Goeury, Cédric; Audouin, Yoann; Zaoui, F.; Ata, Riadh; El Idrissi Essebtey, S.; Torossian, A.; Rouge, D.

Interoperability applications of TELEMAC-MASCARET System

Interoperability applications of TELEMAC-MASCARET System

Y. Audouin[1], C. Goeury[1], F. Zaoui[1], R. Ata[1], S. El Idrissi Essebtey[1], A. Torossian[1], D. Rouge[1]
[1]EDF R&D National Laboratory for Hydraulics and Environment (LNHE)
6 quai Watier, 78401 Chatou, France
Email: yoann.audouin@edf.fr

Abstract— **This paper is focused on the application of the recently implemented TelApy module (www.opentelemac.org). TelApy aims to provide a python wrapper of TELEMAC API (Application Program Interface). The goal of TelApy is to have a full control on the simulation while running a case. For example, it must allow the user to stop the simulation at any time step, get values of some variables and change them. In order to make this possible, a Fortran structure called instantiation was developed with the API. It contains a list of strings pointing to TELEMAC variables. This gives direct access to the physical memory of variables, and allows therefore to get and set their values. Furthermore, changes have been made in TELEMAC main subroutines to make hydraulic cases execution possible time step by time step. It is useful to drive the TELEMAC-MASCARET SYSTEM APIs using Python programming language. In fact, Python is a portable, dynamic, extensible, free language, which allows (without imposing) a modular approach and object oriented programming. In addition of benefits of this programming language, Python offers a large amounts of interoperable libraries. The link between various interoperable libraries with TELEMAC-MASCARET SYSTEM APIs allows the creation of an ever more efficient computing chain able to more finely respond to various complex problems. Therefore, the TelApy module has the ambition to enable a new way of use for the TELEMAC-MASCARET system. In particular one can think about high performance computing for the calculation of uncertainties, optimization, code coupling and so on. The objectives of the paper are to present some examples of the TelApy module in the case of Uncertainty Quantification, Optimization, Reduced Order Model and Monitoring System.**

I. Introduction

This paper will give a short explanation of the new Python module TelApy which is used to control the APIs of TELEMAC-MASCARET SYSTEM. The TELEMAC-MASCARET SYSTEM APIs are developed in Fortran. However, it is relatively easy to use these Fortran routines directly in Python using the "f2py" tool of the python Scipy library [7]. This tool will make it possible to compile Fortran code such as it is accessible and usable in Python. This compilation step is directly integrated into the compilation of the TELEMAC-MASCARET SYSTEM and is thus transparent to the user. Moreover, in order to make the TelApy module more user friendly, a python wrapper has been developed to encapsulate and simplify the different API Python calls. This set of transformation constitutes the TelApy module.

The first section of this paper is dedicated to a short description of how the APIs and TelApy are working, then four applications of that module are described.

II. TelApy package

A. Fortran APIs

An API (Application Programming Interface) is a library allowing to control the execution of a program. Here is part of the definition from Wikipedia:

“In computer programming, an application programming interface (API) specifies a software component in terms of its operations, their inputs and outputs and underlying types. Its main purpose is to define a set of functionalities that are independent of their respective implementation, allowing both definition and implementation to vary without compromising each other.

In addition to accessing databases or computer hardware, such as hard disk drives or video cards, an API can be used to ease the work of programming graphical user interface components, to allow integration of new features into existing applications (a so-called "plug-in API"), or to share data between otherwise distinct applications. In practice, many times an API comes in the form of a library that includes specifications for routines, data structures, object classes, and variables.”

The API’s main goal is to have control on a simulation while running a case. For example, it must allow the user to stop the simulation at any time step, retrieve some variables values and change them. In order to make this possible, a Fortran structure called instance was developed in the API. This structure is described later on. The instance structure gives direct access to the physical memory of variables, and allows therefore the variable control. Furthermore, modifications have been made in TELEMAC-MASCARET SYSTEM main subroutines to make hydraulic cases execution possible time step by time step. All Fortran routines are available in the directory "api" of TELEMAC-MASCARET SYSTEM sources.

