Simulation of mean residence time of water in the modeled object

Daniel Stefanik

1. Explanation of aim

The aim of this exercise was to create model of physical phenomenon - Mean residence time of water in the modeled object. Implementation of computer simulation helps to understand the mechanisms describing the phenomenon and it theoretical concept. Another objective is to create box models:

- piston-flow model,
- exponential model,
- dispersion model.

Next step is to find the most fitting value of mean residence time parameter.

2. Used tools

Simulation was created in python language using Jupyter Notebook application. Libraries and their use:

- *pylab* - creating all diagrams,
- *math* - compute some mathematical expressions.

3. Results

a. Methodology

In the "black-box" model calculation of integral was discretized using rectangular method. Assuming that the input data has step 1 (1 month) the rectangular integration was simply sum of proper values.

b. Parameters

When using “black-box” model We have provided the input and output values and the aim is to calculate parameters. In various methods we can have different variables:

- in piston-flow model: *tt* (mean residence time);
- in exponential model: *tt* (mean residence time);
- in dispersion model: *tt* (mean residence time), *pe* (paclet number).

4. Mean residence time - qualitative

Searching for variables using observation:

- piston-flow model is the simplest method and the result is not correct. We can observe (on diagrams 3.1, 3.2, 3.3) that increasing *tt* leads to less accurate approximation of observed values (on diagrams blue line - real data, red line - model approximation);

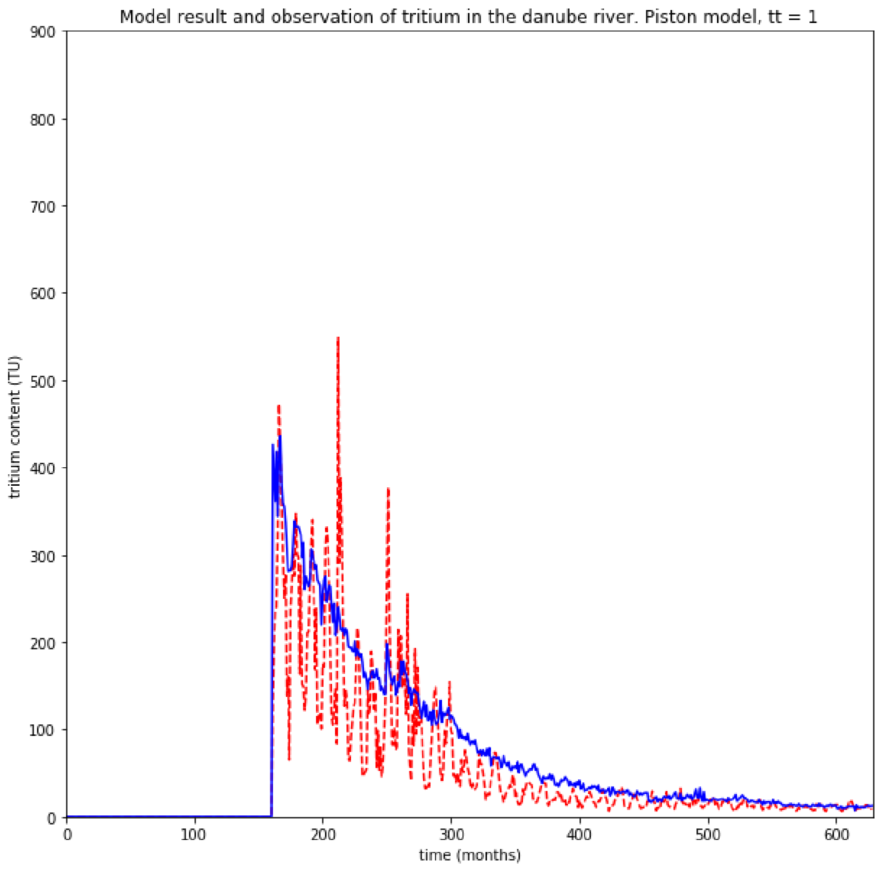

Diagram 3.1. Model result and observation of tritium in the danube river for piston-flow model and mean residence time parameter (*tt* = 1).

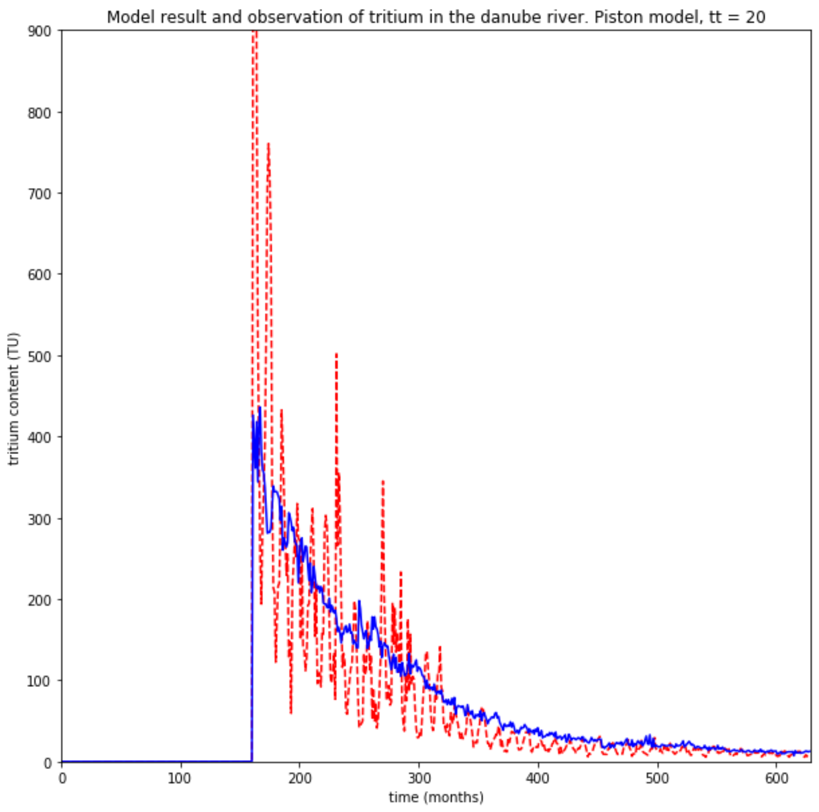

Diagram 3.2. Model result and observation of tritium in the danube river for piston-flow model and mean residence time parameter (*tt* = 20).

