OM-MADE
for non-programmers:
a short guide

Anne-Julie TINET and Pauline COLLON

Université de Lorraine, CNRS, GeoRessources, F- 54000 Nancy, France

Contacts : anne-julie.tinet@univ-lorraine.fr ; pauline.collon@univ-lorraine.fr

Contents

Preamble

OM-MADE (https://github.com/OM-MADE/OM-MADE) is an open-source program designed to simulate one-dimensional solute transport in multiple exchanging conduits and storage zones. OM-MADE is based on the resolution of classical mass conservation equations. In OM-MADE, all parallel and exchanging flow zones are divided along the direction of flow into reaches, in which all model parameters are kept constant. The total flowrate may be modified through lateral in and outflows. The solute may also be affected by decay processes either in mobile or immobile zones. Each reach is subdivided into discrete segments of equal length. The partial differential equations can be solved using two second order schemes, one based on an operator-split approach, the other on Crank-Nicholson pondered scheme.

OM-MADE has been validated against analytical and existing numerical solutions. It is also successfully demonstrated on one tracer test realised in the karstic system of Furfooz.

If you use OM-MADE, please cite the following reference: "**Tinet A.J., Collon P., Philippe C., Dewaide L., Hallet V. (Subm.). OM-MADE: an open-source program to simulate one-dimensional solute transport in multiple exchanging conduits and storage zones. Computers & Geosciences**"

- ⇨ This paper details all technical aspects of OM-MADE, the hypothesis, the equations, the numerical schemes (Appendix A) and the validations made against analytical and numerical solutions.

Installation and requirements

OM-MADE is written in Python™ v3. Therefore, it might not be compatible with earlier versions. Futhermore it uses several Python libraries (numpy and matplotlib).

We thus recommend to use an integrated environment of development like Spyder that you can easily download here : https://pythonhosted.org/spyder/installation.html or with the Python Scientific Distribution Anaconda : https://www.anaconda.com/download/ This solution will automatically install all packages (numpy and matplotlib) that are required.

Content of the package

On the GitHub platform, you will find in OM-MADE folder, all files required to launch the program:

- `OM-MADE_generic_Main.py` : the main script to launch a "standard" OM-MADE simulation, i.e. reading the `_INPUTFILES.txt` file to determine the paths of the input data and output folder. No crossplots are directly generated by this program. Results are provided in ASCII files as described in Tinet et al., 2019.
- `_INPUTFILES`.txt : this text file contains the path of the folder were the input data files should be located and the output files should be written.
- The folder `Codes_OMMADE` should not be modified by non-programmers. It contains the core of the python program:

  `classDataPoint.py` : this file contains the class DataPoint which contains the physical data required for computation for one point (flow type and location) of the domain. It allows the calculation of advection (either with explicit, Lax-Wendroff scheme: `advectionPoint_explicite` which returns the concentration post-advection, or with Crank-Nicholson centred scheme: `advectionPoint_cranknicholson)`, and updates the equation system to compute the dispersion (implicit scheme: `dispersionPoint`), the mass exchange between two flow types point at the same location (`massExchange`) and mass exchange through either degradation on lateral flow (`massLoss`). This class is used in `readData.py` and in the class `DataPoint`.

  - `classParameters.py` : The class Parameters contains all physical parameters for one zone (flow type and reach) of the domain, namely :
    - area of the zone [L^2] : `A_`
    - dispersivity [L^2 / T], the dispersivity is considered constant: `D_`
    - exchange rates [L^2/T], the exchange rate is generated by the method `setAlpha` and relates 2 zones : `alpha_`
    - degradation rate [1/T], the degradation rate is considered constant: `lambda_`
    - lateral inflow [L^2/T], inflow of the domain: `ql_`
    - lateral outflow [L^2/T], inflow of the domain: `qlout_`
    - lateral concentration [CU], equilibrium concentration constraining the in flow: `cl_`

    This class is used in `timeLoops.py`.

  - `readData.py` : this file contains several functions used to read and store the different input parameters.
    - `readGeneralData` reads the simulation data (time and space steps, simulation time, …)
    - `readDataset` reads the physical parameters associated to each reach and zone
    - `readBound` reads the boundary conditions associated to the input of the model
  - `timeLoops.py` : This module contains the time loop functions. Advection loop contains the iteration to perform advection for one time step. Several time steps are possible due to the Courant-Friedrich-Levy (CFL) condition: if the user time step is bigger than the maximal time step for the CFL condition (`dtCFL`), the advection is resolved several successive times using `dtCFL`, until reaching the user time step `dt`. If the simulation time step `dt` < `dtCFL`, then the advection is only solved once with dt. The time loop does the overall loop for each time step, until reaching the total simulation time. It returns a matrix containing the concentration for each printing location at each printing time step with the form C[zone][location, time].

