薬物動態～点滴投与反復 1-コンのあてはめ計算

薬物動態、点滴反復投与 1-コンパートメントモデルのあてはめ計算（prog02_14.py）

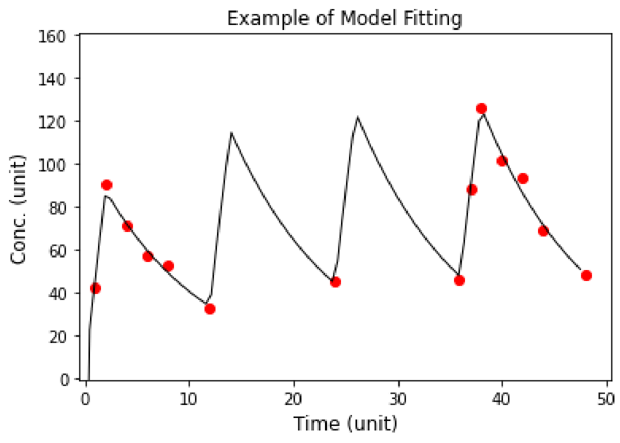
＊前回までの記事を確認した上で下記を参照のこと。

(1) 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))、点滴時間（TINF)。反復投与は一定用量・一定間隔（TAU）とした。

(2) minimize 関数を用いた。残差平方和を関数 obj_func() に、モデル式を model_eq141() に定義。

(3) 薬物動態パラメータはリスト（P[]）とし、グローバル変数とした。

以上の内容で作成した例を次ページ「prog02_14.py」に示した。出力結果は次のとおり。



```
-- Estimated Parameters --
     CL = 0.0998
     Vd = 1.0409
     Ke = 0.0959
 *  t1/2 = 7.2270
```

```
      message: Optimization terminated successfully.
      success: True
       status: 0
          fun: 106.38455065918137
            x: [ 9.984e-02  1.041e+00]
          nit: 21
         nfev: 43
final_simplex: (array([[ 9.984e-02,  1.041e+00],
                       [ 9.984e-02,  1.041e+00],
                       [ 9.983e-02,  1.041e+00]]), array([ 1.064e+02,
1.064e+02,  1.064e+02]))
```

以上

```python
"""# Program 02-14 "prog02_14.py"""
# Fitting to One-Compartment inf 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
TINF = 2
RATE = DOSE / TINF

Nparam = 2
P = [0 for j in range(Nparam)]
P[0] = 0.1 # CL
P[1] = 1.0 # Vd (Vd/F)

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,
             37.0, 38.0, 40.0, 42.0, 44.0, 47.98])
    data_y = np.array([42.4, 90.5, 71.3, 57.1, 52.4, 32.9, 45.2, 46.3,
             88.3, 126.2, 101.3, 93.3, 69.3, 48.2])
    return data_x, data_y

def obj_func(P):
    SS = 0.0
    for i in range(Npoint):
        xx = x[i]
        Cp = model_eq141(xx, i, P)
        SS = SS + (y[i] - Cp) ** 2
    return SS
```

```
def model_eq141(xx, i, P):
    # 1-Comp. inf 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
    E3 = 1 - np.exp(-Ke * TAU)
    E2 = 1 - np.exp(-Ke * (tn - 1) * TAU)
    if x2 < TINF: # during infusion
        E4 =     np.exp(-Ke * (TAU + x2 - TINF))
        E4 = E4 - np.exp(-Ke * (TAU + x2))
        Cp = (1 - np.exp(-Ke * x2)) + E2 / E3 * E4
        Cp = RATE / VD / Ke * Cp
    else: # after infusion
        E1 = 1 - np.exp(-Ke * tn * TAU)
        E4 = (1 - np.exp(-Ke * TINF)) * np.exp(-Ke * (x2 - TINF))
        Cp = RATE / VD / Ke * E1 / E3 * E4
    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_eq141(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()
```