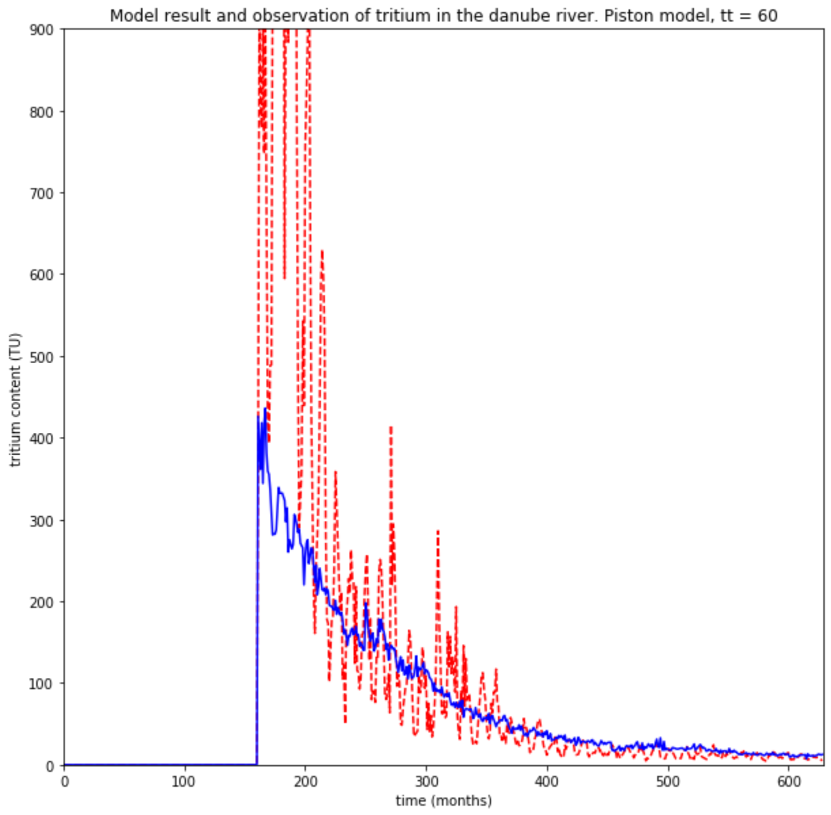

Diagram 3.3. Model result and observation of tritium in the danube river for piston-flow model and mean residence time parameter (*tt* = 60).

- exponential model reproduces quite well the observation of tritium in river. We can observe that the model result is most accurate for parameter *tt* equal around 60. Example runs we can see in diagrams (3.4, 3.5, 3.6);

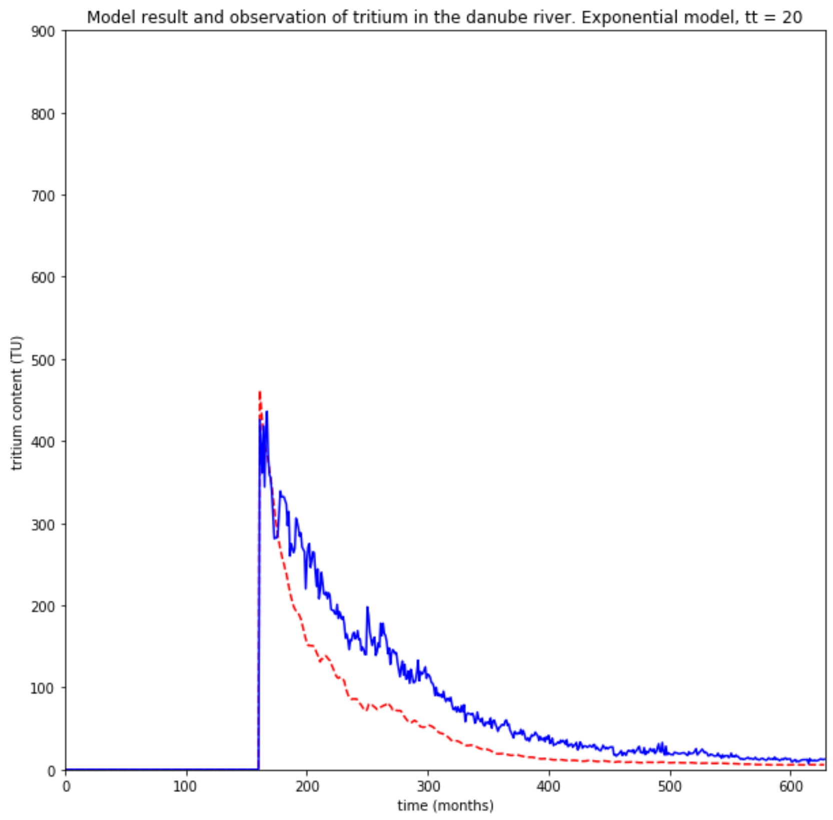

Diagram 3.4. Model result and observation of tritium in the danube river for exponential model and mean residence time parameter (*tt* = 20).

