Simulation of Transport of the pollutants in the river

Daniel Stefanik

1. Explanation of aim

The aim of this exercise was to create model of physical phenomenon - transport of the pollutants in the river using Taylor model. Implementation of computer simulation helps to understand the mechanisms describing the phenomenon and it theoretical concept. Another objective is to test stability of system for various parameters. Next step is to verify model by checking if it behaves in accordance with the law of mass conservation.

2. Used tools

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

- *pylab* - creating diagram of relationship between tracer concentration and position and between tracer concentration and time,
- *display* from *IPython* - proper display of the plot,
- *numpy* - initiation of matrix and grid for plots.

3. Results

a. Parameters and stability

Change of parameters can destabilize the model or cause other undesirable behavior. Considering the *dx* parameter:

- for 0.1 or less the system is not stable,
- in range between 0.2 and 1.0 system behaves as expected,
- for value greater than 1.0 the computation is not accurate.

Considering *dy* parameter:

- for 0.01 or less computation is to slow to see the result on the diagram,
- in range between 0.5 and 1.0 system behaves as expected,
- for value greater than 8.0 system is not stable.

We can observe behaviour of destabilized system on diagram 3.1.

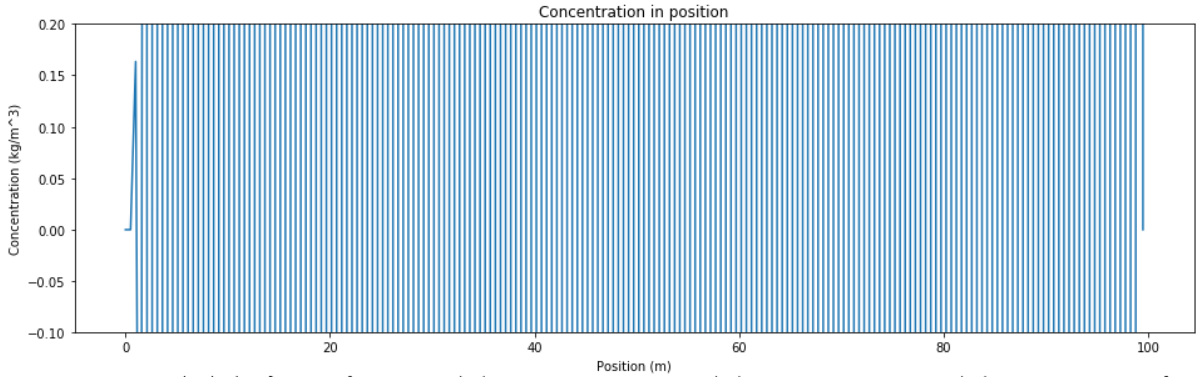

Diagram 3.1. Destabilized system

b. Tracer concentration and position

Diagrams below (Diagram 3.2.) present 3 stages of moving tracer in 1D model of the river. We can observe advection process as well as dispersion process. The “wave” of tracer concentration is more flat on the next stages.