In the rest of the paper we will make reference to the TELEMAC-2D part of the API but it is also available for other modules (so far for SISYPHE and soon TELEMAC-3D)

An instance is a Fortran structure that gathers all the variables alterable by the API. The definition of the "instance" structure is made in a Fortran module dedicated to this purpose and is composed of:

- An indice defining the instance I.D.
- A string which can contain error messages.
- Some pointers to the concerned module variables. This is what makes it possible to manipulate the variables of the module by having a direct access to their memory location.

In addition to the instance definition, the module includes all routines needed to manipulate it (creation, deletion, and so on).

The way the instance is defined (pointers) allows manipulation of variables during the simulation. So, to get information on the variables the following set of functionalities has been implemented:

- get the list of variables reachable with the API.
- get the information on the variable :
    - type of a variable (integer, Boolean, real, string).
    - Access (read-only, read-write).
    - Number of dimensions.
    - …
- get the size of a variable for each of its dimensions.
- get/set the value of a variable for a given index.

The list of variables that can be accessed is given in Table 1. Not every variable within TELEMAC-2D is there. However adding a new variable is pretty easy, the 5 steps procedure is described in the TelApy user documentation [1].

MODEL.AT: Current time
MODEL.BCFILE: Boundary condition file name
MODEL.BND_TIDE: Option for tidal boundary conditions
MODEL.BOTTOMELEVATION: Level of the bottom
MODEL.CHESTR: Strikler on point
MODEL.FAIR: Fair on point
MODEL.COTE: Prescribed elevation value
MODEL.CPL_PERIOD: Coupling period with sisyphe
MODEL.DEBIT: Discharge on frontier
MODEL.DEBUG: Activating debug mode
MODEL.FLUX_BOUNDARIES: Flux at boundaries
MODEL.GEOMETRYFILE: Name of the geomery file
MODEL.METEOFILE: Name of the binary atmospheric file
MODEL.FO2FILE: Name of the formatted data file 2
MODEL.LIQBCFILE: Name of the liquid boundaries file
MODEL.GRAPH_PERIOD: Graphical output period
MODEL.HBOR: Boundary value on h for each boundary point
MODEL.IKLE: Connectivity table between element and nodes
MODEL.INCWATERDEPTH: Increase in the the depth of the water
MODEL.KP1BOR: Points following and preceding a boundary point
MODEL.LIHBOR: Boundary type on h for each boundary point
MODEL.LISTIN_PERIOD: Listing output period
MODEL.LIUBOR: Boundary type on u for each boundary point
MODEL.LIVBOR: Boundary type on v for each boundary point
MODEL.LT: Current time step
MODEL.COMPLEO: Graphic output counter
MODEL.NBOR: Global number of boundary points
MODEL.NELEM: Number of element in the mesh
MODEL.NELMAX: Maximum number of elements envisaged
MODEL.NPOIN: Number of point in the mesh
MODEL.NPTFR: Number of boundary points
MODEL.NTIMESTEPS: Number of time steps
MODEL.NUMLIQ: Liquid boundary numbers
MODEL.POROSITY: Porosity
MODEL.RESULTFILE: Name of the result file
MODEL.SEALEVEL: Coefficient to calibrate sea level
MODEL.TIDALRANGE: Coefficient to calibrate tidal range
MODEL.UBOR: Boundary value on u for each boundary point
MODEL.VBOR: Boundary value on v for each boundary point
MODEL.VELOCITYU: Velocity on u
MODEL.VELOCITYV: Velocity on v
MODEL.WATERDEPTH: Depth of the water
MODEL.X: X coordinates for each point of the mesh
MODEL.XNEBOR: Normal X to 1d boundary points
MODEL.Y: Y coordinates for each point of the mesh
MODEL.YNEBOR: Normal Y to 1d boundary points
MODEL.EQUATION: Name of the equation used

TABLE 1. ACCESSIBLE VARIABLES THROUGH THE API.

