Audouin, Yoann; Fontaine, Jacques; Fouquet, Thierry; Goeury, Cédric; Leroy, Agnès; Pham, Chi-Tuan; Souillé, Frank; Taccone, Florian; Duron, Luc; Daou, Mehdi-Pierre; Sécher, Matthieu

A new Python3 module for TELEMAC-MASCARET dedicated to post-treatment: Postel

A new Python3 module for TELEMAC-MASCARET dedicated to post-treatment: Postel

Audouin Y., Fontaine J., Fouquet T., Goeury C., Leroy A., Pham C.-T., Souillé F. and Taccone F.
Laboratoire National d'Hydraulique et Environnement, EDF R&D, Chatou France
yoann.audouin@edf.fr

Daou M.-P.
Artelia Eau & Environnement
Echirolles France

Duron L.
Compagnie Nationale du Rhône
Lyon France

Sécher M.
Centre d'Ingénierie Hydraulique EDF
La Motte Servolex France

***Abstract* — This paper describes the work started during the April 2019 Coding Week. The goal was to write a Python module to perform dedicated TELEMAC-MASCARET post-treatments using NumPy, SciPy, Matplotlib. The aim was to have a fully documented, validated and easy to use toolkit. Also it should make it possible for users to easily add their own modifications to the plots. The documentation was done using Jupyter notebooks which serve as documentation, validation and examples. The module can process to 2D/1D extractions from a 3D/2D mesh file, extraction from a TELEMAC listing, extraction from a shape file, plot of vectors and streamlines, plot with masked dry zones, plot over a background image (that can be extracted from a WMS flux), post-treatment for MASCARET files, spectrum specific plots, computation of fluxes, volumes, wet sections and statistics calculation, interpolation on a regular grid.**

I. Introduction

TELEMAC-MASCARET is a suite of scientific codes developed as interconnected modules entirely written in Fortran. Surrounding TELEMAC-MASCARET and also distributed as open source codes, an ensemble of pre- and post-processing scripts entirely written in Python have been developed over the last 5 years ([10], [11] and [12]). In fact, Python is a portable, dynamic, extensible, free language, which allows (without imposing) a modular approach and object oriented programming. Python has been developed since 1989 by Guido van Rossum and many volunteer contributors. In addition of the benefits of this programming language, Python offers a large amounts of interoperable libraries enabling to facilitate post-processing tasks on 2D or 3D TELEMAC-MASCARET computation results.

In order to gather all those tools and give them a place to flourish, a new Python [8] module "POStreatment TELemac" (Postel) is created in the TELEMAC-MASCARET system [7]. It was developed only in Python 3 as Python will be retired by the end of the year. This module will be available in TELEMAC-MASCARET 8.1.

This module is developed with the guidelines below in mind:

- A user friendly list of classes and functions.
- Well documented and with examples.
- Easily customizable (you should be able to do it the way you want).

To match those guidelines the first thing done is to split this development in two parts "Data" and "Plot".

The "Data" part contains all the functions around extracting and manipulating data. It can be to extract 1D, 2D data from a file, interpolate data along a polyline or compute fluxes for example. This part mainly uses NumPy [2] and SciPy [1] libraries.

The "Plot" part contains all the functions to plot graphs. It contains 1D, 2D and 3D plotting functions and allows plotting mesh, vectors, scalar map and contours for instance. This part is mainly based on the Matplotlib [3] library. This part can both take as input data extracted with the "Data" part or custom data.

To provide a useful documentation and proper examples, Python Notebooks are used [4].

First, this paper gives a brief explanation of the internal structure of the module and a layout of a code using it. Then, a more thorough description of the "Data" part of the module is given. Finally we will show some of the graphs we can do with the "Plot" part.

II. Structure

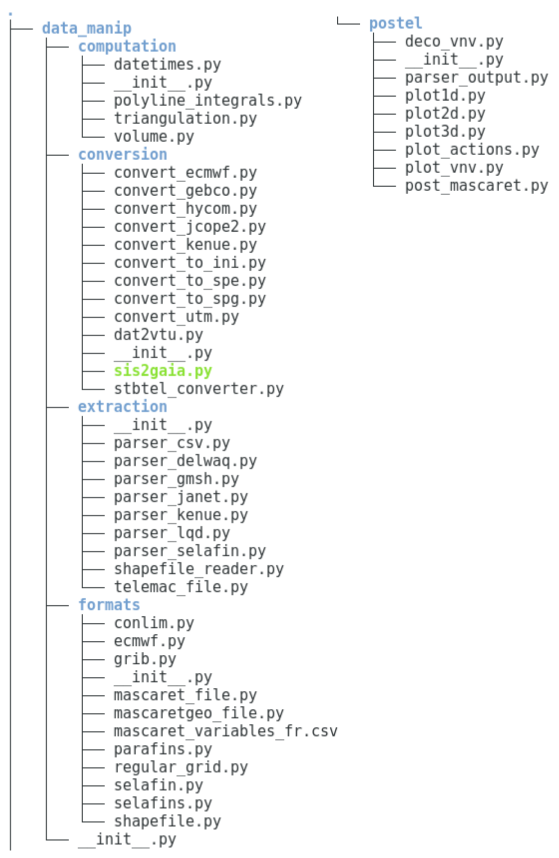
This part presents the organisation of the code sources of the module. Then, a post-treatment example using the Postel module gives a framework of its use. The access to the notebook documentation is finally introduced.

