LAB REPORT: LAB 7
Application of box models to Upper Danube catchment simulation

MODELING OF PHYSICAL SYSTEMS

Patryk Gaczyski

Sunday 6th May, 2018

1 Aim of laboratory

Main goal of this laboratory was to estimate mean residence time of water in Danube river with black-box model using exponential transit function.

2 Simulation description

As laboratory aim states, it is required to estimate mean residence time of water in Danube river. To achieve that, black box model is being used - which means, system won't be controlled directly, but only by adjusting its input parameters according to obtained output, without knowledge about its internals.

To model such phenomena, calculation of concentration of the tracer - Tritium 3H in this particular case - can be utilized. In this case, mathematical formula describing that model looks as follows:

$$C(t) = \int_{-\infty}^{t} C_{in}(t') \cdot g(t-t') \cdot e^{-\lambda \cdot (t-t')} dt' \tag{1}$$

Where:

- t - time variable
- $C(t)$ - output function
- $C_{in}(t')$ - input function
- $g(t-t') = \frac{e^{\frac{-(t-t')}{t_t}}}{t_t}$ - exponential time transit distribution function
- t_t - mean residence time
- $\lambda = 4.696 * 10^{-3}$ - radioactive decay constant

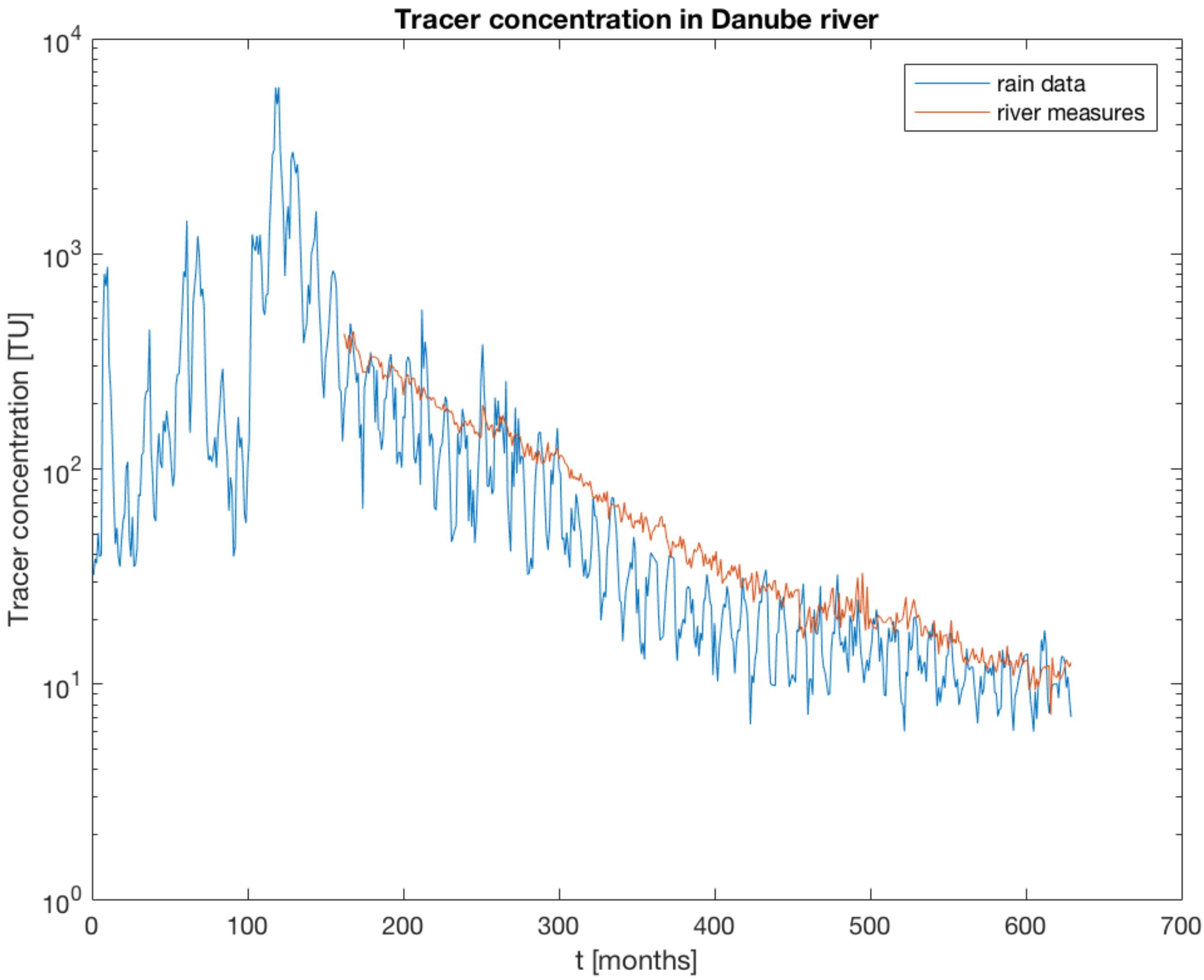
To evaluate such model, two data sets were provided:

1. `opady.prn` - file that contains input function values
2. `dunaj.prn` - file with calculated amount of tracer in Danube river in Vienna station (output to compare output to)

3 Implementation and results

3.1 Input and output data analysis

Plotting provided datasets, gives the following chart:



It is worth noticing that first 160 river measures are missing, however, they could be extrapolated from rain data. Other fact worth mentioning is that, it is clearly visible (not so obvious at the first glance), that river data is similar and slightly shifted to the right comparing with rain measures. This shift between input and output data can be used for mean residence time calculation.

3.2 Convolution integral

To calculate previously mentioned convolution integral with exponential time transit distribution function, the following matlab function were developed:

Listing 1: Convolutional integral implementation with exponential transit function

```
function sumsum = easy_integral(c_in, i, dt, tt, lambda)
    sum = 0;
    t = i * dt;
    for j = 1:i-1
        tp = j*dt;
        sum = sum + c_in(j) * ...
                    tt^(-1) * ...
                    exp(-1 * (t - tp) / tt) * ...
                    exp(-1 * lambda * (t - tp));
    end;
    sumsum = sum * dt;
end
```

It takes as parameters:

1. c_in - input function values vector;
2. i - timestamp for which to calculate;
3. dt - timestep;
4. tt - mean residence time;
5. lambda - radioactive decay constant.

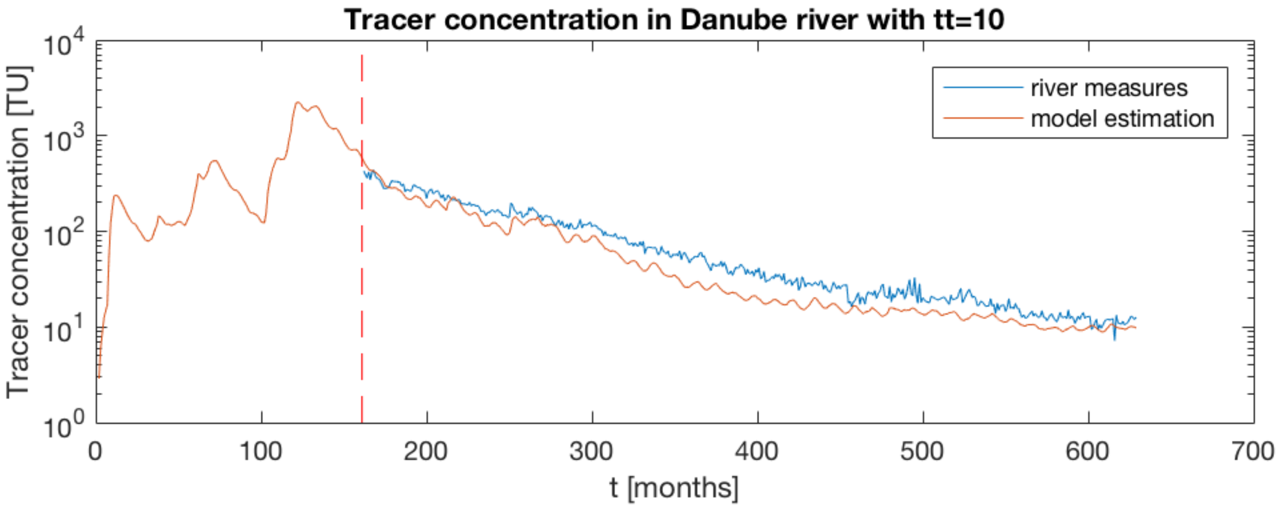
And returns calculated sum. Therefore, this function was run multiple times in the for loop with the following parameters:

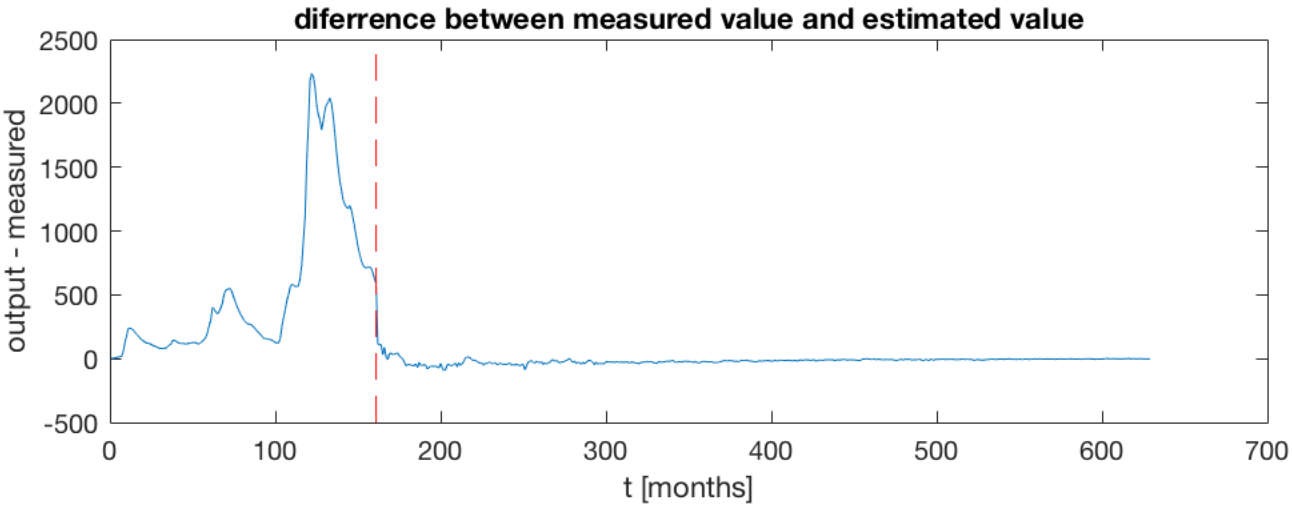
Listing 2: Convolutional integral implementation with exponential transit function

```
    rain = importdata('opady.prn');
    dunaj_river = importdata('dunaj.prn');
    num = rain(end, 1);
    tt = 10;                    % month
    lambda = 4.696e-3;          % 1/month?
    dt = rain(2,1) - rain(1,1); % month
    output = zeros(num,1);      % vector with output values


```

```
    for i= 1:num
        output(i) = easy_integral(rain(:,2), i, dt, tt,
           lambda);
    end

    figure;
    semilogy(dunaj_river(:,2));
    hold on;
    semilogy(output1);
```

With presented result:





One can conclude, that model approximates real values quite well, looking at difference between model output values and real datapoints. However, it still can be done better - tt value in this case, was guessed.

3.3 Calculating possible tt values

One way, to obtain tt value which suits well this model, is to check the next values occurring in the sequence while error value decrees. Example script performing such action is available below:

Listing 3: Convolutional integral implementation with exponential transit function

```

rain = importdata('opady.prn');
dunaj_river = importdata('dunaj.prn');
lambda = 4.696e-3;            % 1/month?
dt = rain(2,1) - rain(1,1); % month
mean_residence = [];
old_rmse = 0;

for tt = 1:1000
    num = rain(end, 1);

    output = zeros(num,1);       % vector with output
       values
    for i= 1:num
        output(i) = easy_integral(rain(:,2), i, dt, tt,
           lambda);
    end

    % count error, excluding values that does not make
       sense
    errors = (dunaj_river(161:num,2) - output(161:num));
    errors = errors.^2;

    % root mean square error
    rmse = sqrt(sum(errors)/num);
    if tt == 1
        old_rmse = rmse;
    else
        if rmse > old_rmse
            disp(tt);
            break;
        else
```

```
            old_rmse = rmse;
        end
    end
end
```

This method resulted in conclusion that tt = 8 is most suitable. Which sounds reasonable.

More sophisticated method to check for good tt value is to minimize rooted square error of output and measured values. Achieving that is simple as:

Listing 4: Convolutional integral implementation with exponential transit function