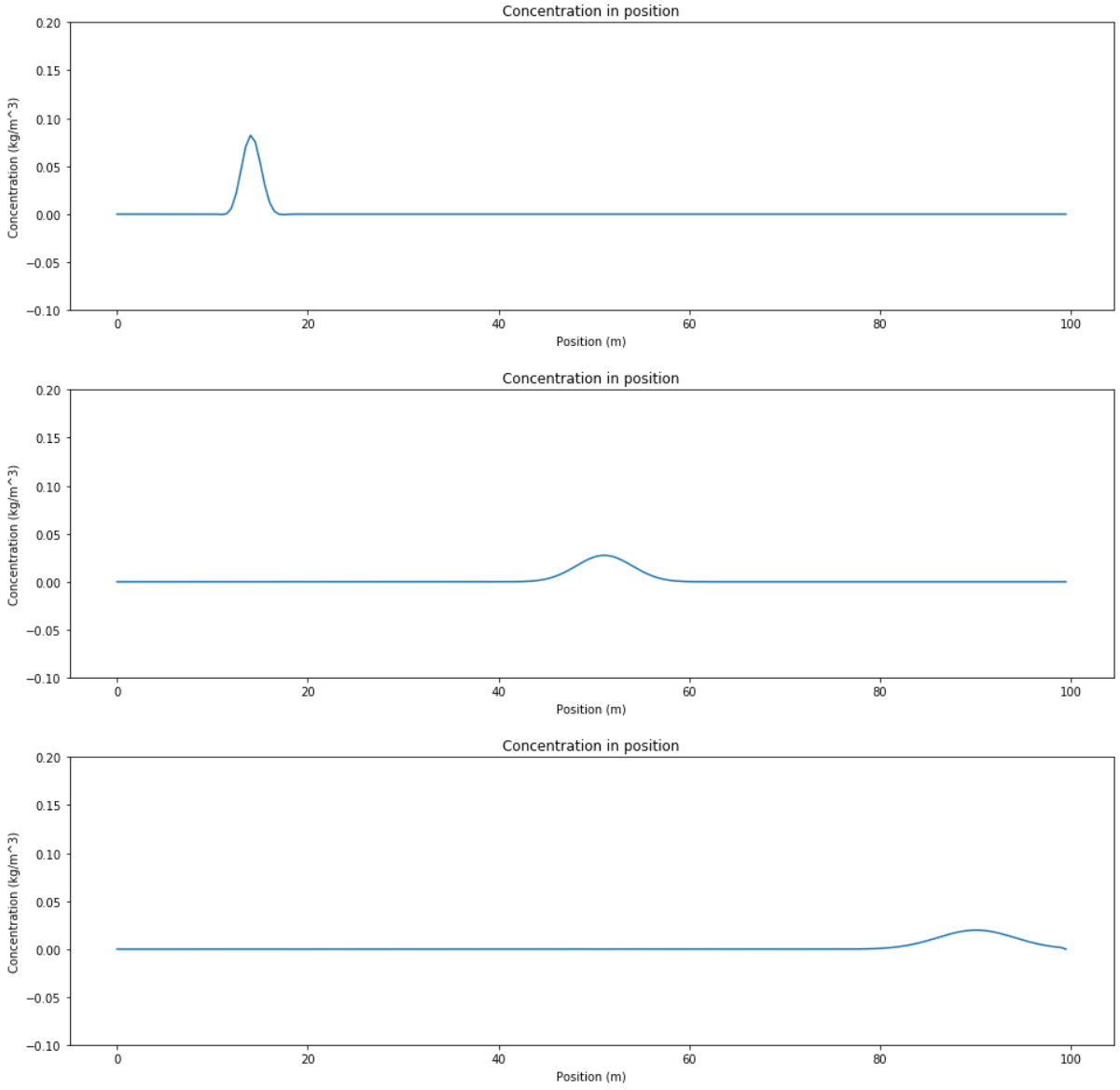

Diagram 3.2. Concentration of tracer in position

c. Mass conservation law and tracer concentration in time

To verify if the model is in accordance with the law of mass conservation We can observe the concentration of tracer in time like in Diagram 3.3. There are considered 3 measurement points:

1. in 20 m of the river - on start of tracer the beginning of fluid influx,
2. in 50 m of the river - halfway,
3. in 90 m of the river - end of considered area.

In this points there was calculated sum of tracer and in the title of Diagram 3.3. we can see the result of this calculations. We can observe that the law of mass conservation was preserved.