The computation control is carried out using some specific routines to launch the simulation. These actions constitute a decomposition of the main function of each TELEMAC-MASCARET modules considered corresponding to the following different computation steps:

- **Configuration setup.** This function initialises the instance and the output. The instance, characterised by the *ID* integer parameter, represents a run of TELEMAC-2D.
- **Reading the steering file.** This function reads the case file and set the variables of the TELEMAC-2D steering file accordingly.
- **Memory allocation.** This function runs the allocation of all the data needed in TELEMAC-2D. Any modifications to quantities of TELEMAC-2D should be done before the call to that function.
- **Initialization.** This function will do the setting of the initial conditions of TELEMAC-2D. It corresponds to the time-step 0 of a TELEMAC-2D run.
- Computation function that runs **one time-step** of TELEMAC -2D. To compute all time steps, a loop on this function must be done.
- **Finalization.** This function concludes the run of TELEMAC -2D and will delete the instance. To start a new execution of TELEMAC-2D the configuration step must be run again.

For each action defined above, the identity number of the instance is used as an input argument allowing all computation variables to be linked with the corresponding instance pointers. These actions must be done in that particular order to insure a proper execution of the computation in the API main program.

The API being entirely in Fortran the preparation for a parallel run that is done by the Python script in a standard TELEMAC-2D run is not done. The partitioning and merging can be done using the API with the "partel" and "gretel" functions but the copying of files must be done elsewhere.

Fig. 1 presents the test case "gouttedo" written with the Fortran API.

```
PROGRAM HOMERE_API
  USE API_INTERFACE
  INTEGER :: ID, LU, LNG, COMM, IERR, VAR_SIZE, I
  INTEGER :: NTIME_STEPS
  CHARACTER(LEN=144) :: CAS_FILE = 't2d_gouttedo.cas'
  CHARACTER(LEN=144) :: DICO_FILE = 'telemac2d.dico'
  CHARACTER(LEN=144) :: DICO_FILE = 'r2d_test.slf'
  …
  CALL RUN_SET_CONFIG_T2D(ID,LU,LNG,COMM,IERR)
  CALL RUN_READ_CASE_T2D(ID,CAS_FILE,&
  DICO_FILE,IERR)
  ! Changing the name of the result file
  VARNAME = 'MODEL.RESULTFILE'
  CALL GET_VAR_SIZE_T2D(ID,VARNAME,VAR_SIZE,&
                        IDUM,IDUM,IERR)
  CALL SET_STRING_T2D(ID,VARNAME,RES_FILE,&
                      VAR_SIZE,0,0,IERR)
  CALL RUN_ALLOCATION_T2D(ID,IERR)
  CALL RUN_INIT_T2D(ID,IERR)
  !Get the number of timesteps
  VARNAME = 'MODEL.NTIMESTEPS'
  CALL GET_INTEGER_T2D(ID, VARNAME, &
                       NTIME_STEPS,&
                       0, 0, 0, IERR)
  DO I=1,NTIME_STEPS
     CALL RUN_TIMESTEP_T2D(ID,IERR)
  ENDDO
  CALL RUN_FINALIZE_T2D(ID,IERR)
END PROGRAM
```

Figure 1. *Example of a simple run of the gouttedo test case in Fortran*

B. Python package

It is relatively easy to use the Fortran API routines directly in Python using the "f2py" tool of the python Scipy library. This tool will make it possible to compile Fortran code such as it is accessible and usable in Python. For more details on this tool, the interested reader can refer directly to [7]. However, based on the advantage of the python language, it is possible to implement a wrapper in order to provide user friendly function of the Fortran API. Thus, a python overlay was developed in order to encapsulate and simplify the different API Python calls. The different python functions written to simplify the use of API are available in the directory "TelApy" withing the TELEMAC-MASCARET scripts folder. A doxygen documentation is available and allows the user to visualize python classes, functions that can be used as well as its input and output variables and so on.