A. *Module structure*

The scripts of the module are located in the same folder as all the other TELEMAC-MASCARET scripts ($HOMETEL/scripts/python3), where $HOMETEL is the path to your installation of TELEMAC-MASCARET. The developments for this module are grouped in two folders: *postel* for "Plot" part and *data_manip* for "Data" part. The Figure 1 shows the content of those folders.

Not all of the content of *data_manip* will be described here but notebooks can be found for most of them.

All the scripts follow the PEP 8 [5] coding convention (we used Pylint [6] to check that convention).

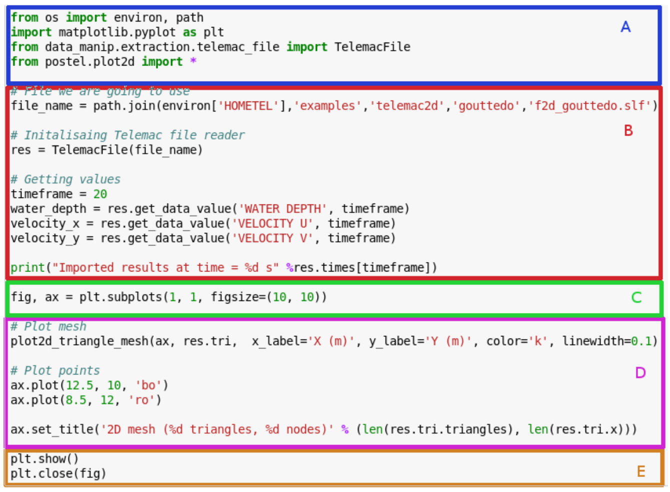


```
.
├── data_manip
│   ├── computation
│   │   ├── datetimes.py
│   │   ├── __init__.py
│   │   ├── polyline_integrals.py
│   │   ├── triangulation.py
│   │   └── volume.py
│   ├── conversion
│   │   ├── convert_ecmwf.py
│   │   ├── convert_gebco.py
│   │   ├── convert_hycom.py
│   │   ├── convert_jcope2.py
│   │   ├── convert_kenue.py
│   │   ├── convert_to_ini.py
│   │   ├── convert_to_spe.py
│   │   ├── convert_to_spg.py
│   │   ├── convert_utm.py
│   │   ├── dat2vtu.py
│   │   ├── __init__.py
│   │   ├── sis2gaia.py
│   │   └── stbtel_converter.py
│   ├── extraction
│   │   ├── __init__.py
│   │   ├── parser_csv.py
│   │   ├── parser_delwaq.py
│   │   ├── parser_gmsh.py
│   │   ├── parser_janet.py
│   │   ├── parser_kenue.py
│   │   ├── parser_lqd.py
│   │   ├── parser_selafin.py
│   │   ├── shapefile_reader.py
│   │   └── telemac_file.py
│   ├── formats
│   │   ├── conlim.py
│   │   ├── ecmwf.py
│   │   ├── grib.py
│   │   ├── __init__.py
│   │   ├── mascaret_file.py
│   │   ├── mascaretgeo_file.py
│   │   ├── mascaret_variables_fr.csv
│   │   ├── parafins.py
│   │   ├── regular_grid.py
│   │   ├── selafin.py
│   │   ├── selafins.py
│   │   └── shapefile.py
│   └── __init__.py
└── postel
    ├── deco_vnv.py
    ├── __init__.py
    ├── parser_output.py
    ├── plot1d.py
    ├── plot2d.py
    ├── plot3d.py
    ├── plot_actions.py
    ├── plot_vnv.py
    └── post_mascaret.py
```

Figure 1: Tree of the content of data_manip and postel

B. *Code structure*

Figure 2 is a template of what a post-treatment code using Postel looks like.



```python
from os import environ, path
import matplotlib.pyplot as plt
from data_manip.extraction.telemac_file import TelemacFile
from postel.plot2d import *

# File we are going to use
file_name = path.join(environ['HOMETEL'],'examples','telemac2d','gouttedo','f2d_gouttedo.slf')

# Initalisaing Telemac file reader
res = TelemacFile(file_name)

# Getting values
timeframe = 20
water_depth = res.get_data_value('WATER DEPTH', timeframe)
velocity_x = res.get_data_value('VELOCITY U', timeframe)
velocity_y = res.get_data_value('VELOCITY V', timeframe)

print("Imported results at time = %d s" %res.times[timeframe])

fig, ax = plt.subplots(1, 1, figsize=(10, 10))

# Plot mesh
plot2d_triangle_mesh(ax, res.tri,  x_label='X (m)', y_label='Y (m)', color='k', linewidth=0.1)

# Plot points
ax.plot(12.5, 10, 'bo')
ax.plot(8.5, 12, 'ro')

ax.set_title('2D mesh (%d triangles, %d nodes)' % (len(res.tri.triangles), len(res.tri.x)))

plt.show()
plt.close(fig)
```

