Gompertz 曲線を使った累積バグ数推定

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。

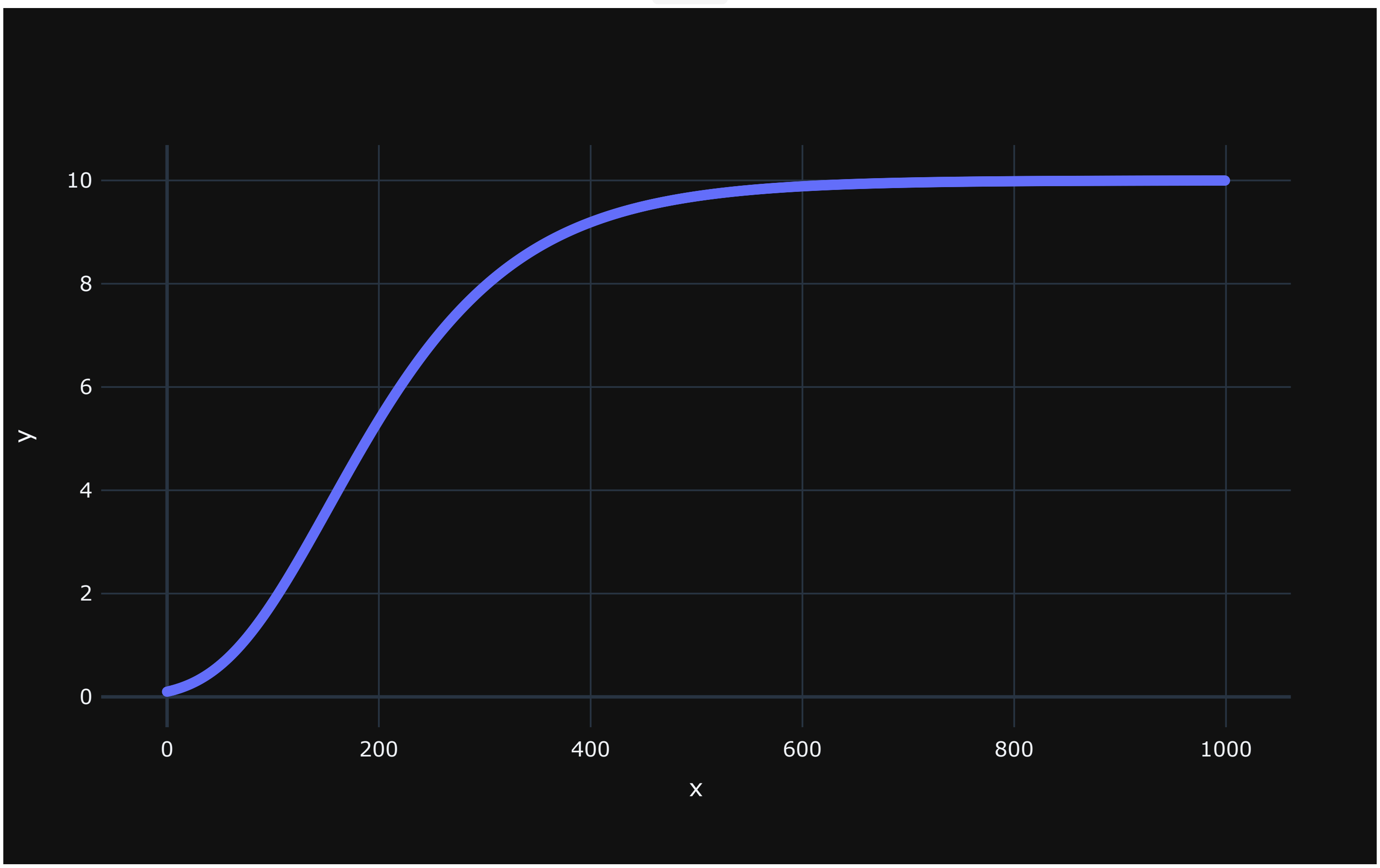
2. 数式

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δとすると、

$$\begin{aligned} \exp(-cx) &\fallingdotseq \exp(-(c'+\delta)x) = \exp(-c'x)\exp(-\delta x) \quad (1) \\ &\fallingdotseq \exp(-c'x)(1-\delta x) \quad (2) \end{aligned}$$

