Python#004★★☆☆ 2022.4.16/ 2024.1.23

薬物動態～静注反復 1-コンのあてはめ計算

薬物動態、静注反復投与 1-コンパートメントモデルのあてはめ計算（prog02_11.py）

＊前回までの記事を確認した上で下記を参照のこと。

(1) データの入力：投与量（DOSE)、薬物動態パラメータ（クリアランス（CL)、分布容積（VD))。反復投与は一定用量・一定間隔（TAU）とした。

(2) 反復投与では不連続モデルとなるため、パラメータ値推定には単回投与時に用いていた curve_fit 関数ではなく、残差平方和を計算してそれを最小化する minimize 関数を用いた。残差平方とは「実測値とモデルによる理論値の差の二乗をすべてのデータについて合計したもの（詳細説明略、プログラム内では SS)」で、関数 obj_func() に定義した。

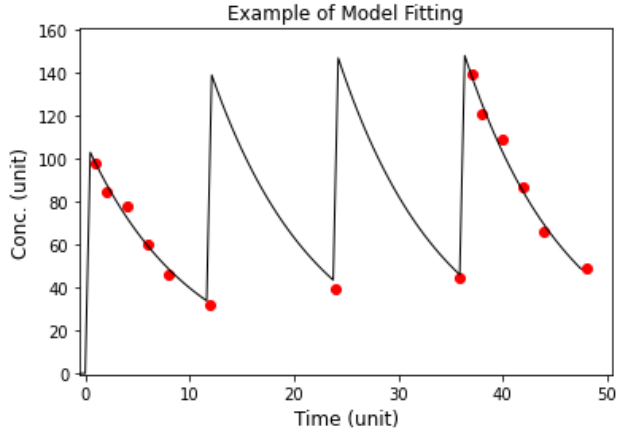
(3) 残差平方和を求める上でモデルによる理論値が必要になる。反復投与の投与回（tn）を計算した上でその時刻での理論値（Cp）を求める式を model_eq111() に定義した。

(4) 今後他の解析モデルも組み込むことなどを考慮して、薬物動態パラメータはリスト（P[]）とし、またグローバル変数とした。

(5) 最小二乗法の計算後、理論値をシミュレートする関数（draw_plot_pred()）を定義した。各軸の最小値、最大値を任意に設定できるようにするなど、工夫の余地がある。

(6) 残差平方和最小化のための minimize 関数では Nelder-Mead 法（薬物動態領域では Simplex 法として知られる）を用いた。

以上の内容で作成した例を次ページ「prog02_11.py」に示した。出力結果は次のとおり。




```
-- Estimated Parameters --
     CL = 0.0925
     Vd = 0.9263
     Ke = 0.0998
 * t1/2 = 6.9444
```

```
final_simplex: (array([[0.0924589 , 0.92631791],
       [0.0924652 , 0.92631394],
       [0.09245854, 0.92641289]]), array([153.65430315, 153.65430638, 153.65436891]))
           fun: 153.65430314933752
       message: 'Optimization terminated successfully.'
          nfev: 101
           nit: 53
        status: 0
       success: True
             x: array([0.0924589 , 0.92631791])
```

以上

```
"""# Program 02-11 "prog02_11.py"""
# Fitting to One-Compartment iv model after multiple dosing
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as opt

# Global varialbes
DOSE = 100.
TAU = 12.0 # Dosing Interval

Nparam = 2
P = [0 for j in range(Nparam)]
P[0] = 0.5
P[1] = 1.0
# Estimated CL, VD = 0.092 and 0.926
Npoint = 14
x = [0 for j in range(Npoint)]
y = [0 for j in range(Npoint)]

#
def set_data():
    # Data input in this program
    data_x = np.array([1.0, 2.0, 4.0, 6.0, 8.0, 11.9, 23.9, 35.9,
             37, 38, 40.0, 42.0, 44.0, 48.0])
    data_y = np.array([97.9, 84.3, 77.9, 60.4, 45.7, 31.7, 39.4, 44.4,
             139.6, 121.1, 109.2, 86.5, 65.8, 48.6])
    return data_x, data_y

def obj_func(P):
    SS = 0.0
    for i in range(Npoint):
        xx = x[i]
        Cp = model_eq111(xx, i, P)
        SS = SS + (y[i] - Cp) ** 2
    return SS

def model_eq111(xx, i, P):
    # 1-Comp. iv model
```

```
    tn = 0
    x2 = 0
    while xx > tn * TAU:
        tn = tn + 1
        x2 = xx - (tn - 1) * TAU
    CL = P[0]
    VD = P[1]
    ke = CL / VD
    E1 = 1 - np.exp(- ke * tn * TAU )
    E2 = 1 - np.exp(- ke * TAU )
    Cp = DOSE / VD * E1 / E2 * np.exp(- ke * x2)
    return Cp

def draw_plot_obs(data_x, data_y):
    # Plotting observed data
    plt.title("Example of Model Fitting", fontsize = 12)
    plt.xlim(-0.5, 50.5)
    plt.ylim(-0.5, 160.5)
    plt.xlabel("Time (unit)", fontsize = 12)
    plt.ylabel("Conc. (unit)", fontsize = 12)
    plt.plot(data_x, data_y, "o", color = "red")
    #plt.show()

def draw_plot_pred():
    # Simulation of predicted curve
    tx = [0 for j in range(100)]
    cy = [0 for j in range(100)]
    for i in range(100):
        tx[i] = (i - 1) * (48.0 - 0.0) / (100 - 1)
        cy[i] = model_eq111(tx[i], i, P)
    plt.plot(tx, cy, linestyle = "solid", linewidth = "1", color = "black")
    plt.show()

def print_output():
    # Output
    print ("-- Estimated Parameters --")
    print ('    CL = {:.4f}'.format(CL))
    print ('    Vd = {:.4f}'.format(VD))
```

```
    print ('    Ke = {:.4f}'.format(CL / VD))
    print (' * t1/2 = {:.4f}'.format(np.log(2) / (CL / VD)))


#
if __name__ == '__main__':
    x, y = set_data()
    draw_plot_obs(x, y)
    results = opt.minimize(obj_func, P, method='Nelder-Mead')
    print (results)
    P[0] = results.x[0]
    P[1] = results.x[1]
    CL = P[0]
    VD = P[1]
    draw_plot_pred()
    print_output()
```