Laboratory exercise no. 5: Application of box models to Upper Danube catchment simulation

Piotr Gawryś < *pgawrys2@gmail.com* >

May 8, 2018

1 Introduction

The goal of this laboratory is to implement simulation capable of calculating a mean residence time of water in the Upper Danube. In the simulation we will use exponential transit time distribution function, practice calculating integrals programmatically, learn about box models and explore two methods of finding optimal parameters.

1.1 Mathematical Model

To calculate a mean residence time of water we are going to use mathematical model shown in the equation below:

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

Where:

- $C(t)$ – output function.
- $C_{in}(t')$ – input function.
- $g(t - t')$ – transit time distribution function.
- $\lambda = 4.696 * 10^{-3}$ – radioactive decay constant (tritium).
- t – time variable.

- t' – integration parameter.

One more important thing to note is that model used in this exercise is considered as *box model*. This means that the modelled object is treated as *black-box* – we reason about it solely on the basis to its response to given inputs. Concentration of the tracer in the precipitation is used as an input and the concentration of the tracer is the output.

1.2 Data for modelling

The input data is delivered in "opady.prn" file which contains mean amount of tritium in the precipitations during given time. There is also second file – "dunaj.prn" containing real measurements which will be used to validate our model. Those are illustrated on Figure 1.

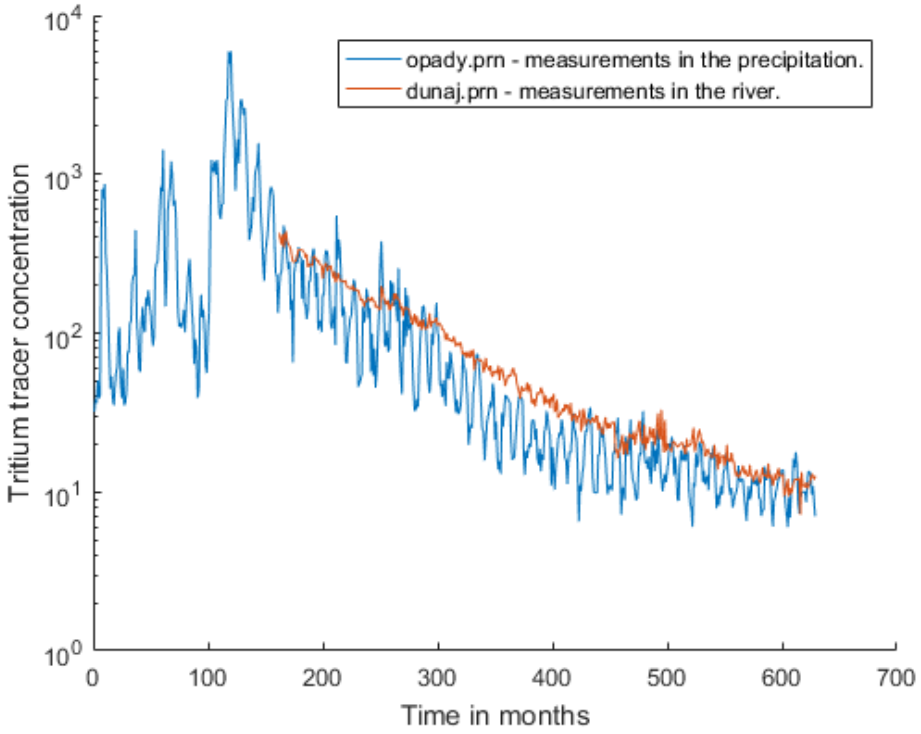

Figure 1: Measurements of tritium concentration in the precipitation and in Danube river.

2 Simulation

2.1 Calculating convolution integral

First step to implement discussed model is to calculate the convolution integral for selected transit time distribution function. During this laboratory we decided to choose **exponential model** which is expressed as follows:

$$g(t-t') = t^{-1}{}_t \cdot exp(-(t-t')/t_t) \tag{2}$$

Simple way to actually write it in Matlab is substituting $g(t-t')$ in the equation 1 and executing it in loop to reflect integral. Function representing this equation can be found at the end of report in subsection 4.2.

One of the variables **tt** (to denote t_t) is mean residence time. It is taken as a parameter to function but it's missing. In the next sections we will explore two methods to acquire this value and compare them.

2.2 Mean residence time using trial-error method

One of the methods to derive **tt** is called **trial-error method**. The way it works is that we calculate Root Mean Squared Error and take successive **tt** values while error is decreasing, otherwise we stay with last value. To calculate Root Mean Squared Error we need to have some results to compare and those are fortunately available in "dunaj.prn" file. The implementation is provided in section 4.3.