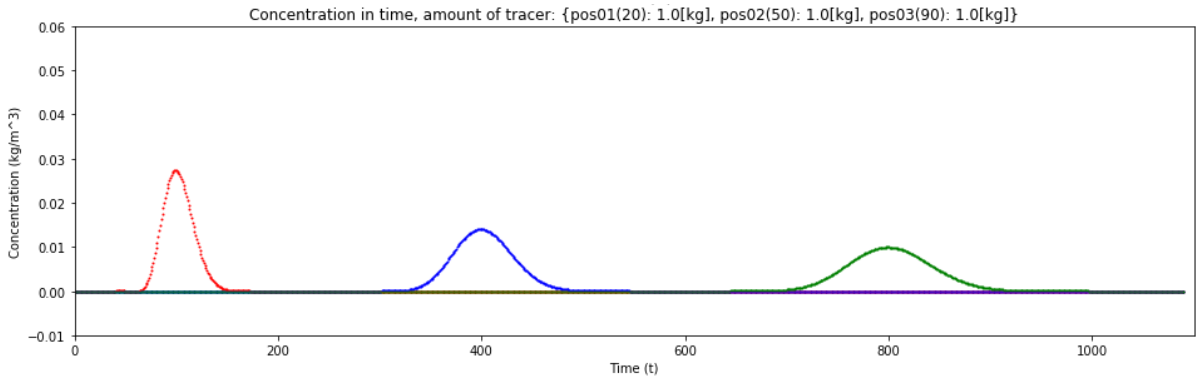

Diagram 3.3. Concentration of tracer in time

4. Conclusions

For some parameters (e.g. *dx = 2.0, dt = 2.0*) we can observe a phenomenon which has not reflection in reality: concentration of tracer is less than 0.0 (see Diagram 4.1.). This is consequence of using Taylor model.

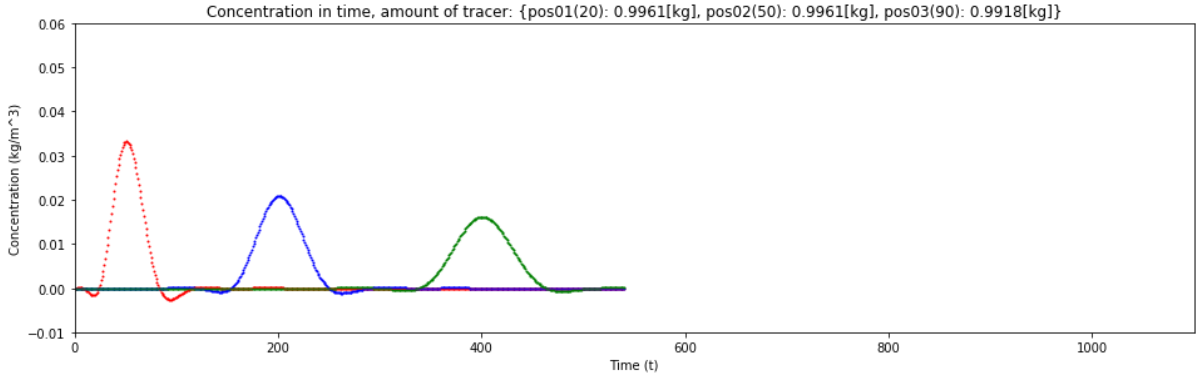

Diagram 4.1. Concentration of tracer in time

Created model of transport of the pollutants in the river using Taylor model show us how on in what time the tracer is distributed in river. Considering various parameters can provide information of stability of the system. Verification of correctness of the system using the law of mass conservation was successful.