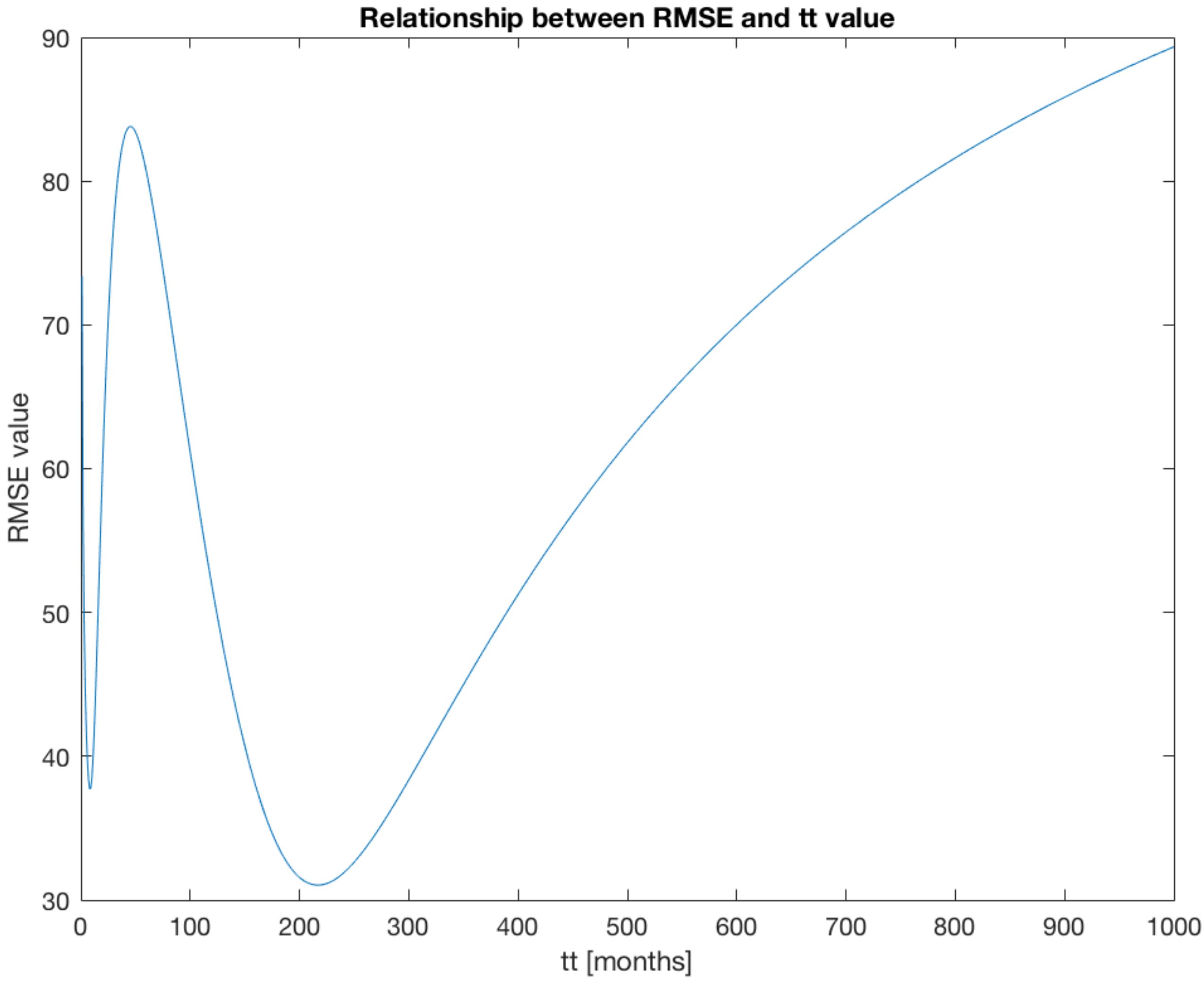
```

rain = importdata('opady.prn');
dunaj_river = importdata('dunaj.prn');
lambda = 4.696e-3;          % 1/month?
dt = rain(2,1) - rain(1,1); % month
mean_residence = [];

for tt = 1:1000
    num = rain(end, 1);

    output = zeros(num,1);      % vector with output values
    for i= 1:num
        output(i) = easy_integral(rain(:,2), i, dt, tt, lambda);
    end

    % count error, excluding values that does not make sense
    errors = (dunaj_river(161:num,2) - output(161:num));
    errors = errors.^2;

    % root mean square error
    rmse = sqrt(sum(errors)/num);
    mean_residence(tt,1) = tt;
    mean_residence(tt,2) = rmse;
end

% find tt with minimal RMSE
[val, idx] = min(mean_residence);
disp(val);
disp(idx);
```

resulting in:

Listing 5: Convolutional integral implementation with exponential transit function

```
>> river_flow
            1.00           31.04

            1.00          217.00
```