The package follows the structure below:

- api
  - api_module: The generic python class for TELEMAC-MASCARET SYSTEM APIs.
  - generate_study: Automatic generator of a template for a run using the API.
  - hermes: Input and Output library allowing to write and read different mesh formats. Wrapper on the Fortran module.
  - sis: The SISYPHE python class for APIs.
  - t2d: The TELEMAC-2D python class for APIs.
- tools
  - genop: Optimization tool based on genetic algorithms.
  - newop: Optimization tool based on the SciPy minimizer function.
  - polygon: Function allowing to give point indices which are in a polygon defined by the user.

Fig. 2 displays all accessible classes of TelApy package.

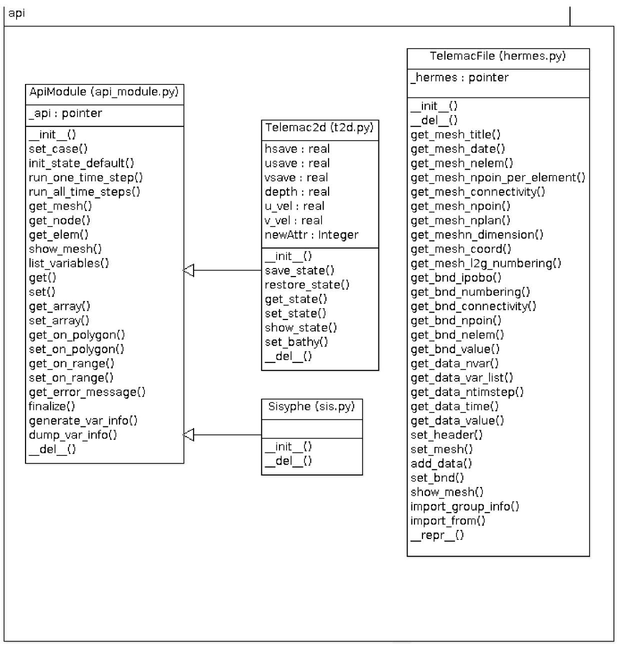


Figure 2. *Unified Modelling Language (ULM) of the classes in TelApy*

Fig. 3 presents the python scripting of the previous presented test case "gouttedo".

```
#!/usr/bin/env python
# Class Telemac2d import
from TelApy.api.t2d import Telemac2d
import shutil
# Creation of the instance Telemac3d
t2d = Telemac2d("t2d_gouttedo.cas")
# Running partel
t2d.set_case()
# Initalization
t2d.set("MODEL.RESULTFILE","r2d_test.slf")
t2d.init_state_default()
# Run all time steps
t2d.run_all_time_steps()
# Running gretel
t2d.finalize()
# Instance delete
del(t2d)
```

Figure 3. *Example of a simple run of the "gouttedo" test case in Python*

III. Applications

The section below will describe different type of application of the new module TelApy.

A. *Uncertainties Quantifications*

Here we will describe two works done using TelApy and the software OpenTurns[4] to run a MonteCarlo study.

TelApy was used to create what will be called the study function in the form:

$$Y = f(X) \qquad (1)$$

Where X is the list of the input uncertain variables and Y the interest variable. In this study, the interest variable is the water depth in all computational domain.

1) *Study area*

The area chosen for this study extends over a reach of the Garonne river measuring about 50 km, between Tonneins (upstream), downstream of the confluence with the Lot river, and La Réole (downstream) (see Fig. 4).

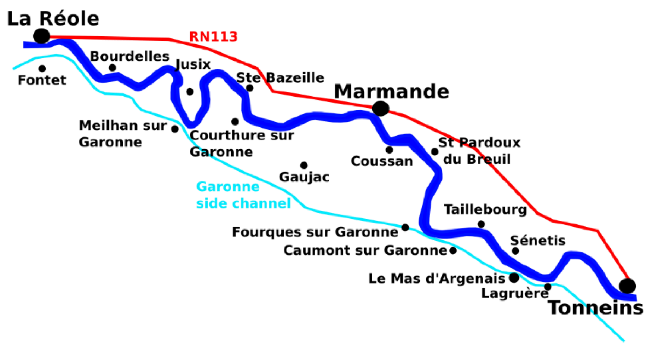


Figure 4. *Study area of the Garonne*

This part of the valley was equipped in the 19th century with infrastructure to protect against floods of the Garonne river which had heavily impacted local residents. A system of longitudinal dykes and weirs was progressively built after that flood event to protect the floodplains, organize submersion and flood retention areas. This configuration is also similar to the characteristic of other managed rivers such as the Rhone and the Loire.