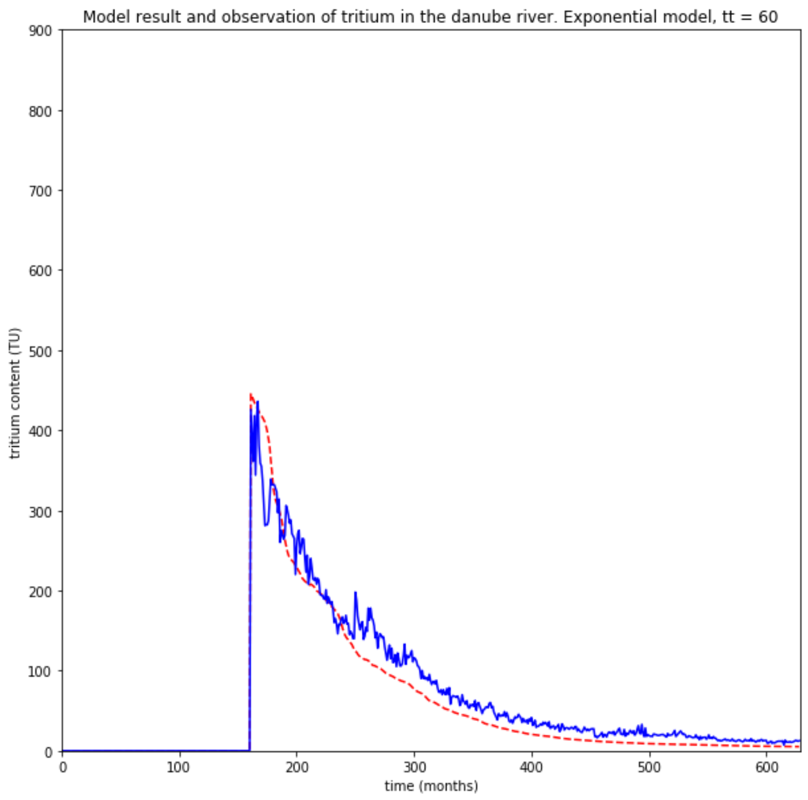

Diagram 3.5. Model result and observation of tritium in the danube river for exponential model and mean residence time parameter (*tt* = 60).

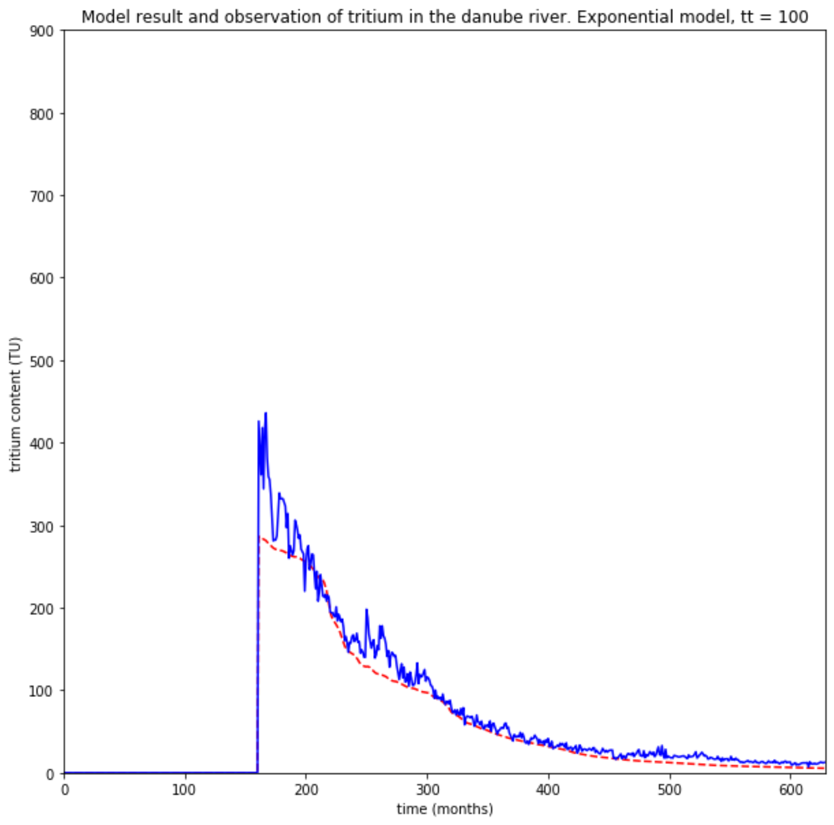

Diagram 3.6. Model result and observation of tritium in the danube river for exponential model and mean residence time parameter (*tt* = 100).

- dispersion model is the most complex model from those listed, but it give the most accurate result. We can observe that the model result is most accurate for parameter *tt* equal around 12 and the *pe* equal to 10.0. Example runs we can see in diagrams (3.7, 3.8);