- The folder `Demo` contains example of input ASCII files, launched by `OM-MADE_generic_Main.py` :
    - `_BOUND.txt` gathers boundary parameters.
    - `_PARAM.txt` describes each reach and flow zone.
    - `_SIMUL.txt` contains the parameters of the simulation (time step, space step, printing times and steps…)
- The folder `Validations` contains all validation simulations (input data, main program and results). For each validation, a specific main program has been written that directly produces the result crossplots that are presented in the paper (Tinet et al., subm.), thanks to the use of matplotlib library:
    - `_Main_PureAdvection.py` launches the data of the `PureAdvection` folder. It provides the comparison with an analytical result of a pure advective transport.
    - `_Main_PureDispersion.py` launches the data of the `PureDispersion` folder. It provides the comparison with an analytical result of a pure dispersive transport.
    - `_Main_AdvectionDispersion.py` launches the data of the `Advection_Dispersion` folder. It provides the comparison with an analytical result of a transport combining advection and dispersion.
    - `_Main_WSADECompare.py` launches the data of the `Comparison_WSADE` folder. It provides the comparison with an analytical result corresponding to a Weighted Sum Advection Dispersion Equation used to model the transport in two parallel advective zones (Field & Leij, 2012).
    - `_Main_DADECompare.py` launches the data of the `Comparison_DADE` folder. It provides the comparison with the analytical solution called Dual Advection Dispersion Equation used to model the transport in two exchanging advective zones (Field & Leij, 2012).
    - `_Main_OtisAppli1.py` launches the data of the `Comparison_OTIS_App1` folder. It provides the comparison with the Application 1 published with the OTIS software (Runkel, 1998). It considers a case with several reaches and a storage zone.
    - `_Main_OtisAppli3.py` launches the data of the `Comparison_OTIS_App3` folder. It provides the comparison with the Application 3 published with the OTIS software (Runkel, 1998). It considers a case with one reach and a first order degradation rate.
- The folder `Furfooz_TracerTest3` contains the input data, main program and results corresponding to the tracer test realized in the Furfooz karstic system and presented in the paper. It consists in an example of application with 2 successive reaches, 3 flow zones (2 are mobile zones, 1 is a storage one), with exchanges between zones and internal decay of the tracer. Again, in this application, a specific main program (`tracage3_main.py`) has been written and is provided allowing to directly visualize the Breakthrough Curves in the Spyder interface.

The architecture of OM-MADE is summarized in this figure, from Appendix B of Tinet et al., 2019:

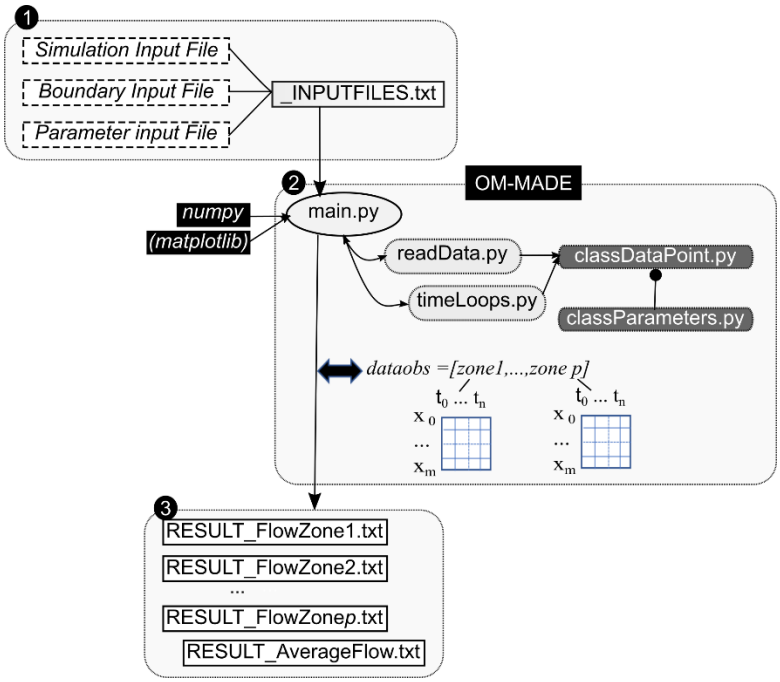


Figure 1: OM-MADE architecture (Tinet et al., 2019): Three text input files are required by the *_INPUTFILES.txt file* (1). They contain all information to run a specific simulation. OM-MADE (2) should be launched by running the main program (several examples are provided), which calls four python modules. The result of a simulation is stored in the *dataobs* variable, which is a list of arrays containing, for each zone, the concentration for each printed time and distance. This variable can be used to generate several text files containing the result of the simulation (3)).

Input parameters

The easiest way to use OM-MADE is to create a sub-folder in the OM-MADE directory for each simulation. You will store in it the input data and the output data would be automatically written in the same folder.

Thus, you should modify the `_INPUTFILES.txt` file:

_INPUTFILES.txt

```
1  # ================================================================================
2  #  NAME of INPUT FILES
3  #
4  # Line 1 : Directory path for input and output files: it can be different from the repos
5  # Line 2 : Name of Simulation parameter input File
6  # Line 3 : Name of Boundary condition input File
7  # Line 4 : Name of Physical Parameter input File
8  #
9  # ================================================================================
10 #Directory path for input and output files
11 E:/Recherche/KARST/OMMADE/OMMADE/Demo/
12
13 # Simulation input File
14 _SIMUL.txt
15
16 # Boundary input File
17 _BOUND.txt
18
19 # Parameter input File
20 _PARAM.txt
21
```

- On line 11 : update the Directory path. Take care to use the symbol “/” and not the default “\” to indicate the path. Also, do not forget to add a final “/” at the end of the path.

- You can decide to use alternative names for the input parameters files, stored in the previous directory. In this case, update the names consistently: lines 14, 17 and 20.

Three input files are thus required by the program. In the generic adaptation used by the `OM-MADE_generic_Main.py`, the 3 files are stored in the `Demo` folder.

- `_SIMUL.txt` contains the parameters of the simulation (Numerical scheme, time step, space step, printing times and steps…)

_SIMUL.txt

```
1  # ================================================================================
2
3  #                    SIMULATION PARAMETERS
4
5  # ================================================================================
6
7  # Numerical scheme (0 = operator split, 1 = Crank-Nicholson)
8  0
9
10 # Space discretisation (dx,xmax)
11 1 669
12
13 # Time discretisation (dt, tmax)
14 180 56700
15
16 # Initial concentration
17 3.7 3.7
18
19 # Printing information, X,T (0=all steps, 1=regular, 2=imposed)
20 2 0
21
22 # Number of printing locations
23 5
24 # Printing locations (if necessary)
25 38
26 105
27 281
28 432
29 619
30 # Number of printing times
31
```

WARNING: when changing the time-step, the input concentration has to be carefully adapted as a linear interpolation of concentrations is performed to get the input concentrations for each time-step (see below). It is especially true for Dirac injections were the injected mass will correspond to the input concentration x 1 time-step.

Note that if you want to print less time or space steps than computed, the total number of printing times should be linked to time discretization parameters by:

Number of printing times = coef.(tmax/dt) + 1 with *coef*, the coefficient of your choice <= 1.

Of course, the chosen printing locations should be compatible with the chosen space step and maximal distance.

- `_BOUND.txt` gathers boundary parameters: number of boundary points (number of lines written after line 10), and then the time in a first column, followed by the corresponding concentration in each zone.

It is very important to note that at the inlet, a time-dependent imposed concentration conditions each zone. Therefore, both continuous and step-wise tracer injection can be simulated. **A Dirac injection can be approximated by imposing concentration at one-time step only**. When an inlet boundary condition

is not explicitly given in the input data file, its **value is computed by linear interpolation** (Figure 2). The total mass injected in the outlet along the simulation and for all zones may be calculated using the following equation, given that the concentration Cp in unit of mass per unit of volume. Boundary condition at the outlet corresponds to a zero diffusive flux condition. At initial step, a constant concentration is imposed in all reaches for each zone.