2) *The hydraulic model*

The TELEMAC-2D model, constituted by a triangular mesh of some 41,000 nodes with an extremely small mesh size around the dykes, has a constant discharge upstream imposed at Tonneins and downstream, a stage-discharge relationship corresponding to the stream gauge at La Réole. This model has been realized by Besnard and Goutal (2008) [1].

In this study, we investigate the effect of two uncertainty sources on water level calculation for extreme flood event, the roughness coefficient and the upstream discharge. In fact, the hydraulic roughness is uncertain because flow measures are not available or reliable for calibration and validation. Discharge is also uncertain because it results from extrapolation of discharge frequency curves at very low exceeding probabilities. Five different zones were defined for the roughness and two discharges. The figures below show the probability density functions that will be applied for the study.

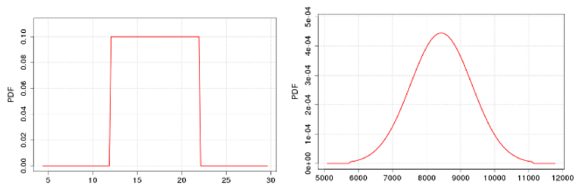


Figure 5. *Probability density function of the Strickler coefficient in the floodplain (left) and the upstream discharge*

3) *Results*

To handle the uncertainty with the Monte Carlo technique, it is important to run a lot of simulations in order to have reliable results. In this work, around 70,000 Monte Carlo computations have been carried out. EDF's cluster has been used to run these simulations. MPI library was used for launching and managing the uncertainty quantification study. Post-processing of the huge amount of results files is tackled through some Python scripts specifically developed within OpenTURNS.

The obtained results are analyzed twofold: On one hand, the effect of variability of random inputs is assessed at some specific points (assumed to be around an industrial plant, for example). On the other hand, a global statistical analysis all over the domain is done, as shown in Fig. 6.

A spatial distribution of the mean water depth and its variance is obtained. These results are of utmost importance for dimensioning of protecting dykes. Furthermore, there are very useful when establishing scenarios for flood managing.

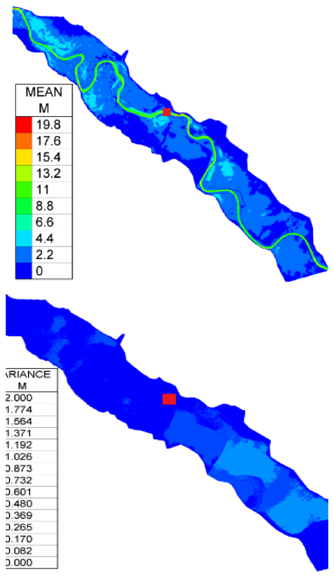


Figure 6. *Mean and variance all other the domain (🟥 node 37 242)*

To be sure that the obtained results are reliable, it is important to verify the convergence of them, especially by plotting the graph of the dispersion coefficient (σ/μ) as a function of *N*: if the convergence is not visible, it is necessary to increase *N* or if needed to choose another propagation method to estimate the uncertainty [8].

Fig. 7 shows the convergence of the dispersion coefficient and the mean of the water depth at the node number 37 242 located on Fig. 6.