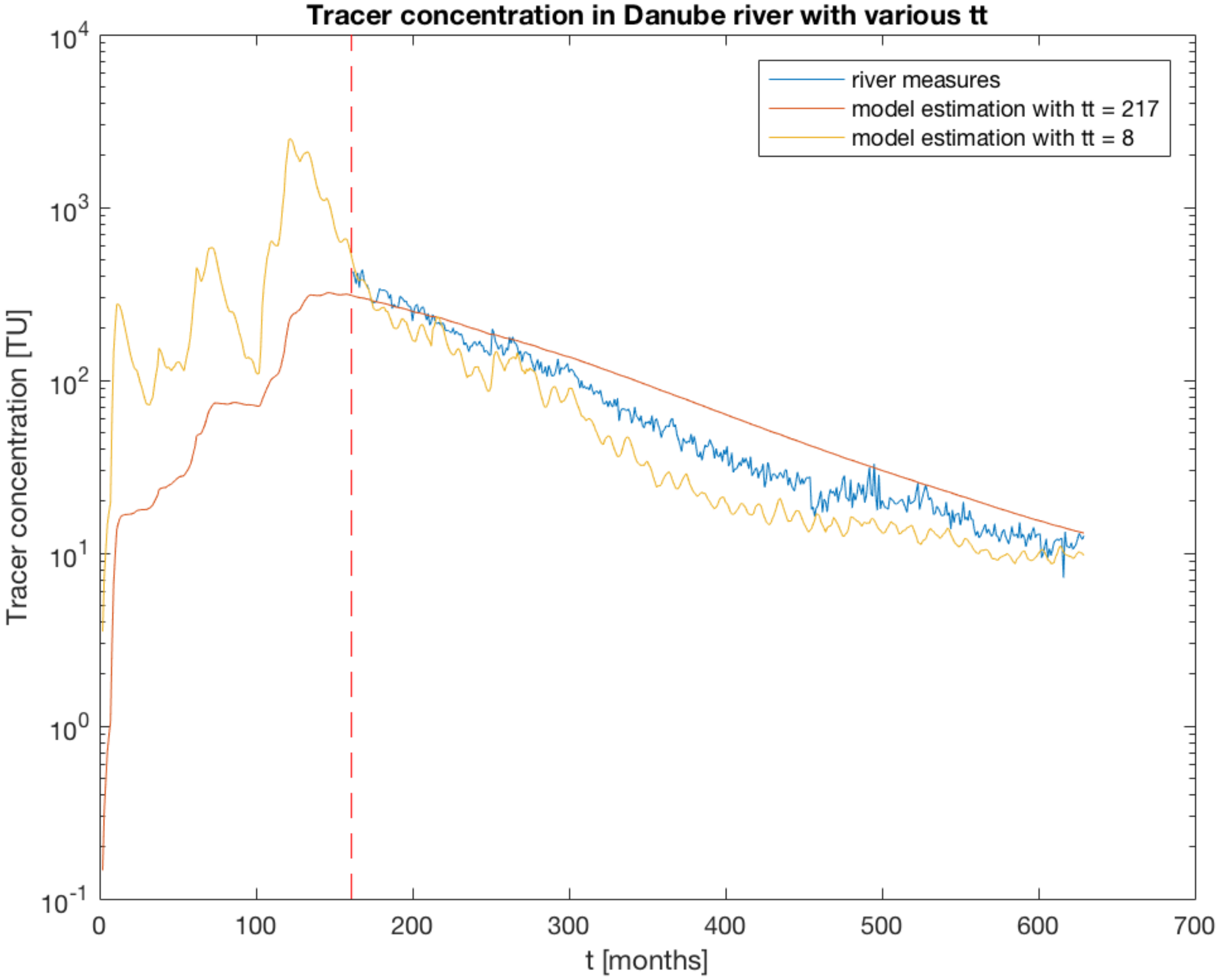
Which looks ridiculous, nonetheless, this is tt which gives the smallest RMSE. But, if one would take a look into chart presenting function of RMSE depending on tt value, then it is clearly visible that in fact 217 is global minimum of this function, however local minimum in $tt = 8$ is more likely to be true.



Listing 6: Convolutional integral implementation with exponential transit function

```
% check for local minimum at the beginning
>> min(mean_residence(1:100,:))
                1.00              37.73

                1.00               8.00
```

To check, whether these tt values makes sense, another chart with measured values compared to model values with tt = 217 and tt = 8 were developed:



3.4 Results comparison

Both methods resulted in similar results stating that mean residence time is equal to 8 months. However, trial error method seems to be more primitive and vulnerable for local minima, which in this case turned out to be helpful, but inverse modeling (minimizing rmse in this case) looked more promising, and actually proved first result, but not directly, because it found global minimum which does not make much sense in this case (or is it?)

4 Conclusion

This report covered simplified methods to calculate the mean residence time of water in the upper part of the Danube river using black box modeling technique with exponential transit function. Results obtained by two different methods agree with each other, and looks reasonable from author perspective. Output values produced by generated model fits not-so-well-but-acceptable provided example.