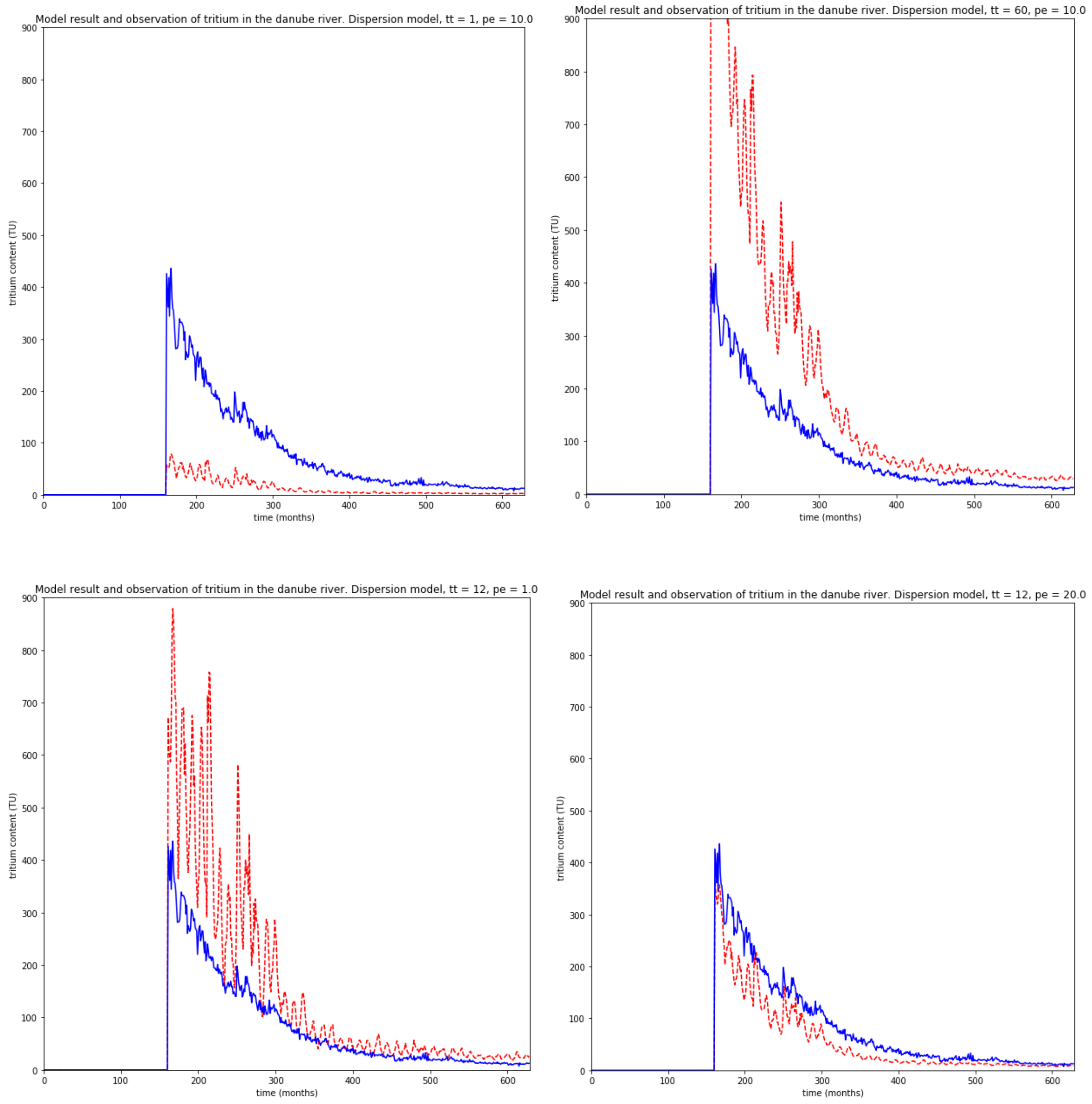

Diagram 3.7. Model result and observation of tritium in the danube river for dispersion model and parameters:

- top left: *tt* = 1, *pe* = 10.0;
- top right:: *tt* = 60, *pe* = 10.0;
- bottom left: *tt* = 12, *pe* = 1.0;
- bottom right: *tt* = 12, *pe* = 20.0;

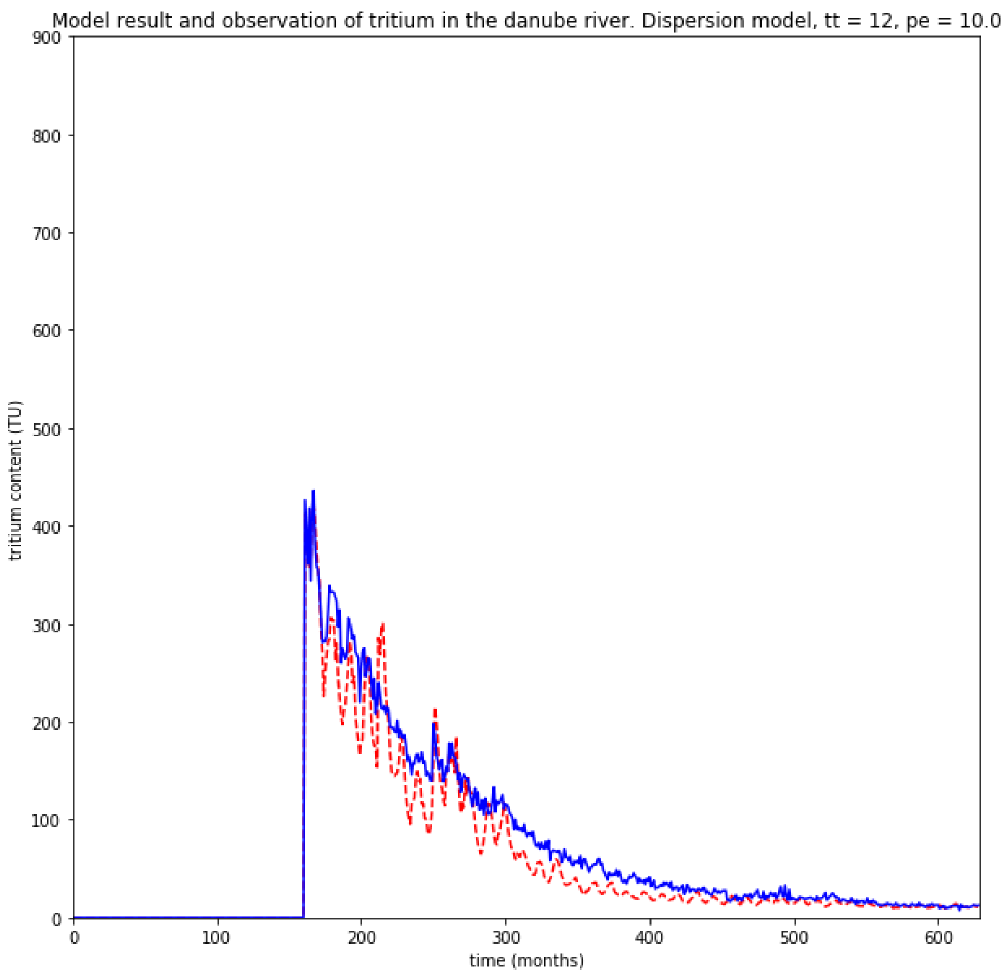

Diagram 3.8. Model result and observation of tritium in the danube river for dispersion model and parameters: *tt* = 12, *pe* = 10.0

5. Mean residence time - quantitative

In this stage the parameters *tt* and *pe* are computed automatically using error function (square error) between the observed output and computed values for all models:

- piston-flow - in this model the minimum of error function is for *tt* = 1. The error is equal to 1261712.63085 (See Diagram 3.9). Observed parameters:
    - *tt* in range 1 to 161;