$$M = \sum_{p=mobile} \int_{t_0}^{t_{max}} C_p(x=0,t) Q_p \delta t$$

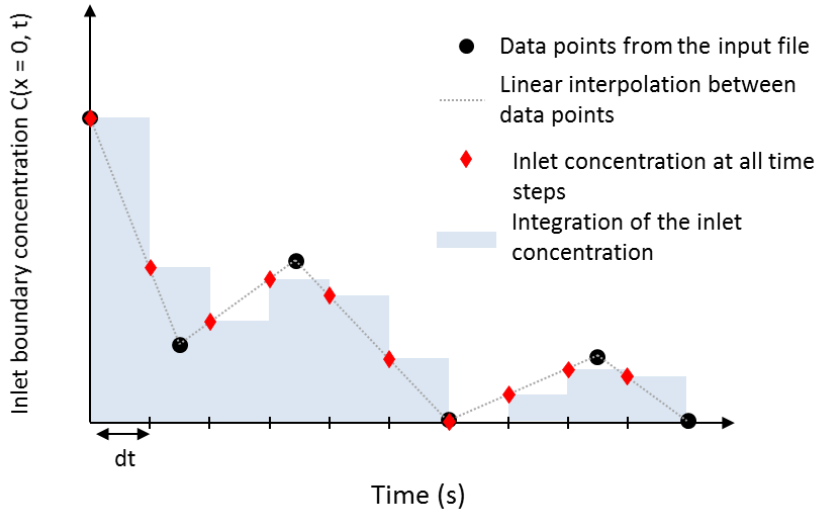


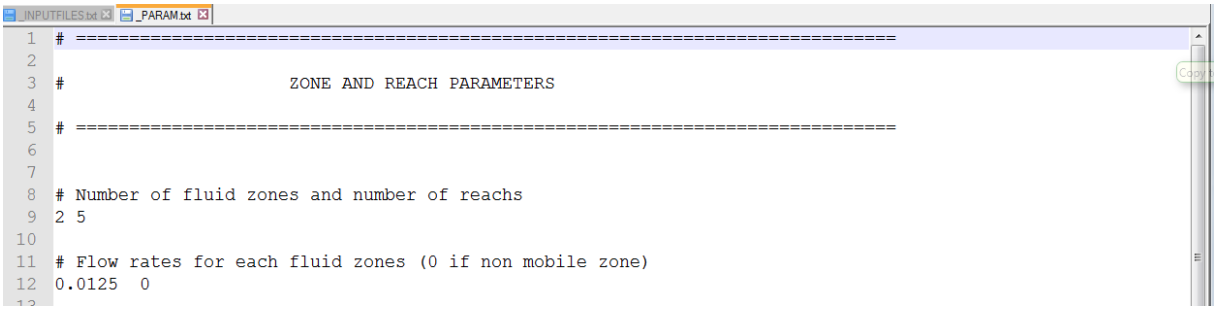
Figure 2: Boundary conditions (Tinet et al., 2019): the input concentration is computed by a linear interpolation between the successive values provided by the user.

_BOUND.txt

```
# ================================================================================

#                      FILE BOUNDARY DATA

# ================================================================================

# Number of boundary points
6

# Time   Concentration for each flow type
0  3.7  3.7
539 3.7 3.7
540  11.4  3.7
11339  11.4  3.7
11340  3.7  3.7
56700  3.7  3.7
```

In the above input boundary file example, the input concentration is equal to 3.7 CU until the third timestep (t=540s for dt=180s), then it rises to 11.4 during 60 timesteps (until t=11340 s) and goes back to 3.7 CU until the end of the simulation.

- `_PARAM.txt` describes each reach for each flow zone p. It starts with the number of fluid zones and reaches, and the flow rates for each zone. A null flowrate indicates a storage zone.



In the above example, there is 2 flow zones and 5 reaches. The first zone has an input flowrate of 0.0125 [L3/T], while the second one is an immobile zone.

Then, for each reach, the different parameters should be provided after indicating the starting point and length of the reach:

Name	Unit	Definition
A	$[L^2]$	Cross-sectional area of the zone p
Q	$[L^3.T^{-1}]$	Inflow inside the zone p
D	$[L^2.T^{-1}]$	Dispersion coefficient in p
q_{lin}	$[L^2.T^{-1}]$	Lateral inflow rate per unit of distance
q_{lout}	$[L^2.T^{-1}]$	Lateral outflow rate per unit of distance
C_{lin}	$[CU.L^{-3}]$	Solute concentration in lateral inflow
α_{pq}	$[L^2.T^{-1}]$	Exchange coefficient between zones p and q multiplied by the exchange surface area
λ	$[T^{-1}]$	First order decay coefficient in p

You can choose the units you want (meters or decimetres for L, seconds or hours for T, ppb or mg/l for CU, …) but you should take care to preserve self-consistency between them.