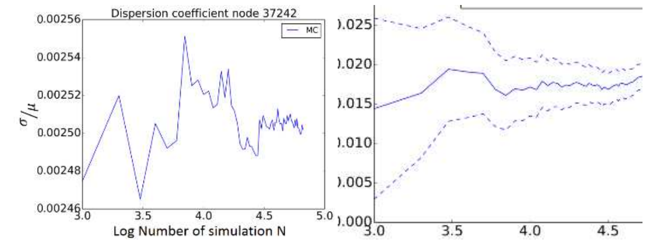


Figure 7. *Convergence graphs of the dispersion coefficient and the mean according to the logarithm of the number of simulations*

These graphics show that the convergence of results are guaranteed from 30,000 simulations of Monte Carlo Technique. These results are then used to provide reference statistical estimators in the comparison of the efficiency of the Monte Carlo-like methods.

For a more detailed explanation of the study you can read the article written by C. Goeury, et *al* [5].

B. *Automatic Calibration*

Two sensitive tidal parameters concerning the inflow conditions of a maritime case are optimized to better model the sea behaviour. The case is the Alderney race as described in [6].

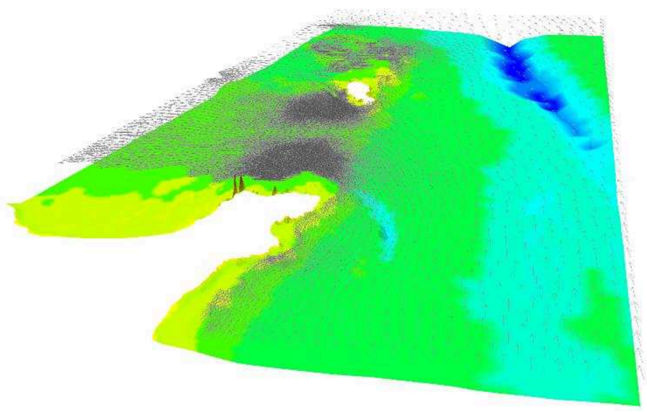


Figure 8. *Bottom and a velocity field in the strait between the Alderney Channel Island and Cap de La Hague*

The problem is mathematically formulated as given by (2). Here the boundaries are [-50, 50] for both parameters.

$$\begin{aligned} &\min_{x\in\mathbb{R}^2} f(x) \\ &s.t.\ \underline{x} \le x \le \overline{x} \end{aligned} \tag{2}$$

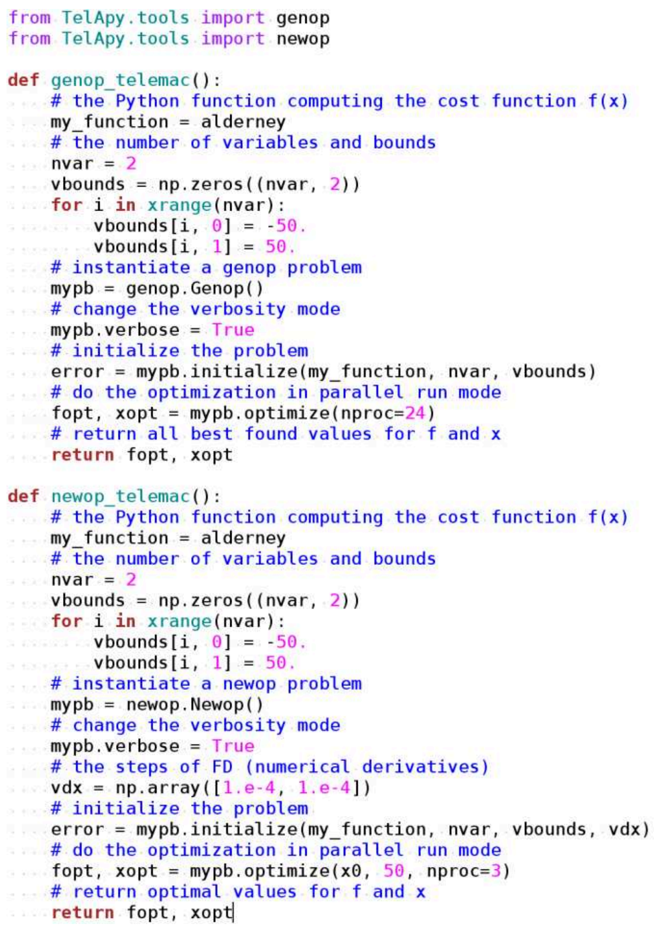
Where x-components are tidal parameters and stand for the difference between the Telemac results and measurements:

$$\begin{aligned} f(x) &= \|y(x)\|_2 \\ y(x) &= \left(T_1(x) - M_1, T_2(x) - M_2, \dots, T_p(x) - M_p\right) \end{aligned} \tag{3}$$