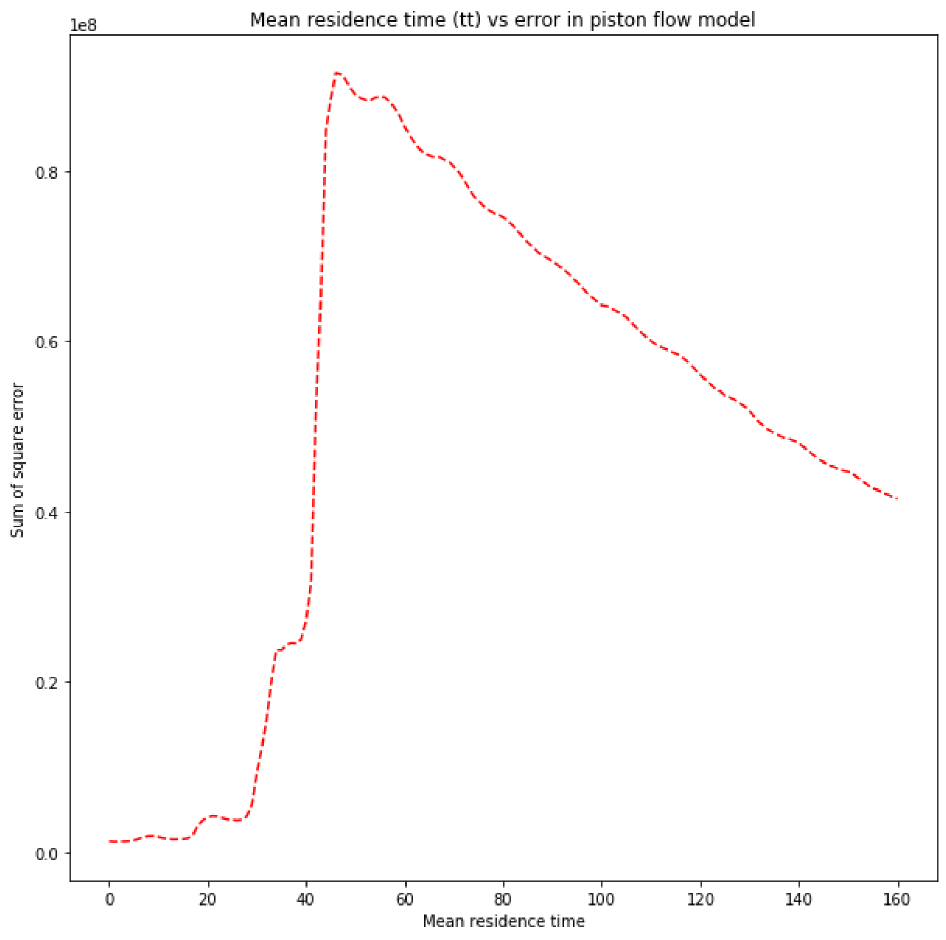

Diagram 3.9. Mean residence time (tt) vs error in piston-flow model

- exponential model - in this model the minimum of error function is for *tt* = 80. The error is equal to 154867.193487 (See Diagram 3.10). Observed parameters:
    - *tt* in range 1 to 800;

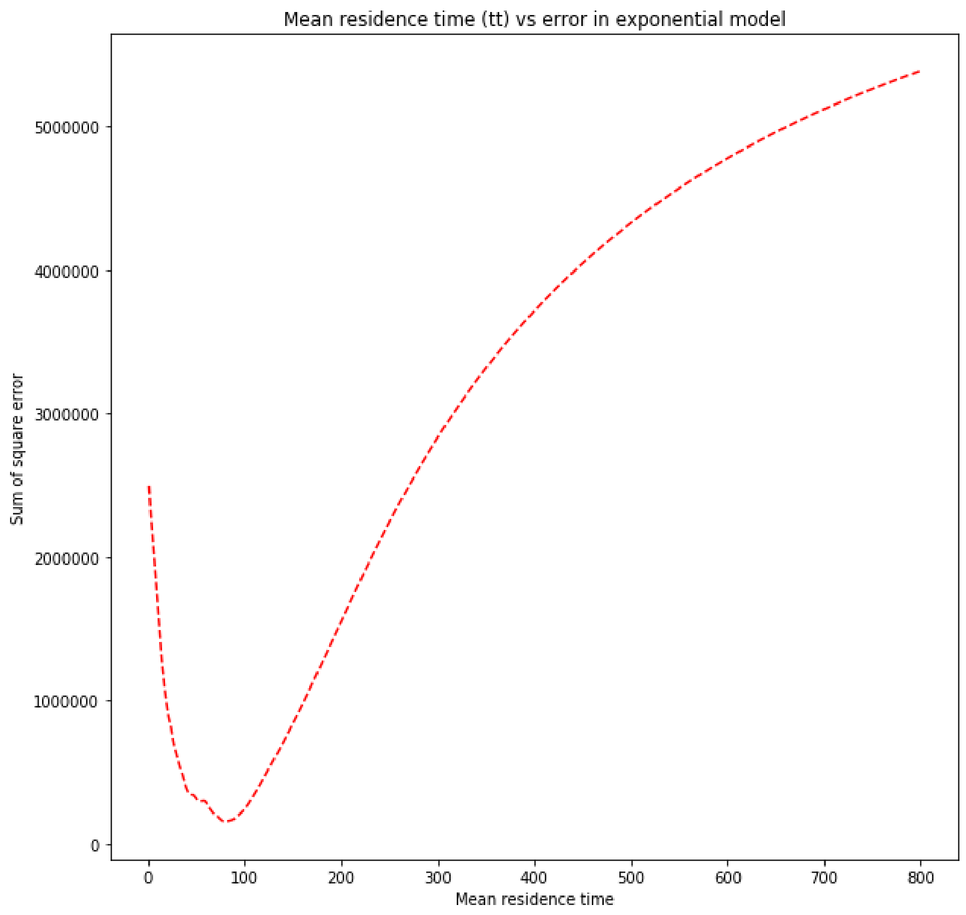

Diagram 3.10. Mean residence time (tt) vs error in exponential model

- dispersion model - in this model the minimum of error function is for *tt* = 14 and *pe* = 10. Error is equal to 219854.7434. In this method observation was divided into two steps to decrease time of computation:
    - Step I - observed parameters:
        - *tt* in range 1 to 100 with step 10;
        - *pe* in range 1 to 100 with step 10;
    - Step II - observed parameters:
        - *tt* in range 10 to 30 with step 1;
        - *pe* in range 10 to 30 with step 1.

6. Conclusions

Created model - mean residence time of water in object - show us various box models to calculate parameters using the “black-box” method. Searching of parameters: mean residence time and Peclet number leads us to conclusion which of models gives the most accurate results for modeled object. Comparing the errors:

piston-flow model:	1261712.63085,
exponential model:	154867.193487,
dispersion model:	219854.7434.

We can assume that the exponential model gives the most accurate results.