In this model I chose to check possible values from $t_t = 1$ and it resulted in $t_t = 8$ using step = 1. After tuning it to 0.001 it took significantly more time (minutes instead of seconds) but allowed to find more precise value of mean residence time which was **7.241**.

While this approach works for this use case there is a danger with this method - if we were to choose different range of values we could get different answer. This is easy to see on the figure 2 where we can see that the result would be much different for higher initial t_t. For instance if we were to choose range from 150 to 300 the minimal mean residence would be something around 210.

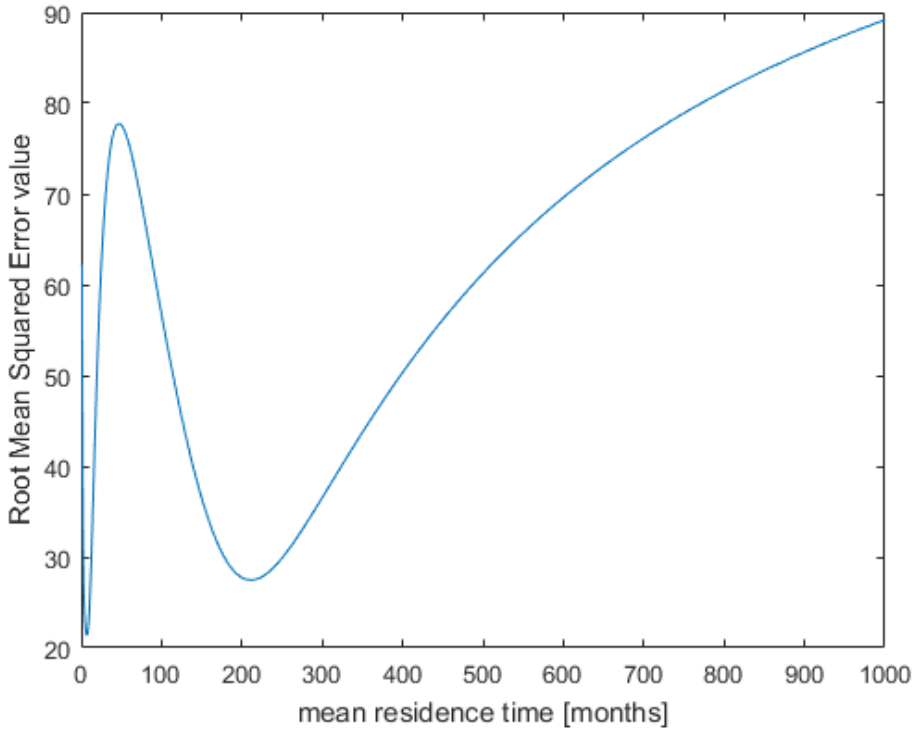

Figure 2: Relation of Root Mean Squared Error to different t_t values

Furthermore, this implementation is not ideal because it could go wrong for small oscillations in Root Mean Squared Error missing proper minimums.

2.3 Mean residence time using objective function

There is other approach which is included in the report. Instead of searching for optimal t_t value looping through results manually we can use MATLAB's **fminunc** function which is going to find the value of the local minima manually. Code can be found in the subsection 4.4. There are two snippets – **fminunc** requires taking function as a parameter and MATLAB version which was used to do this report doesn't allow having functions in script file.

The result was **7.2401** and the execution time was much faster than in case of manual approach as presented in previous section.

2.4 Comparison

Both trial-error method and using MATLAB's built in function provided similar results, 7.241 and 7.2401 respectively. Trial-error method could probably provide the same result with even lower time step but it would approach tens of minutes of execution time in comparison to seconds for *fminunc*. In case of this simulation the former time could be feasible but it is easy to imagine use case with much more variables leading to serious performance problems.

What's more it is probably better to use proven built in functions instead of implementing everything yourself which can lead to errors and sub-optimal solutions.

