y}$$

$$(10) \because \frac{\partial}{\partial W} \mathbf{W}^\top \mathbf{X}^\top \mathbf{X} \mathbf{W} = 2\mathbf{X}^\top \mathbf{X} \mathbf{W}$$

## 3. プログラム

### 3.1. Gompertz 方程式

```
def gompertz(x: np.array, k=10, b=0.01, c=0.01) -> np.array:
    '''Gompertz
    ...
    return k * np.power(b, np.exp(-c * x))
```

### 3.2. 正規方程式

```
def normal_eq(X: np.array, Y: np.array) -> np.array:
    '''正規方程式
    ...
    return np.linalg.pinv(X.T @ X) @ X.T @ Y # 一般逆行列
```

### 3.3. fit 関数

```
def fit(t: np.array, input_y: np.array) -> dict:
    """
    cの設定値と実際の値には誤差がある。
    その誤差でc設定値を補正し新たな回帰モデルを作成、これを繰り返す。
    """
    c_init = 0.02
    c = c_init
    Y_i = np.log(input_y)

    i = 0
    while True:
        t_x = np.insert(
            np.vstack([np.exp(-c * t), t * np.exp(-c * t)]).T,
            0, 1,
            axis=1
        )
        alpha, beta, gamma = normal_eq(t_x, Y_i)
        delta = -gamma / beta
        c += delta
        i += 1
        if abs(delta) < 1e-15 or i >= 100:
            break

    print(f"c初期値 : {c_init} -> {i}回目 : {c}")
    print(f"収束予測値 : {np.round(np.power(np.e, alpha))}")

    return {
        "k": np.power(np.e, alpha),
        "b": np.power(np.e, beta),
        "c": c
    }
```

### 3.4. データ

テスト日	バグ発見個数	テスト回数
1	4	15
2	1	25
3	13	102
4	0	18
5	3	97
6	0	26
7	2	158
8	0	131

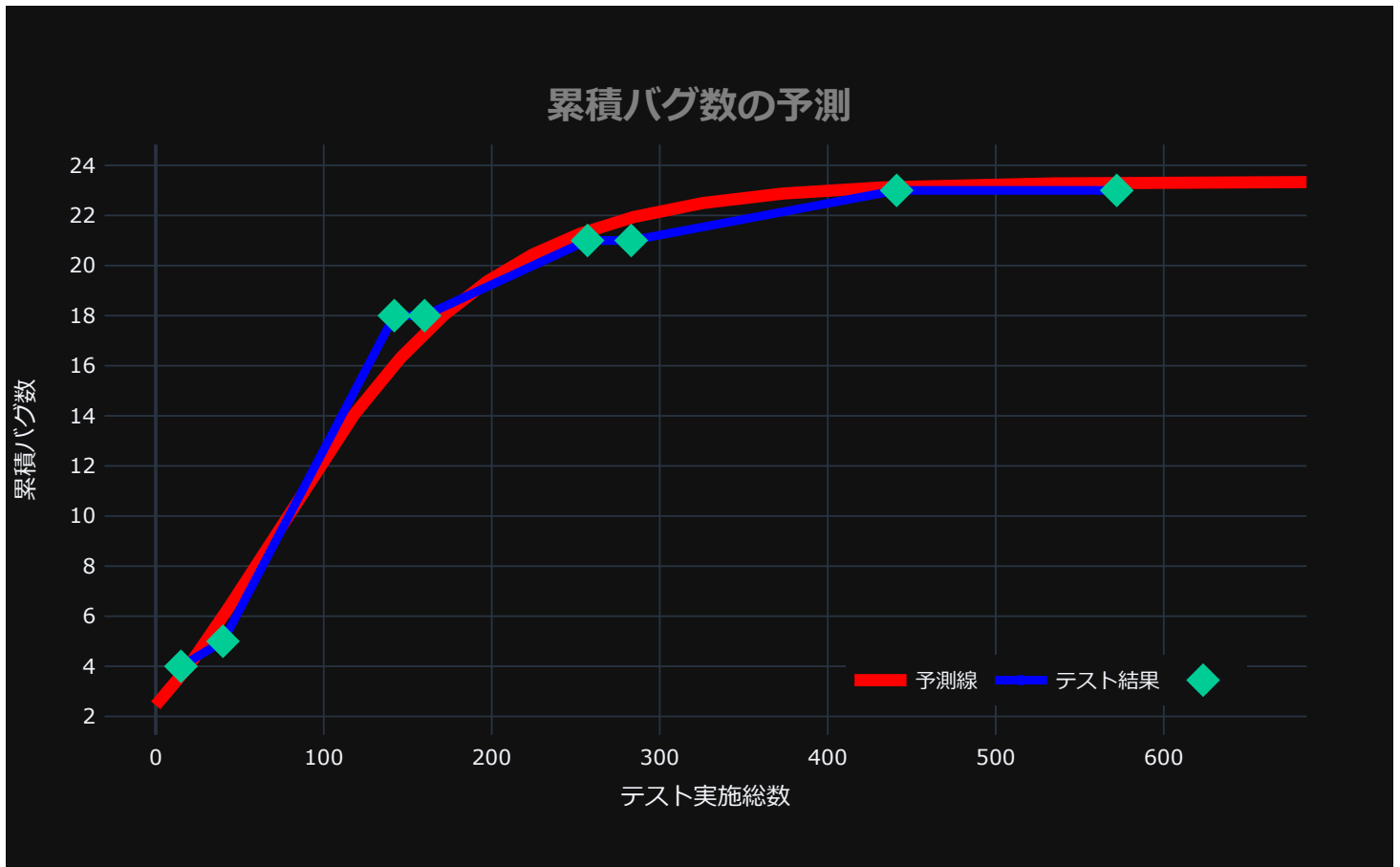
### 3.5. main

```
data = pd.DataFrame(  
    {  
        "bug": [4, 1, 13, 0, 3, 0, 2, 0],  
        "test": [15, 25, 102, 18, 97, 26, 158, 131]  
    }  
)  
display(data)  
data = pd.DataFrame(  
    {  
        "Cumsum_bug": np.cumsum(data["bug"]),  
        "Cumsum_test": np.cumsum(data["test"])  
    }  
)  
display(data)  
parameter = fit(data["Cumsum_test"], data["Cumsum_bug"])  
x_arr = np.arange(0, np.round(data["Cumsum_test"].max() * 1.2))  
result = gompertz(x_arr, **parameter)
```