7. Code listening

```
% matplotlib inline
import pylab as pl

import math


# opady
Cin = [32.04048, 38.26488, 36.34968, 50.4264, 39.12672, 39.8928,
423.8904, 807.888, 706.3824, 869.1744, 308.9784, 236.2008, 141.3984,
76.2816, 44.6808, 53.2992, 38.9352, 35.1048, 46.596, 57.1296, 61.9176,
101.1792, 107.8824, 48.5112, 38.9352, 60.0024, 35.1048, 36.0624,
42.7656, 76.2816, 74.8452, 116.5008, 118.416, 209.388, 230.4552,
229.4976, 444.9576, 122.2464, 91.6032, 60.0024, 57.1296, 108.84,
146.1864, 106.9248, 101.1792, 167.2536, 148.1016, 186.4056, 155.7624,
137.568, 105.0096, 82.9848, 94.476, 245.7768, 276.42, 276.42, 386.544,
558.912, 827.04, 779.16, 1425.54, 329.088, 147.144, 239.0736, 587.64,
713.0856, 858.6408, 1206.2496, 966.8496, 633.6048, 688.188, 570.4032,
262.056, 134.6952, 109.7976, 115.5432, 107.8824, 118.416, 140.4408,
101.1792, 150.9744, 187.3632, 252.48, 292.6992, 191.1936, 140.4408,
112.6704, 63.8328, 91.6032, 79, 39.2, 43.2, 123, 174.5, 125.5, 140.5,
104, 61.8, 56, 101, 136, 487, 1232, 1078, 1032, 1210, 990, 1228, 877,
550, 521, 647, 648, 1265, 1900, 2900, 3035, 5930, 4960, 5950, 3010,
2200, 1558, 788, 1334, 1664, 1171, 2745, 2973, 2686, 2365, 2610, 1783,
1073, 649, 384, 432, 475, 719, 582, 1003, 1080, 1173, 1575, 941, 578,
426.5, 275, 213, 322, 368, 432, 578, 780, 835, 792, 703, 373, 235, 229,
134.3, 189, 236.8, 241, 323, 475, 423, 359, 299, 252, 280.2, 138.9,
161.8, 65.3, 237.1, 266.3, 301.7, 277.1, 348.6, 301.5, 299, 164.4,
288.8, 152.5, 150, 122.6, 140.1, 211.2, 213.1, 270, 315.4, 341.8, 266.9,
169.2, 239.6, 105.4, 119, 119.6, 101.2, 175.6, 172.2, 309.6, 332.6,
319.3, 266.2, 191.4, 114.2, 105.8, 124.1, 146.8, 84.3, 550.9, 292.1,
390.1, 333.3, 216.1, 127.3, 146.3, 133.8, 66.7, 64.2, 91.8, 108.95,
126.1, 130.8, 165.9, 217.4, 205.6, 149.2, 93.7, 45.8, 48.6, 51.65, 54.7,
146.4, 116.6, 125.3, 190.4, 170.7, 124, 144, 57.1, 99.2, 53.9, 80, 45.3,
58.2, 93.4, 138, 267.9, 379, 226, 166, 83.3, 82.8, 103, 76.4, 80.2, 215,
159, 208, 148, 174, 150, 117.9, 256.1, 77, 53.7, 41.3, 120, 82.4, 193,
95.2, 172, 105, 115, 112, 71.9, 44.5, 32.3, 32.8, 38.6, 34.3, 63.5,
93.4, 116, 147, 149, 123, 107, 70.5, 49.5, 42, 53.6, 99.2, 84.6, 118,
117, 155, 100, 95.2, 47.1, 47.9, 39.8, 50.7, 41.8, 34.8, 65.9, 53.8,
51.3, 76.5, 68.7, 56.8, 50.6, 38.3, 31.6, 38.1, 31, 31.9, 34.6, 44.2,
74.8, 60.4, 60.3, 50.3, 38.1, 19.8, 24, 26.8, 25.3, 42.1, 47.6, 63.5,
73.8, 73.2, 61.5, 50.8, 34.7, 24.5, 24.2, 15.8, 21.5, 27.2, 30.2, 32.8,
37, 34.1, 48.3, 42.7, 21.3, 27.7, 16.8, 13.8, 15.6, 13, 31.5, 25.6,
32.3, 40.8, 39.7, 38.6, 37.55, 36.5, 21.5, 16, 16.3, 16.7, 22.2, 28.6,
34.2, 39.8, 39.3, 38.8, 38.3, 17.2, 12.8, 18.1, 16.3, 18.8, 20.1, 19.1,
27.4, 28.5, 27.7, 21.3, 24.9, 16.4, 14.8, 18.7, 13.5, 17.3, 15.9, 23.9,
25.3, 32.4, 28.8, 24.4, 22.9, 11, 19.2, 13.6, 10.3, 14.8, 17.3, 21.1,
23.6, 22.7, 28.6, 25.5, 20.8, 16.4, 15.9, 11.2, 15.3, 17.9, 17.6, 21.7,
31.3, 25, 23.9, 21.2, 14.6, 6.5, 11, 10.1, 13, 14.3, 19.9, 24.3, 23.9,
28.9, 31.5, 34, 25.5, 13, 10, 9.9, 9.9, 9.8, 17.2, 17.3, 25.9, 22.1,
25.4, 20.3, 17.3, 12.9, 9.7, 10.4, 11.1, 10.6, 10.2, 17.9, 17.5, 23.1,
23.8, 29.3, 18.6, 15.1, 7.2, 10.5, 10.6, 8.9, 22.2, 17.4, 17, 19.2,
28.6, 14.1, 12, 11.3, 10.1, 8.9, 9, 14.5, 17.5, 17.1, 24.7, 32.3, 18.1,
15.1, 15.7, 14, 16.7, 13.1, 11.3, 20.8, 19.4, 13.5, 18.5, 15, 24.7,
18.7, 14, 12.1, 10.1, 11.7, 13, 11.5, 13.8, 15.59, 20.21, 18.88, 22.26,
19.19, 12.71, 16.48, 13.16, 8.92, 8.8, 14.45, 18.27, 15.79, 15.85,
19.62, 15.59, 17.94, 17.64, 10.41, 8.27, 7.99, 6.02, 11.38, 10.86,
17.53, 14.3, 15.05, 20.4, 20.41, 18.34, 11.54, 13.61, 11.15, 12.25,
12.67, 10.24, 15.59, 17.63, 16.01, 18.91, 19.49, 11.64, 7.89, 9.66,
8.22, 9.58, 10.94, 9.61, 13.47, 13.64, 17.32, 17.13, 12.9, 10.69, 9.63,
10.55, 7.98, 8.71, 9.47, 9.85, 10.6, 14.67, 11.85, 11.62, 12.11, 11.9,
9.58, 8.075, 6.57, 8.79, 10.92, 8.92, 9.51, 12.68, 11.51, 11.18, 11.94,
10.93, 12.82, 9.08, 9.09, 7.06, 7.65, 7.73, 14.44, 11.88, 12.2, 14.3,
13.49, 11.64, 7.75, 6.07, 8.73, 9.26, 10.22, 11.09, 12.07, 12.55, 13.53,