5. Code listening

```
import numpy as np

% matplotlib inline
import time
import pylab as pl
```

```
from IPython import display

# left side - Dirichlet condition
# c(0,t) = 0
# right side - von Neumann condition
# dc/dx (L,t) = 0
# Initial condition:
# c(x,0) = f(x)
def do_plot(v, grid):
   pl.subplot(211)
   pl.plot(grid, v)

   display.clear_output(wait=True)
   display.display(pl.gcf())
   pl.cla()
   pl.ylim(-0.1, 0.2)

   pl.xlabel('Position (m)')
   pl.ylabel('Concentration (kg/m^3)')
   pl.title('Concentration in position')
#     pl.xlim(0,length)
#     time.sleep(0.2)


def do_plot_time(n, v, c):
   pl.plot(n, v, c, markersize=1)


def f(x):
   #     print("x: "+str(x)+" lip: "+str(int(lip))+" r: "+str(x != int(lip)));
   result = c0
   if x != int(lip / dx):
       result = 0.0
   else:
       print("TRUE")
   return result


def init_matrix():
   matrix = np.zeros((SIZE_X, SIZE_T))
   matrix[0, :] = 0.0
   for x in range(0, SIZE_X):
       matrix[x, 0] = f(x)
   print(matrix.round(4))
   return matrix


def do_compute(n, j, c):
   return c[j, n] + p1 * c[j + 1, n] - p2 * c[j, n] + p3 * c[j - 1, n] + p4 * c[j - 2,
n]
# Variable initiation

length = 100.0  # 100.0 # m
```

```
width = 5.0  # m
depth = 1.0  # m
U = 0.1  # m/s - advection coefficient - mean flow velocity
D = 0.01  # m^2/s - dispersion coefficient
lip = 10.0  # m - location of the injection point
lmp = 90.0  # m - location of the measurement point
air = 1.0  # kg - amount of injected tracer

dx = 2.0  # - spatial resolution
dt = 2.0  # - time step

epochs = 1100.0
# f(x) = 1 # - function describing the initial distribution of the tracer


c0 = air / (width * depth * dx)  # - initial concentration in the injection point

Ca = U * dt / dx
Cd = D * dt / (dx ** 2)

SIZE_T = int(epochs / dt)  # - number of time steps
SIZE_X = int(length / dx)  # - number of computational nodes

plotEvery = 10

p1 = Cd * (1 - Ca) - (Ca / 6) * (Ca ** 2 - 3 * Ca + 2)
p2 = Cd * (2 - 3 * Ca) - (Ca / 2) * (Ca ** 2 - 2 * Ca - 1)
p3 = Cd * (1 - 3 * Ca) - (Ca / 2) * (Ca ** 2 - Ca - 2)
p4 = Cd * Ca + (Ca / 6) * (Ca ** 2 - 1)

c = init_matrix()

fig = pl.figure(figsize=(16, 10))

gridX = np.arange(0, length, dx)
gridT = np.arange(0, epochs, dt)

s1 = 0
s2 = 0
s3 = 0

point01 = 20
point02 = 50
point03 = 90

mp01 = int(point01 / dx)
mp02 = int(point02 / dx)
mp03 = int(point03 / dx)

print(c0 * dt)
for n in range(SIZE_T - 1):
   for j in range(2, SIZE_X - 1):
       c[j, n + 1] = do_compute(n, j, c)
```

```
   mpv1 = c[mp01, n] * dt * (width * depth) / 10
   mpv2 = c[mp02, n] * dt * (width * depth) / 10
   mpv3 = c[mp03, n] * dt * (width * depth) / 10

   s1 += mpv1
   s2 += mpv2
   s3 += mpv3

   if n % plotEvery == 1:
       do_plot(c[:, n], gridX)
       print("s1: " + str(s1) + " s2: " + str(s2) + " s3: " + str(s3))

       pl.subplot(212)
       pl.xlim(0, epochs)
       pl.ylim(-0.01, 0.06)
       pl.xlabel('Time (t)')
       pl.ylabel('Concentration (kg/m^3)')
       pl.title('Concentration in time, amount of tracer: {pos01(' + str(point01) + '):
' + str(
           round(s1, 4)) + '[kg], pos02(' + str(point02) + '): ' + str(round(s2, 4)) +
'[kg], pos03(' + str(
           point03) + '): ' + str(round(s3, 4)) + '[kg]}')

   do_plot_time(n, mpv1, 'ro')
   do_plot_time(n, mpv2, 'bo')
   do_plot_time(n, mpv3, 'go')
```