Figure 2: Examples of code using Postel

Where:

- Part A: imports the different Python modules needed.
- Part B: extracts or computes the data. This is the “Data” part.
- Part C: initialises the Matplotlib figures.
- Part D: adds plots to the figures and sets its parameters.
- Part E: displays or saves the figure.

Parts C, D, E represent the “Plot” part.

Most of the examples you will see below follow that structure.

C. *Notebooks*

The Jupyter Notebooks are a mixed between a “IPython” [9] session and a website. This gives you a platform in which you can write a Python code and execute it. This leads to an interactive documentation where you can easily modify the code to match your own data.

This is used for documentation, examples and validation of the module. To start exploring the notebooks, run the following command:

```
jupyter-notebook $HOMETEL/notebooks/index.ipynb
```

This notebook will be your guide into the world of not only Postel but also all the Python scripts orbiting TELEMAC-MASCARET. It is an index of all the functionalities you have access to and a link to a notebook for all of them.

To display them you will need to install jupyter (https://jupyter.org/) follow the instructions on their website.

III. DATA MANIPULATION

This part will give an overview of the functionalities of data extraction and computation functions of the Postel module.

A. *How to extract data from files*

The data extraction part allows us to obtain different types of information from a mesh or a TELEMAC result file (whether in med or selafin format). The class *TelemacFile* contains several parameters and methods designed for this purpose. The properties *meshx* and *meshy* give the coordinates of each node of the mesh, while the method *get_timeseries_on_nodes* extracts a time series of variables from the graphical outputs on a given node or point defined by its coordinates. It is also possible to retrieve the spatialized value of a variable at a given frame with the method *get_data_value*. This spatial extraction can also be interpolated on a regular grid with the method *interpolate_on_grid*, which permits to compare two results on different meshes using the *field_diff_on_grid* method. Finally, the method *get_liq_bnd_info* provides information on the location of boundaries, their type, as well as the table of correspondences between the global node number and the boundary condition number.

When the results come from a TELEMAC-3D simulation, it is possible to use them in several ways thanks to the module. It is possible to extract horizontal sections, either following a plane of the three-dimensional mesh with the *get_data_on_horizontal_plane* method, or having a fixed elevation with the *get_data_on_horizontal_slice* method. It is also possible to extract the data on a vertical section with fixed coordinates using the *get_data_on_vertical_plane* method, and to perform a time series of the value of a variable in the entire water column at a given point in the horizontal plane with *get_timeseries_on_vertical_ segment.*

In addition to the extraction of data from a result file, Postel offers the possibility of reading into the listing outputs to retrieve certain information thanks to the class *OutputFileData*. This gives, among other things, the possibility to know the time profile of the simulation (which physical time corresponds to which iteration) with the method *get_time_profile*, the name of the study with the method *get_name_of_study*, or the mass fluxes at the boundaries and the mass-balance data with the method *get_value_history_output*. The user can also customize his query by giving a word to find in the listing so that the code returns information on the corresponding line using the method *get_user_defined_output*.

It is also possible to read shapefiles and extract polylines or polygons with the *shapefile_reader* class, but also to read and write information in csv format based on *savetxt* function of NumPy.

B. *How to compute data from extractions*

After the extraction of the data from the result files, the Postel module provides various functionalities to calculate the values of interest associated with the hydraulic variables.

For a given polygonal chain, it is indeed possible to compute the wet area with the function *wet_area_2d* after a TELEMAC-2D result file extraction of the water depth. The *flux_2d* function can be used to compute the flux of a scalar through this chain. By additionally extracting the velocity components in both directions, it is also possible to calculate the flow rate through this polygonal chain using the *flux_2d* function. This same function is also able to calculate solid discharge if the corresponding variables are previously extracted from a SISYPHE result file.

The function *volume_calculation* allows the integration of a variable extracted from a TELEMAC result file over the entire domain. It can be used, for example, to calculate the total volume of water in a domain by taking as an input, the extraction of water depths in the results file and the information of coordinates and triangulation of the mesh from *TelemacFile*. This can also be applied to erosion/deposit volume computation, if the evolution variable of the SISYPHE results file is extracted.