13.69, 13.85, 10.89, 8.39, 7.2, 6.01, 9.24, 6.87, 10.06, 10.4, 16.09,
14.23, 17.79, 15.09, 8.77, 7.32, 8.61, 9.9, 10, 10, 10.05, 8.6, 11.38,
13.48, 13.37, 12.64, 9.63, 10.82, 8.44, 7.04]

# dunaj
COut = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 426, 395,
361, 418, 344, 417, 436, 381, 359, 355, 337, 310, 281, 282, 282, 287,
313, 339, 332, 332, 332, 327.5, 323, 297, 314, 260, 275, 269.5, 264,
269, 306, 301, 294, 284, 288, 271, 268, 265, 220, 254, 271, 275, 246,
254, 265, 264, 241, 223, 244, 208, 218, 240, 228, 214, 213, 216, 208,
215, 212, 195, 194, 194, 190, 189, 201, 184, 192, 188, 183, 186, 176,
160, 165, 158, 146, 158, 157, 165, 167, 159, 161, 169, 158, 159, 145,
149, 146, 140, 140, 198, 188, 168, 160, 151, 158, 161, 139, 143, 154,
149, 178, 163, 178, 166, 163, 156, 141, 148, 128, 141, 146, 144, 141,
142, 130, 122, 113, 124, 132, 115, 128, 110, 111, 120, 105, 122, 115,
106, 106, 109, 119, 133, 108, 119, 116, 117, 121, 125, 111, 116, 115,
```

```python
112, 106, 105, 101, 90.2, 100, 90.1, 92.5, 89.3, 91, 87.8, 95.3, 87.6,
82.7, 86.1, 86.7, 83.3, 87.8, 79.1, 73.3, 73, 76.1, 70.1, 76.2, 71.7,
70.1, 77.8, 70.5, 78.8, 58.2, 66.4, 68.7, 68.2, 67.7, 66.5, 68.4, 64.9,
56.5, 59.3, 69.9, 62, 60.8, 58, 62.5, 54.5, 53, 57.7, 56.2, 58.1, 55.3,
62.7, 50.5, 58.5, 60.1, 55.4, 51.6, 46.5, 51, 51.7, 54.5, 54.5, 53.8,
59, 60.3, 53.7, 55.5, 48.2, 45.3, 42, 38.6, 46.3, 44.5, 43.1, 44.6,
43.7, 48.8, 42.5, 47.4, 42.9, 36.8, 35.9, 38.6, 34.5, 38.45, 42.4, 41.1,
41.9, 45.5, 42.9, 41, 35.3, 40.3, 36.5, 36.2, 38.5, 33.9, 39.7, 37.1,
41.5, 31.8, 33.4, 38.1, 29.3, 30.7, 30.7, 32.1, 33.7, 31.9, 35.7, 33.3,
35.5, 33.3, 30.7, 36.2, 28, 29.1, 30.7, 31.1, 27.1, 29.7, 29.8, 32.9,
24, 26.4, 33.5, 31, 26.1, 28.4, 28.8, 28.3, 26.9, 28.15, 29.4, 28.2,
27.2, 30.2, 30.3, 24.7, 27.4, 25.1, 23.6, 22.9, 25.6, 29.5, 27.1, 25,
27.1, 25.1, 26.5, 27.5, 27.2, 27.6, 17.9, 18.5, 16.3, 18.2, 18.8, 20.85,
22.9, 17.2, 20.6, 19.9, 17.3, 21.2, 23.4, 22.2, 22.9, 21.3, 24.1, 19.7,
22.5, 23.3, 26.8, 28.1, 21, 20.7, 19.9, 26.6, 18.3, 22.3, 20.2, 21.8,
25.6, 23.8, 21.8, 25, 19.6, 21.6, 26.45, 31.3, 22.1, 21.3, 32.9, 21.6,
17.7, 28.4, 18, 20.2, 19.25, 18.3, 18.3, 17.9, 21, 20.5, 20.2, 19.5,
19.2, 19.4, 19.95, 20.225, 20.5, 19.3, 18.25, 17.2, 21.6, 19.3, 19.55,
19.8, 20.3, 22.85, 25.4, 17.9, 19.55, 21.2, 23, 24.8, 22.55, 20.3, 21,
19.8, 16.7, 17, 18.7, 18.2, 17.35, 16.5, 16.5, 16.7, 19.8, 16.5, 17.2,
13.8, 17.8, 15, 16.7, 16.85, 17, 14.5, 15.9, 19.8, 15.6, 16.2, 15.4,
17.7, 15.7, 15.7, 13.7, 12.6, 13.4, 12.3, 12.6, 13.3, 13.9, 14.7, 13,
13.8, 13.1, 12.5, 12.9, 14.8, 12.6, 12.6, 11.6, 12.4, 13.3, 13.7, 12.5,
11.9, 11.3, 12.4, 14.3, 12.7, 12.6, 12.9, 12.2, 15.1, 12.2, 12.9, 12.3,
10.9, 11.95, 13, 11.9, 13.4, 14.1, 12.3, 12.8, 12.85, 12.9, 9.4, 9.9,
11, 12.1, 9.4, 10.6, 9.7, 10.5, 11.6, 11.85, 12.1, 11.8, 12.1, 12.2,
7.2, 13.3, 11, 10.9, 10.7, 11.5, 10.5, 10.85, 11.2, 12.1, 13, 12.5, 12,
12.6]

amount_of_zeros = 161

lamb = math.log(2) / (12.3 * 12)

# len(Cin) =629
size = len(Cin)


# Box models


def piston_flow_model(t, tt):
   if t - tt <= 0:
       print("t: " + str(t) + " tt: " + str(tt) + " c[" + str(t - tt) +
"]= " + str(Cin[t - tt]))
   return Cin[t - tt] * math.exp(-lamb * tt)
```

```python
def exponential_model(t, tt):
    c_sum = 0.0
    for i in range(t):
        gt = math.exp(-(t - i) / tt) / tt
        c_sum += Cin[i] * gt * math.exp(-lamb * (t - i))
    return c_sum


# def dispersion_model(t, tt, pe):
#     c_sum = 0.0
#     for i in range(t):
#         p1 = math.pow(4 * math.pi * pe(t - i) / tt, -0.5)
#         p2 = 1 / (t - i)
#         p3a = -pow((1 - (t - i)) / tt, 2)
#         p3b = (4 * pe(t - i) / tt)
#         p3 = math.exp(p3a / p3b)
#         gt = p1 * p2 * p3
#         c_sum += Cin[i] * gt * math.exp(-lamb * (t - i))
#     return c_sum


def dispersion_model(t, tt, pe):
    c_sum = 0.0
    for i in range(t):
        p1 = math.pow(4.0 * math.pi * pe / tt, -0.5)
        p2 = 1.0 / (t - i)
        p3a = -pow((1.0 - (t - i)) / tt, 2)
        p3b = (4.0 * pe / tt)
        p3 = math.exp(p3a / p3b)
        gt = p1 * p2 * p3