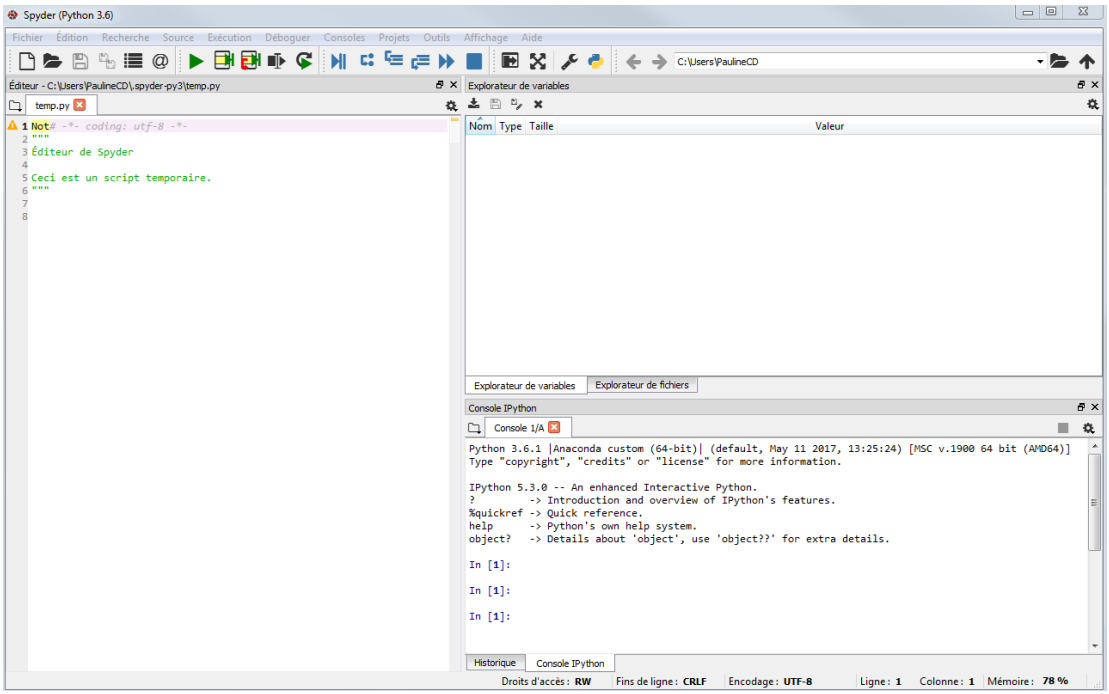
The exchange coefficients are written as a matrix a_{ij}, with i and j the numbers of the zones. Thus, the matrix should be symmetric with a null diagonal (e.g. : reach 3 in the following figure).

```
# ------------------------------------------------------------------------------
# REACH 1
# Start point and length of the reach
0. 38
# For each fluid zone
# Area [L2]  Dispersivity [L2/T] Degradation rate [1/T] Lateral inflow [L2/T] Lateral outflow [L2/T] Lateral concentration [CU]
0.30      0.12      0      0     0     3.7
0.05      0      0      0     0     0
# Exchange coefficients for each zone
0 0
0 0
# ------------------------------------------------------------------------------
# REACH 2
# Start point and length of the reach
38 67
# For each fluid zone
# Area [L2]  Dispersivity [L2/T] Degradation rate [1/T] Lateral inflow [L2/T] Lateral outflow [L2/T] Lateral concentration [CU]
0.42      0.15      0      0     0     3.7
0.05      0      0      0     0     0
# Exchange coefficients for each zone
0 0
0 0
# ------------------------------------------------------------------------------
# REACH 3
# Start point and length of the reach
105 176
# For each fluid zone
# Area [L2]  Dispersivity [L2/T] Degradation rate [1/T] Lateral inflow [L2/T] Lateral outflow [L2/T] Lateral concentration [CU]
0.36      0.24      0      4.545e-6     0     3.7
0.36      0      0      0     0     0
# Exchange coefficients for each zone
0 1.08e-5
1.08e-5 0
# ------------------------------------------------------------------------------
```