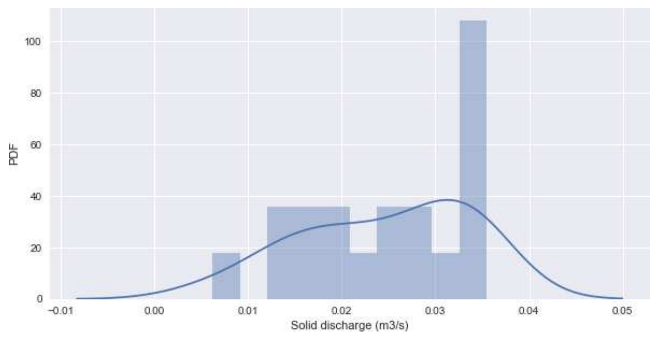


Figure 3: Probability Density Function of the solid discharge

Statistical information from your data can be simply computed using the Python module Pandas have a look at the notebook *statistic* for a more detailed explanation. Figure 3 shows the result of the said notebook.

C. *How to post-treat MASCARET files*

The first step for MASCARET file post-treatment is to be able to extract the data. There are two main classes to allow this:

- The class *MascaretGeoFil*e allows reading and writing geometry files.
- The class *MascaretFile* allows reading different types of a MASCARET result file (whether in Rubens or Opthyca formats).

These classes contain several attributes and functions designed for post-treatment tasks.

Let us begin with the class *MascaretGeoFile*, the *summary* method gives the reaches and associated sections in a text format. These two items describe geometry in MASCARET. In this class, the *reaches* attribute stores the object list of reaches and each reach contain multiple Section objects.

The class *Reach* contains methods to get the pk with *get_section_pk_list* method, to get the index list of section in reach object with *get_section_id_list* method, in addition to get of the section index of geometry from previously index with *get_section_idx* method.

The class *Section* contains mainly geometric attributes which are:

- *axis*: x, y coordinates of hydraulic axis.
- *x, y, z*: 1D-arrays for coordinates *x y z respectively*.
- *distances*: 1D-array with cumulative distance from first point along the section.
- *nb_points*: number of points.
- *limits*: a dict with the position and name of all limits.

Note in class *MascaretGeoFile* that the save method allows us to write a geometry file after having modified the object attributes.

The *MascaretFile* is based on the same principle with the attribute *reaches* containing its associated sections (but in this case the geometry is not set because the result file do not provide it). This class contains the post-treatment methods for results files. The *summary* method gives variables and number of temporal frame in addition to the information provided bythe corresponding method in class *MascaretGeoFile*. The *get_position_var* method returns the variable index from the variable name given. The values along a reach for one time step are extracted with *get_values_at_reach* method for the variable index and reach index. Furthermore the *get_series* method allows us to get the temporal values at a given section.

Then, the step after data extraction is to be able to draw them in a graphic. For this, the same 1D plots function is used than TELEMAC (to see IV.A).

Another post-treatment step is writing results for specific variables or specific time steps with *write_optfile* method. Finally there is also *export_as_lig* method allowing continuation of computation with the creation of LIDOP file from a given time step.

The different examples can be found in the notebook *example_mascaret*.

IV. Plot

This part will give examples of the different visualization tools available in Postel ranging from simple 1D plots to more complex 2D plots with multiple layers and 3D plots.

D. How to do 1D plots

From data extracted from TELEMAC result file, 1D plot can be done with abscissa and ordinate data. This can either be timeseries or plot along polygonal chain or any other data. Several plot examples are given in the *example_plot1d* notebook as well as the corresponding extraction function. In Figure 4 and Figure 5 an example of plot of time series on points and plot along a polygonal chain are shown.

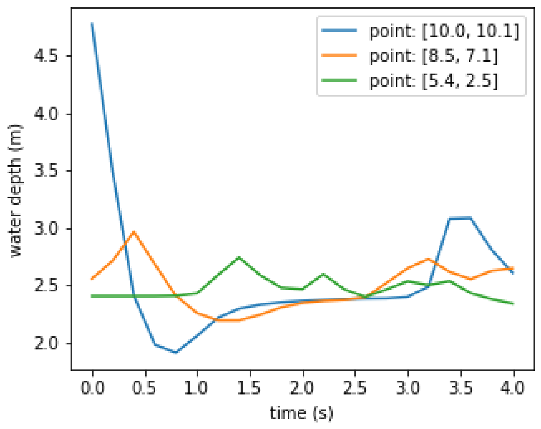


Figure 4: Example of 1d plot of time series on different points

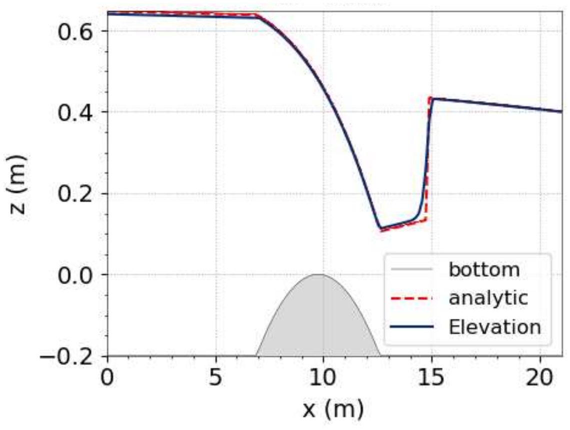


Figure 5: Example of 1d plot along a polyline

E. How to do 2D plots (x, y) plane view

In this section we mainly focus on the functions available in *postel/plot2d.py* which are described in detail in the *example_plot2d* notebook.