        c_sum += Cin[i] * gt * math.exp(-lamb * (t - i))
    return c_sum


def do_plot(result, c_out):
    pl.figure(figsize=(10, 10))
    pl.xlim(0, len(Cin))
    pl.ylim(0, 900)
    pl.plot(result, 'r--')
    pl.plot(c_out, 'b-')

# Result show methods
```

```
def show_piston_flow_model_result(tt):
    results = []
    for iterator in range(amount_of_zeros, len(Cin)):
        results.append(piston_flow_model(iterator, tt))
    do_plot(([0] * amount_of_zeros) + results, COut)


def show_exponential_model_result(tt):
    results = []
    for iterator in range(amount_of_zeros, len(Cin)):
        results.append(exponential_model(iterator, tt))
    do_plot(([0] * amount_of_zeros) + results, COut)


def show_dispersion_model_result(tt, pe):
    results = []
    for iterator in range(amount_of_zeros, len(Cin)):
        results.append(dispersion_model(iterator, tt, pe))
    do_plot(([0] * amount_of_zeros) + results, COut)


def calc_error(ov, cv):
    return pow(ov - cv, 2)
#     return abs(ov - cv)

# Find minimum error methods


def show_piston_flow_model_error():
    errors = []
    tt_rang = range(0, amount_of_zeros)
    min_val = float("inf")
    o_tt = tt_rang[0]
    for i_tt in tt_rang:
        er = 0.0
        for iterator in range(amount_of_zeros, len(Cin)):
            r = piston_flow_model(iterator, i_tt)
            # print("i: " + str(iterator) + " COut: " +
str(COut[iterator]) + " r: " + str(r)
            #       + " e: " + str(calc_error(COut[iterator], r)) + " er:
" + str(er))
            er += calc_error(COut[iterator], r)
        if er < min_val:
            min_val = er
            o_tt = i_tt
        errors.append(er)

    print("Minimal error: " + str(min_val) + ", optimal tt: " +
 str(o_tt))

    pl.figure(figsize=(10, 10))
    pl.plot(tt_rang, errors, 'r--')
    pl.xlabel('Mean residence time')
    pl.ylabel('Sum of square error')
    pl.title('Mean resident time (tt) vs error in piston flow model')


 def show_exponential_model_error():
    errors = []
    tt_rang = range(1, 800)
    min_val = float("inf")
    o_tt = tt_rang[0]
    for i_tt in tt_rang:
        er = 0.0
        for iterator in range(amount_of_zeros, len(Cin)):
            r = exponential_model(iterator, i_tt)
            er += calc_error(COut[iterator], r)
        if er < min_val:
            min_val = er
            o_tt = i_tt
        errors.append(er)

    print("Minimal error: " + str(min_val) + ", optimal tt: " +
 str(o_tt))

    pl.figure(figsize=(10, 10))
    pl.plot(tt_rang, errors, 'r--')
    pl.xlabel('Mean residence time')
    pl.ylabel('Sum of square error')
    pl.title('Mean resident time (tt) vs error in exponential model')


 def show_dispersion_model_error():
    # First Step
    #     tt_rang = range(1,100,10)
    #     pe_rang = range(1,100,10)

    # Second Step
    tt_rang = range(10, 30, 1)
```

```python
    pe_rang = range(10, 30, 1)

    errors = [[0] * len(pe_rang)] * len(tt_rang)
    min_val = float("inf")
    o_tt = tt_rang[0]
    o_pe = pe_rang[0]
    i = 0
    for i_tt in tt_rang:
        j = 0
        for i_pe in pe_rang:
            print("i: " + str(i) + " j: " + str(j) + " i_tt: " +
str(i_tt) + " i_pe: " + str(i_pe))
            er = 0.0
            for iterator in range(amount_of_zeros, len(Cin)):
                r = dispersion_model(iterator, i_tt, i_pe)
                er += calc_error(COut[iterator], r)
            if er < min_val:
                min_val = er
                o_tt = i_tt
                o_pe = i_pe
            errors[i][j] = er
            j += 1
        i += 1
    print("Minimal error: " + str(min_val) + ", optimal tt: " + str(o_tt)
+ ", optimal pe: " + str(o_pe))

# for tt in [1,10,20,50,60]:
#     show_piston_flow_model_result(tt)
# for tt in [1,10,20,50,60]:
# show_exponential_model_result(tt)
# for tt in [1,10,12,20,50,60]:
#     for pe in [1,10,20,50,60]:
# show_dispersion_model_result(tt, pe)


# show_piston_flow_model_error()
# show_exponential_model_error()
show_dispersion_model_error()
```