(2)ではマクローリン展開を使って一次の項まで近似した。

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

(2)式から

$$\begin{aligned} \ln y &= \ln k + \exp(-cx) \ln b \quad (3) \\ &= \ln k + \exp(-c'x)(1-\delta x) \ln b \quad (4) \\ &= \ln k + \exp(-c'x) \ln b - \delta x \exp(-c'x) \ln b \quad (5) \end{aligned}$$

従って以下の重回帰式が得られる。

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

$$\begin{aligned} Y &= \ln y \\ \alpha &= \ln k \\ \beta &= \ln b \\ X_1 &= \exp(-c'x) \\ X_2 &= x\exp(-c'x) \\ \gamma &= -\delta\beta \end{aligned}$$

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

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

以下の重回帰式で予測する。

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

誤差関数は以下のようになる。

$$\begin{aligned}
L(\theta) &= \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 && (6) \\
&= \frac{1}{n}\Big((\boldsymbol{y} - \boldsymbol{X}\boldsymbol{W})^\top(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{W})\Big) && (7) \\
&= \frac{1}{n}\Big(\boldsymbol{y}^\top\boldsymbol{y} - \boldsymbol{y}^\top\boldsymbol{X}\boldsymbol{W} - \boldsymbol{W}^\top\boldsymbol{X}^\top\boldsymbol{y} + \boldsymbol{W}^\top\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{W}\Big) && (8) \\
&= \frac{1}{n}\Big(\boldsymbol{y}^\top\boldsymbol{y} - \boldsymbol{2}\boldsymbol{W}^\top\boldsymbol{X}^\top\boldsymbol{y} + \boldsymbol{W}^\top\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{W}\Big) && (9)
\end{aligned}$$

両辺をWで偏微分する。

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

$$= \frac{1}{n}\left(-\boldsymbol{2X}^\top \boldsymbol{y} + \boldsymbol{2X}^\top \boldsymbol{XW}\right) \tag{11}$$

$\frac{\partial L(\theta)}{\partial W} = 0$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。

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

$$\therefore \boldsymbol{X}^\top \boldsymbol{XW} = \boldsymbol{X}^\top \boldsymbol{y} \tag{12}$$

$$\boldsymbol{W} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y} \tag{13}$$