The first function of 2D plots is *plot2d_triangle_mesh*. It allows the visualization of 2D mesh and takes as argument the triangulation contained in the TELEMAC result file, which is *tri*, a parameter of the *TelemacFile* class. At the same time, it can be useful to visualize boundary conditions types and numbers. It can be done with the *plot2d_annotate_bnd* and *plot2d_annotate_liq_bnd* functions respectively. An example of 2D mesh plot with boundary annotations is shown on Figure 6.

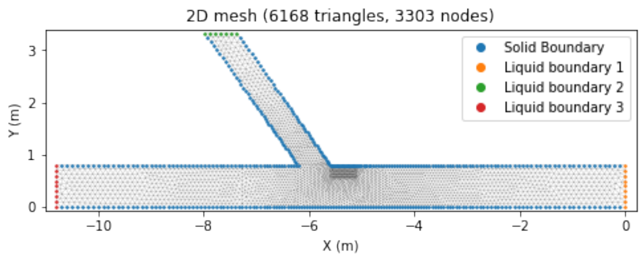


Figure 6: Example of mesh and boundary conditions plot with the confluence test case of TELEMAC-2D

To plot a scalar like water depth, elevation or velocity norm two functions are available in *plot2d*, *plot2d_scalar_map* and *plot2d_scalar_filled_contour.* On Figure 7 and Figure 8 a scalar map and a scalar filled contours are shown with the mesh in the background. In addition, non-filled contours can also be plotted with the function *plot2d_scalar_contour*, as it is shown on figure 7. Note that these functions take as argument a mesh that can either be a triangulation or a regular grid and the scalar data extracted from that mesh. Colour bar can be customized via the *vmin*, *vmax* arguments for maximum and minimum values as well as *nv* for the number of ticks. In the case of contours, a list of values can be provided instead with the *levels* argument.

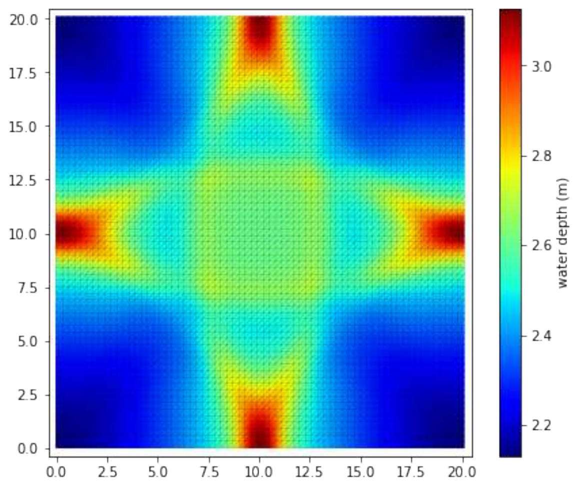


Figure 7: Example of scalar map plot with the gouttedo test case of TELEMAC-2D

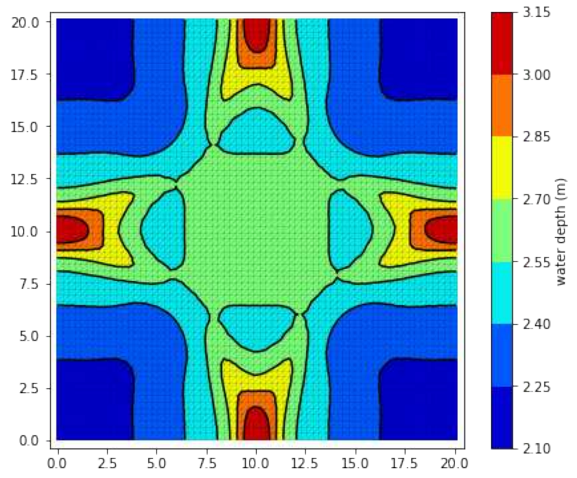


Figure 8: Example of scalar filled contours plot with the gouttedo test case of TELEMAC-2D

The last two functions in *plot2d* are *plot2d_vectors* and *plot2d_streamlines* and are used to visualize vectorial data like velocity, wind velocity or solid and liquid discharges. They take two scalars as inputs *data_x* and *data_y*, which correspond to *x* and *y* components of the vector. As for other 2D plots, *mesh* argument can either be a grid or a triangulation. But the user can also specify a grid resolution with the *grid_resolution* argument to make the functions generate their own grid. This functionality is useful when the triangular mesh is too refined and lead to a large amount of vectors being plotted. Vectors size are adjustable with the *scale* argument as well as streamlines density with the *density* argument. Vectors and streamlines are coloured depending on their norm by default but can also be coloured uniformly with the *color* argument. An example of coloured vectors plot is shown on *Figure 9*.