## 4. 結果

c初期値 : 0.02 -> 23回目 : 0.012628319903561528

収束予測値 : 23.0



バグ総数の予測値は 23 個となった。十分収束していると言えるため、これ以上テストする必要はない。



## I 5. コード

```
1  # %%
2  import numpy as np
3  import pandas as pd
4  from IPython.display import display
5  import plotly.graph_objects as go
6  import plotly.express as px
7  # %%
8  data = pd.DataFrame(
9      {
10         "bug": [4, 1, 13, 0, 3, 0, 2, 0],
11         "test": [15, 25, 102, 18, 97, 26, 158, 131]
12     }
13 )
14 display(data)
15 data = pd.DataFrame(
16     {
17         "Cumsum_bug": np.cumsum(data["bug"]),
18         "Cumsum_test": np.cumsum(data["test"])
19     }
20 )
21 display(data)
22 # %%
23
24
25 def gompertz(x: np.array, k=10, b=0.01, c=0.01) -> np.array:
26     '''Gompertz
27     '''
28     return k * np.power(b, np.exp(-c * x))
29
30
31 def normal_eq(X: np.array, Y: np.array) -> np.array:
32     '''正規方程式
33     '''
34     return np.linalg.pinv(X.T @ X) @ X.T @ Y # 一般逆行列
35
36
37 def fit(t: np.array, input_y: np.array) -> dict:
38     '''
39     cの設定値と実際の値には誤差がある。
40     その誤差でc設定値を補正し新たな回帰モデルを作成、これを繰り返す。
41     '''
42     c_init = 0.02
43     c = c_init
44     Y_i = np.log(input_y)
45
46     i = 0
47     while True:
48
```

```

49     t_x = np.insert(
50         np.vstack([np.exp(-c * t), t * np.exp(-c * t)]).T,
51         0, 1,
52         axis=1
53     )
54     alpha, beta, gamma = normal_eq(t_x, Y_i)
55     delta = -gamma / beta
56     c += delta
57     i += 1
58     if abs(delta) < 1e-15 or i >= 100:
59         break
60
61     print(f"c初期値 : {c_init} -> {i}回目 : {c}")
62     print(f"収束予測値 : {np.round(np.power(np.e, alpha))}")
63
64     return {
65         "k": np.power(np.e, alpha),
66         "b": np.power(np.e, beta),
67         "c": c
68     }
69
70
71 # %%
72 parameter = fit(data["Cumsum_test"], data["Cumsum_bug"])
73 x_arr = np.arange(0, np.round(data["Cumsum_test"].max() * 1.2))
74 result = gompertz(x_arr, **parameter)
75 # %%
76 chart = [
77     go.Scatter(
78         x=x_arr, y=result,
79         line={"dash": "solid", "width": 7, "color": "red"},
80         name="予測線"
81     ),
82     go.Scatter(
83         x=data["Cumsum_test"], y=data["Cumsum_bug"],
84         line={"dash": "solid", "width": 5, "color": "blue"},
85         name="テスト結果"
86     ),
87     go.Scatter(
88         x=data["Cumsum_test"], y=data["Cumsum_bug"],
89         mode="markers", marker={"size": 15, "symbol": "diamond"},
90         name="",
91     )
92 ]
93 fig = go.Figure(chart)
94 fig.update_layout(
95     title={
96         "text": "<b>累積バグ数の予測</b>",
97         "font": {
98             "size": 22,
99             "color": "grey"

```

```

100     },
101     "x": 0.5,
102     "y": 0.9,
103     },
104     legend={
105         "xanchor": "right",
106         "yanchor": "bottom",
107         "x": 0.95,
108         "y": 0.05,
109         "orientation": "h"
110     },
111     xaxis={
112         "title": "テスト実施総数",
113         "dtick": 100
114     },
115     yaxis={
116         "title": "累積バグ数",
117         "dtick": 2
118     }
119 )
120 fig.update_layout(
121     template="plotly_dark",
122     autosize=False,
123     width=800,
124     height=500,
125     margin={
126         "l": 50,
127         "r": 50,
128         "t": 80,
129         "b": 80,
130         "pad": 4
131     },
132 )
133 fig.show()
134 fig.write_image("output_image.svg")
# %%

```

## 6. 改善点

- 計算が収束するかは $c$ の初期値による。大きすぎても小さすぎても収束しない。
- $c$ の値によっては $\mathbf{X}$ に $\text{inf}$ が含まれるようになるので逆行列を計算できなくなる。
- データの粒度が低い。