補足

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

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

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

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

```python
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

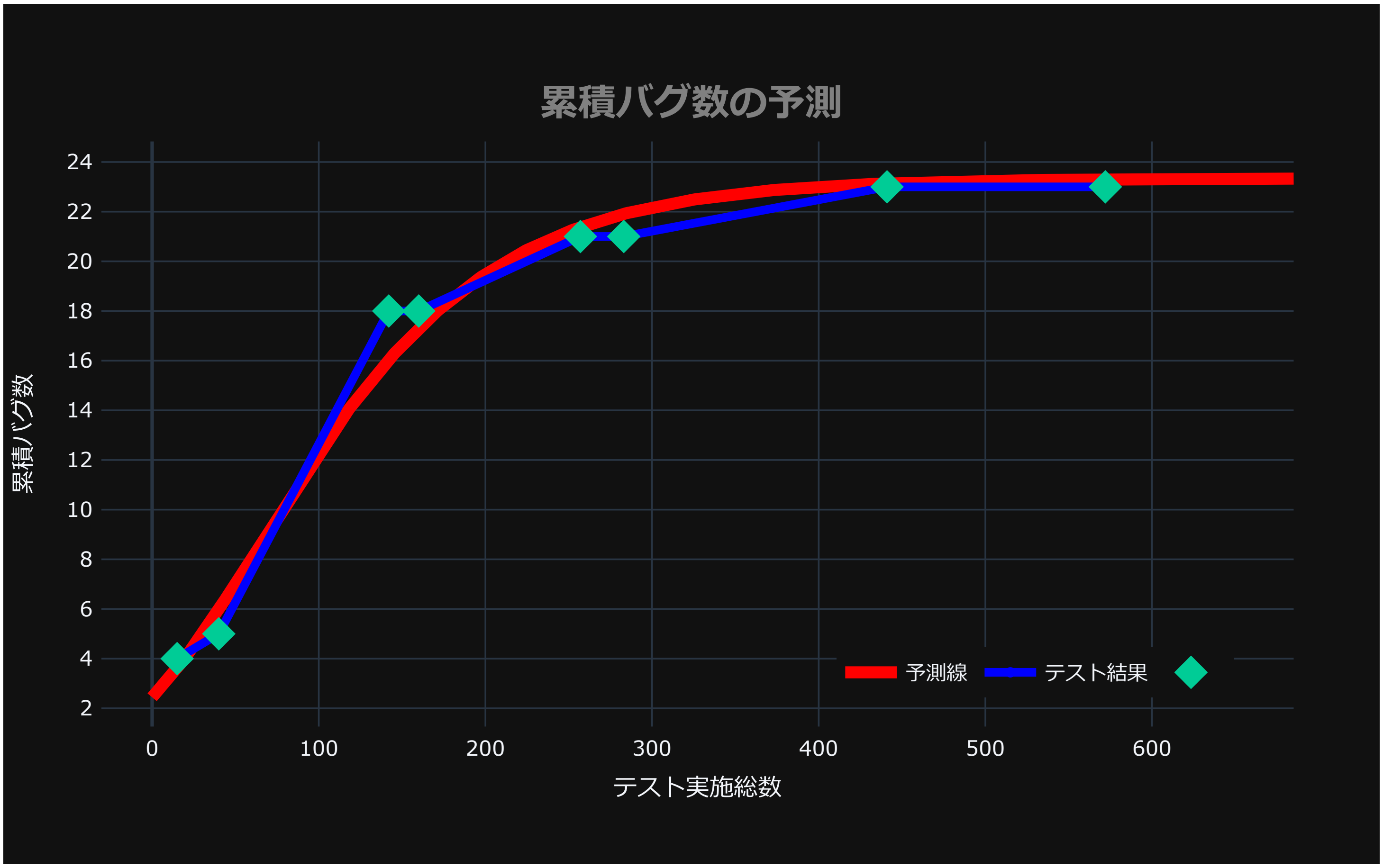
テスト日	バグ発見個数	テスト回数
1	4	15
2	1	25
3	13	102
4	0	18
5	3	97
6	0	26
7	2	158
8	0	131

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。

5. コード

```python
# %%
import numpy as np
import pandas as pd
from IPython.display import display
import plotly.graph_objects as go
import plotly.express as px
# %%
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)
# %%


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))


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


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
    }


# %%
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)
# %%
chart = [
    go.Scatter(
        x=x_arr, y=result,
        line={"dash": "solid", "width": 7, "color": "red"},
        name="予測線"
    ),
    go.Scatter(
        x=data["Cumsum_test"], y=data["Cumsum_bug"],
        line={"dash": "solid", "width": 5, "color": "blue"},
        name="テスト結果"
    ),
    go.Scatter(
        x=data["Cumsum_test"], y=data["Cumsum_bug"],
        mode="markers", marker={"size": 15, "symbol": "diamond"},
        name="",
    )
]
fig = go.Figure(chart)
fig.update_layout(
    title={
        "text": "<b>累積バグ数の予測</b>",
        "font": {
            "size": 22,
            "color": "grey"
```

```
        },
        "x": 0.5,
        "y": 0.9,
    },
    legend={
        "xanchor": "right",
        "yanchor": "bottom",
        "x": 0.95,
        "y": 0.05,
        "orientation": "h"
    },
    xaxis={
        "title": "テスト実施総数",
        "dtick": 100
    },
    yaxis={
        "title": "累積バグ数",
        "dtick": 2
    }
)
fig.update_layout(
    template="plotly_dark",
    autosize=False,
    width=800,
    height=500,
    margin={
        "l": 50,
        "r": 50,
        "t": 80,
        "b": 80,
        "pad": 4
    },
)
fig.show()
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$\boldsymbol{X}$に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。