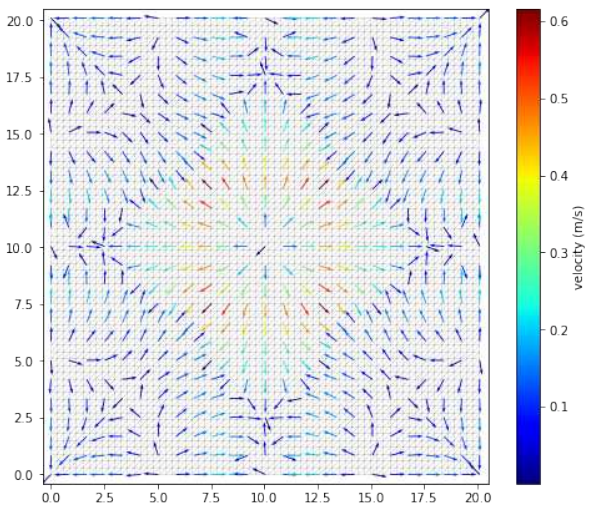


Figure 9: Example of vectors plot with the gouttedo test case of TELEMAC-2D

F. *How to do 2D plots with z elevation as y-axis*

TELEMAC-3D results can be visualized with 2D plots (x, y) presented previously if extraction is performed on a plane, defined at a fixed elevation or along a given computation plane. But 2D plots with elevation as y-axis could also be defined in combination with 2 possibilities for x-axis: curvilinear distance along a line or time.

The first example in Figure 10 displays velocity magnitude and shear velocity vectors in a user defined horizontal section. Although the vertical extraction (on the whole water column) is done along a straight line (y=10m), it could be performed along any arbitrary 2D polygonal chain.

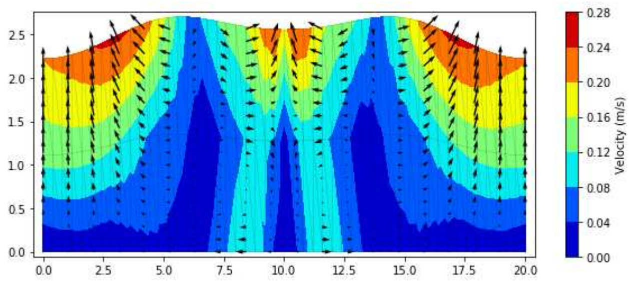


Figure 10: Example of 2D plot corresponding to a horizontal section (plane y=10m with gouttedo test case of TELEMAC-3D)

The alternative 2D plot with z-elevation is used to analyse the vertical distribution of a scalar over time. The Figure 11 presents vertical velocity distribution over time at the centre of the gouttedo domain (x=10m, y=10m).

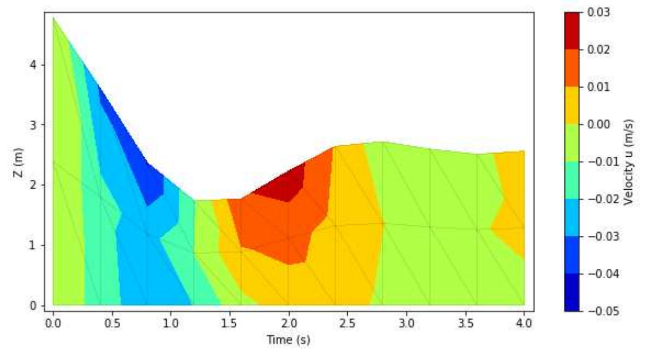


Figure 11: Example of a temporal 2D plot over the vertical with gouttedo test case of TELEMAC-3D

These 2D plots involving elevation are documented in the *3d_extraction* notebook.