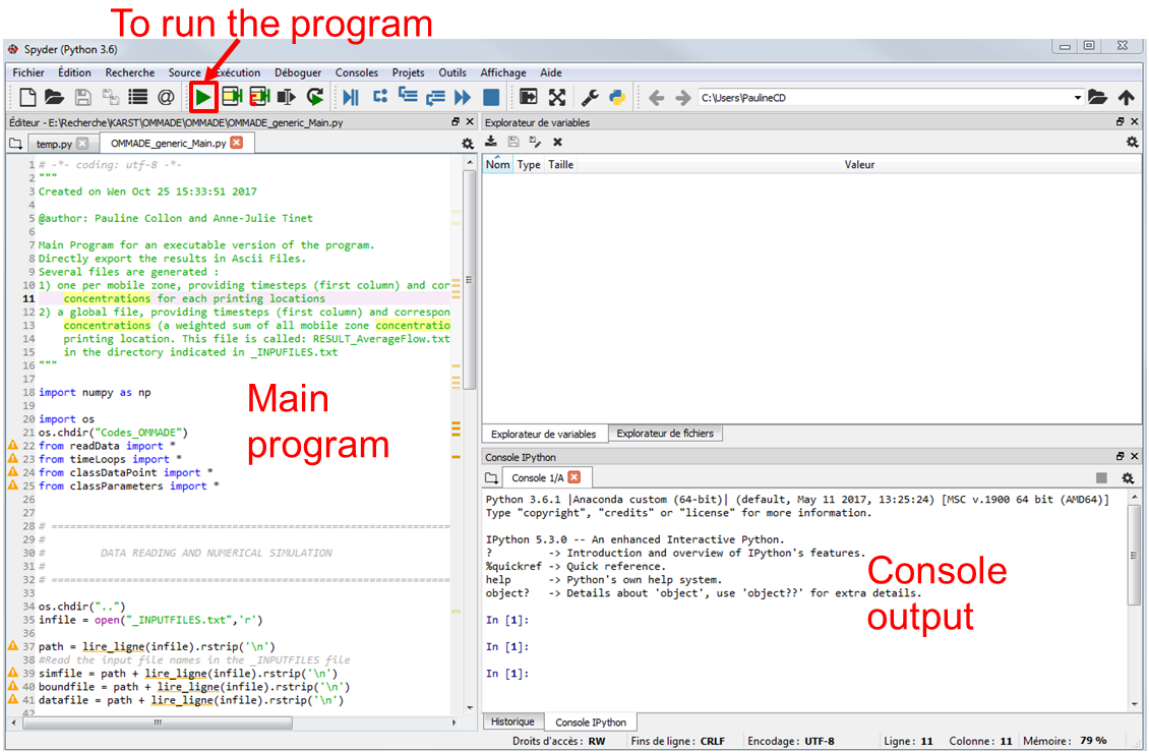
Run OM-MADE in Spyder-Anaconda

Once you have defined the input parameters in the 3 input files, the OM-MADE program can easily be run from Spyder-Anaconda. For example, on windows:

- Open Spyder (Start / Spyder). You obtain the following interface:



- Open the main program you want to launch, e.g. to launch the generic program : **File / Open /** , go to the folder of OM-MADE and select `OM-MADE_generic_Main.py`. The program appears on the left side of the screen.
- To launch the code, just click on the green arrow on the top bar, or on the menu **Run / Run the file.** Let the default parameters for the execution.