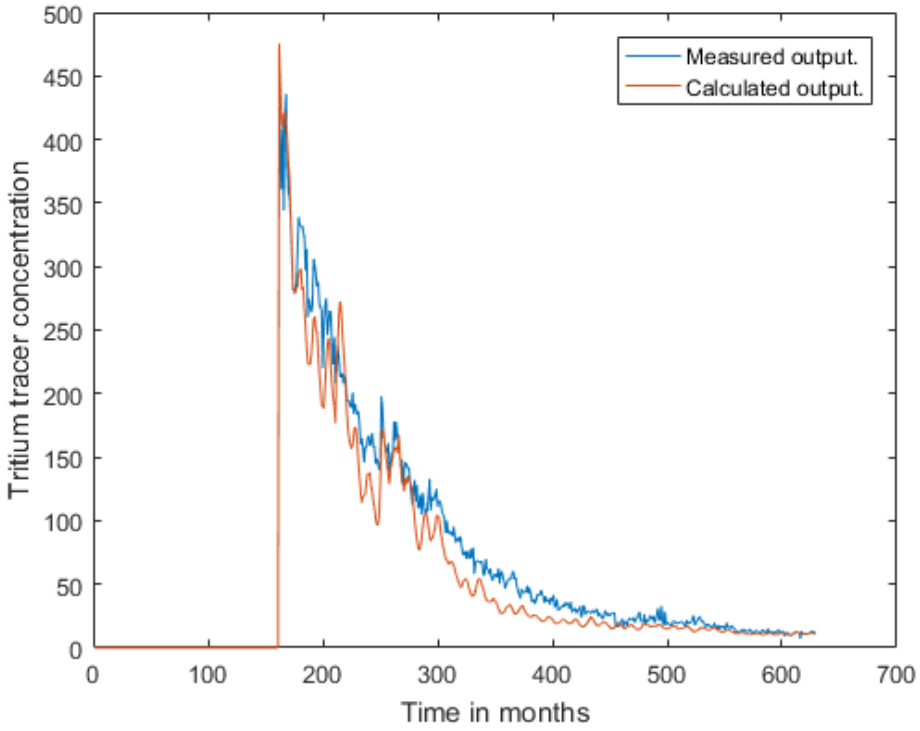

Figure 3: Amount of tritium in Danube river for mean residence time of water = 7.24 in comparison to real measurements.

3 Conclusion

Over the course of this classes we implemented integral equation using exponential transit time distribution function and then used both trial-error and inverse modelling method learning about dangers coming from using manually implemented methods.

Implemented code allowed us to calculate a mean residence time of water for tritium tracer in Upper Danube. Using a mean value of tritium concentration in the precipitation as input data we came very close to real measurements.

4 Appendix – Source Code

4.1 Plotting opady.prn and dunaj.prn

```
figure
hold on
in = importdata('opady.prn');
out = importdata('dunaj.prn');
set(gca, 'YScale','log');
plot(in(:,2));
plot(out(:,2));
xlabel('Time in months');
ylabel('Tritium tracer concentration');
legend('opady.prn - measurements in the precipitation.'
    , 'dunaj.prn - measurements in the river.');
hold off
```

4.2 Convolution integral using exponential model

```
function result = convInt(c_in, t, tt)
    % c_in - input values
    % t    - timestamp
    % tt   - mean residence time
    lambda = 4.696e-3;
    result = 0;
```

```
    for i = 1:t
        result = result + c_in(i, 2) * ...
            1/tt * exp(-(t - i)/tt) * ...
            exp(-lambda *(t - i));
    end;
end
```

4.3 Trial-error method

```
input = importdata('opady.prn');
output = importdata('dunaj.prn');

in_count = size(input, 1);
% we will have to compare subsequent rmse
% so we may initialize one before loop
rmsePrev = 100000000;
% lower the step to get more precise tt value
step = 0.001;

for tt = 1:step:1000
    integralResult = zeros(in_count, 1);
    % we don't have output data for 0 .. 161
    for i = 162:in_count
        integralResult(i) = convInt(input, i, tt);
    end
    errors = (output(:, 2) - integralResult).^2;

    rmseCurr = sqrt(sum(errors) / (in_count - 161));
    % when this condition is met search is over -
       current value of tt is
    % the result
    if (rmseCurr > rmsePrev)
        break;
    else
        rmsePrev = rmseCurr;
    end
end
```

```
disp(tt);
```

4.4 Objective function

```
tt = 1;
[optimal_tt, RMSE] = fminunc(@objective, tt);
optimal_tt(1)
```

```
function err = objective(tt)
    input = importdata('opady.prn');
    output = importdata('dunaj.prn');
    result = zeros(1, 161);
    % we don't have output data for 0 .. 161
    for i=162:size(input, 1)
        result(i, 1) = convInt(input, i, tt);
    end

    err = sum((output(:,2 )- result(:, 1)).^2);
end
```

4.5 Final plot

```
input = importdata('opady.prn');
output = importdata('dunaj.prn');

in_count = size(input, 1);
% calculated optimal tt
tt = 7.24;
integralResult = zeros(in_count, 1);

% we don't have output data for 0 .. 161
for i = 162:in_count
    integralResult(i) = convInt(input, i, tt);
end
```

```
figure;
plot(output(:,2));
hold on;
plot(integralResult);
xlabel('Time in months');
ylabel('Tritium tracer concentration');
legend('Measured output.', 'Calculated output.');
```