G. How to do fancy 2D plots

The logic behind Postel plotting functions is that it allows us to pile up multiple layers. For instance, we can have a scalar map on top of a mesh with vectors and streamlines above it. The amount of layers is not restricted but data overlapping is the main limitation. To overcome this problem, alpha of each layer can be adjusted and mesh triangles can be masked. Masking can be done with a customizable criterion with the *mask_triangles* function. For example dry zones can be masked as on Figure 12. More details are presented in the notebook *example_plot2d_advanced.*

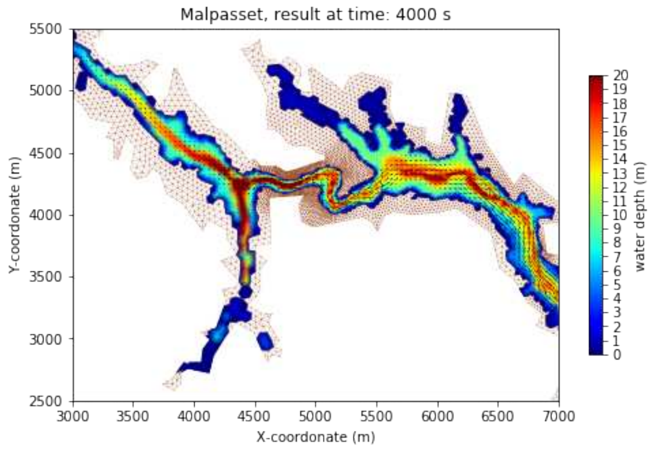


Figure 12: Example of advanced plotting with the Malpasset test case of TELEMAC-2D

In addition to basic 2D plots layers presented previously, images can also be added with the *plot2d_image* function. Images can be background map as presented in the Figure 13. Plotting background map often requires to change the coordinate system of the mesh. This can be done with the *pyproj* Python package. Moreover background image can be taken directly from a web map service using the *owslib* package. An example of how to use those packages is presented in *example_plot2d_background_map.*

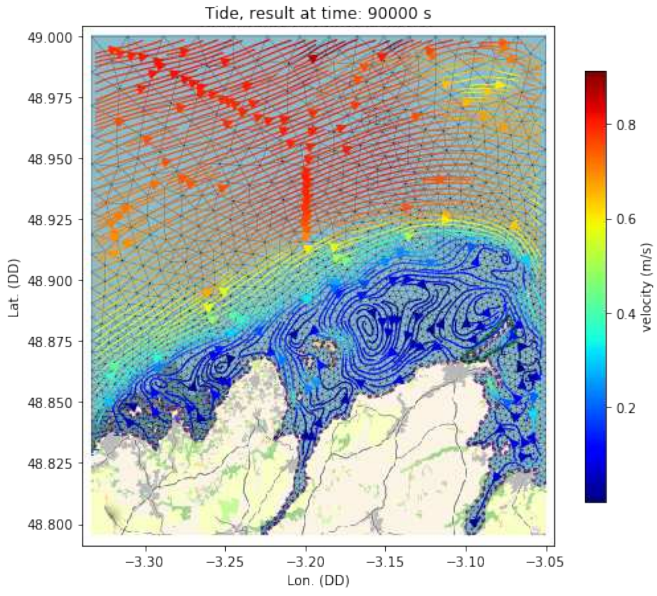


Figure 13: Example of background image plot with the tide test case of TELEMAC-2D

H. How to do 3d plots

3D plotting is limited in Python and only one function is available in Postel: *plot3d_scalar_map*. This allows us to visualize water depth as a 3D surface as presented in Figure 14.

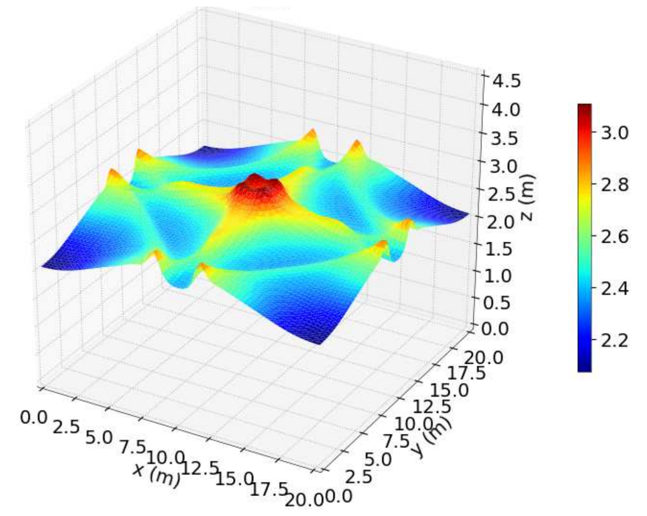


Figure 14: Example of 3D plot with the gouttedo test case of TELEMAC-2D

I. How to plot TOMAWAC spectrum files

Tomawac displays specific outputs, namely the spectrum, that need specific post-treatment. A user can specify several output locations over the domain, such that energy densities will be recorded for all discretised directions and frequencies. A couple of functions were added to get the spectrum graphic output. They are shown in the notebook *spectrum*.

First we added a method called *get_spectrum_name* in *TelemacFile* that gives you the name containing the spectrum associate to a point number given as a parameter. The function *get_list_spectrum_points* will give the list of points available in the file.