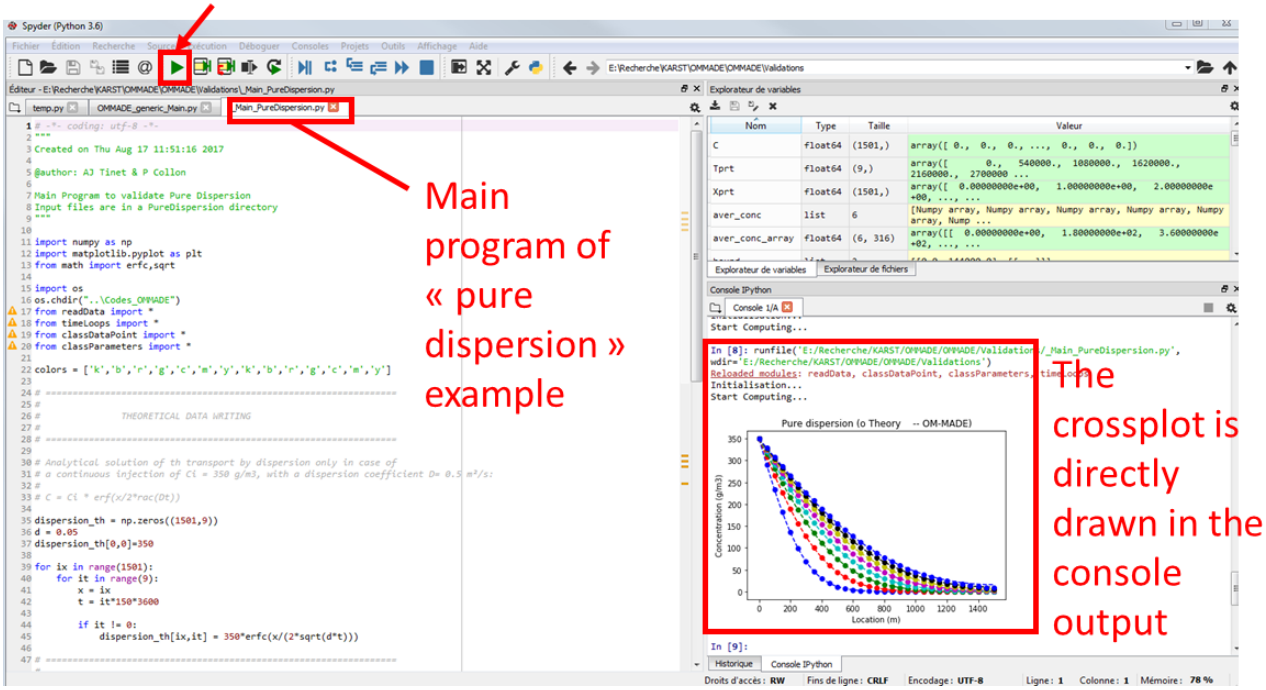
- You will read in the Console output all messages.
    - If a message indicates that the program does not find the data, it means that you have mis-written the path (or the input file names) in the `_INPUTFILES.txt` file.

```
  File "E:\Recherche\KARST\OMMADE\OMMADE\Codes_OMMADE\readData.py", line 31, in readGeneralData
    pfile = open(filename,'r')

FileNotFoundError: [Errno 2] No such file or directory: 'C:/Users/collon5/Desktop/OMMADE/
OMMADE/Demo/_SIMUL.txt'
```

    - If everything goes well, you will get the following message in the console output :

```
In [7]: runfile('E:/Recherche/KARST/OMMADE/OMMADE/OMMADE_generic_Main.py', wdir='E:/Recherche/
KARST/OMMADE/OMMADE')
Reloaded modules: readData, classDataPoint, classParameters, timeLoops
Initialisation...
Start Computing...

In [8]:
```

- In case you have run a specific Main program, like the ones provided in the Validations folder, you will also directly vizualize the crossplots in the console output :



Results

The result of the simulation is written in a matrix called `dataobs` of the form C[zone][location, time]. This matrix is read in the main program `OM-MADE_generic_Main.py` to generate ASCII files, that you can use the make cross-plots:

- one file per flow zone *p* containing the concentration through time at the printing locations, and the corresponding time steps in the first column: `RESULT_FlowZonep.txt`
- An additional output file `RESULT_AverageFlow.txt` contains also the corresponding average concentration for all mobile zones at each printing locations

Acknowledgements

The authors acknowledge USGS and its developers for providing the inspiring OTIS software, as well as M. Field and F. Leij for publishing the DADE solution.

The authors are grateful to both the University of Lorraine (France) and the University of Namur (Belgium) for supporting this research.

A part of this work has been financially supported by the program **Widen Horizons** of **Lorraine Université d'Excellence (LUE).**

References

Field M.S., Leij F.J. (2012). Solute transport in solution conduits exhibiting multi-peaked breakthrough curves. *Journal of Hydrology*, *440–441*, 26–35

Runkel R.L. (1998). One-dimensional transport with inflow and storage (OTIS): a solute transport model for streams and rivers. *Water-Resources Investigations Report 98-4018*

Tinet A.J., Collon P., Philippe C., Dewaide L., Hallet V. (Subm.). OM-MADE: an open-source program to simulate one-dimensional solute transport in multiple exchanging conduits and storage zones. *Computers & Geosciences, Subm.*