Figure 15 shows the integrated spectrum over the directions at a given point for a given record using *TelemacFile.get_spectrum.*

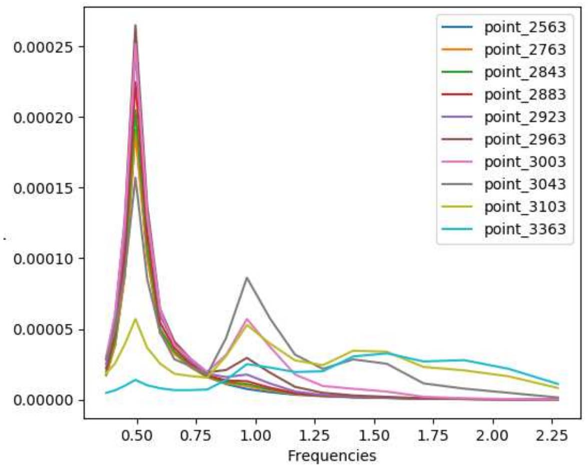


Figure 15: Plot of the TOMAWAC 1D spectrum for different output points

Using *TelemacFile.get_angular_dispersion* you can extract the energy integrated over the frequencies for a given point and record. The wave dispersion can be then observed. Figure 16:6 shows a polar projection example.

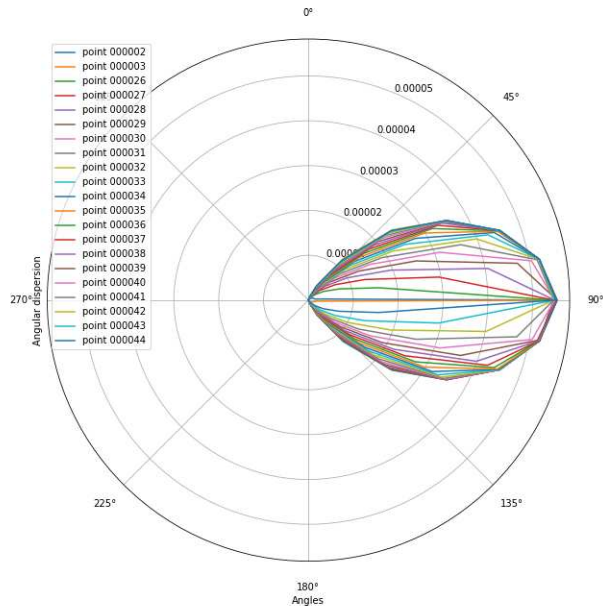


Figure 16: Plot of the energy density along wave directions of a TOMAWAC spectrum file for a list of points.

J. One line simple plots

It is possible to do some of the plots describes before using a one line bash command. This means you will have less control over what you can modify but it is a good way to make simple “on the go” plots.

This script is called *plot.py*. Run it with the option *–h* to see how it works. You can also have a look at the notebook *plot*.

V. Conclusion

We now have a nice Python module documented and easy to use. Containing all of the post-treatment we could do via the xml and more.

This will be available in the next release of TELEMAC-MASCARET (v8p1).

These scripts are bound to keep on evolving as they are used. Therefore contributions are more than welcome if you have scripts that could be integrated do not hesitate to contact the TELEMAC-MASCARET team.

Acknowledgements

Thanks for all the hard work of the Coding Week team that made the creation of this module possible.

References

[1] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open Source Scientific Tools for Python, 2001-, http://www.scipy.org/ [Online; accessed 2019-08-09].

[2] T. E, Oliphant. A guide to NumPy, USA: Trelgol Publishing, (2006).

[3] Hunter, J. D. Matplotlib: A 2D graphics environment, IEEE COMPUTER SOC Volume 9 Number 3 Pages 90-95 (2007)

[4] T. Kluyver, B. Ragan-Kelley, et al. Jupyter Notebooks - a publishing format for reproducible computational workflows, ELPUB, 2016

[5] G. van Rossum, B. Warsaw, N. Coghlan. PEPE8 -- Style Guide for Python Code, https://www.python.org/dev/peps/pep-0008, 2001

[6] Pylint, https://www.pylint.org/

[7] J-M. Hervouet, “Hydrodynamics of Free Surface Flows”, Wiley, 2007, pp. 83–130.

[8] G. van Rossum, Python tutorial, Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, May 1995.

[9] F. Pérez, B. E. Granger, IPython: A System for Interactive Scientific Computing, Computing in Science and Engineering, vol. 9, no. 3, pp. 21-29, May/June 2007, doi:10.1109/MCSE.2007.53. URL: https://ipython.org

[10] S. Bourban, J.-C. Parisi, A. Weisgerber, The TELEMAC’s automated management and continuous integration and validation system, Proceedings of the XXII Telemac User Conference, 2015, Daresburry (England).

[11] P. Prodanovic, A practical toolkit for terrain free surface flow and wave modelling, 2017, PPUTILS user documentation.

[12] L. Duron, Y. Wang, PyTelTools: Python scripts and GUI to automate Telemac post-processing tasks, 2017, Graz (Austria).