With T the Telemac operator, M the measurement for the water level and velocity, and p the number of time steps.

(2) is numerically solved with the help of the TelApy tools newop and genop for comparison. An important difference between these tools is that one requires derivative values and the other is a derivative-free optimizer. For the actual version of newop, derivatives are estimated only by numerical differentiation. Computations for successive evaluations of $f(x)$ or $\nabla f(x)$ are done automatically in parallel mode using the Python module *multiprocessing*.

As an example, Fig. 9 shows how using the TelApy tools on (2) is straightforward. The convergence of genop is presented in the Fig. 10 for a maximum number of 20 iterations. At the end of the convergence, 3,500 calls to TELEMAC-2D have been necessary. This large number can be easily divided by two with a stopping criteria of 10 iterations. Moreover, using a surrogate model for faster simulations, computational costs could also be reduced.



```python
from TelApy.tools import genop
from TelApy.tools import newop

def genop_telemac():
    # the Python function computing the cost function f(x)
    my_function = alderney
    # the number of variables and bounds
    nvar = 2
    vbounds = np.zeros((nvar, 2))
    for i in xrange(nvar):
        vbounds[i, 0] = -50.
        vbounds[i, 1] = 50.
    # instantiate a genop problem
    mypb = genop.Genop()
    # change the verbosity mode
    mypb.verbose = True
    # initialize the problem
    error = mypb.initialize(my_function, nvar, vbounds)
    # do the optimization in parallel run mode
    fopt, xopt = mypb.optimize(nproc=24)
    # return all best found values for f and x
    return fopt, xopt

def newop_telemac():
    # the Python function computing the cost function f(x)
    my_function = alderney
    # the number of variables and bounds
    nvar = 2
    vbounds = np.zeros((nvar, 2))
    for i in xrange(nvar):
        vbounds[i, 0] = -50.
        vbounds[i, 1] = 50.
    # instantiate a newop problem
    mypb = newop.Newop()
    # change the verbosity mode
    mypb.verbose = True
    # the steps of FD (numerical derivatives)
    vdx = np.array([1.e-4, 1.e-4])
    # initialize the problem
    error = mypb.initialize(my_function, nvar, vbounds, vdx)
    # do the optimization in parallel run mode
    fopt, xopt = mypb.optimize(x0, 50, nproc=3)
    # return optimal values for f and x
    return fopt, xopt
```

Figure 9. *Example of an automatic calibration using TelApy on the Alderney TELEMAC-2D case*

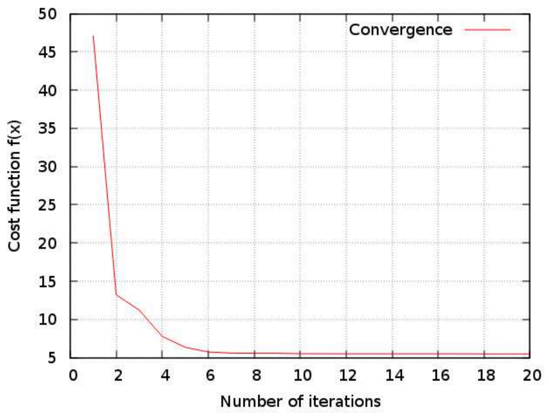


Figure 10. *Convergence of genop.*

C. Surrogate model

Many problems in the sciences and engineering require the determination of an unknown field from a finite set of indirect measurements. Examples include hydrology, oceanography, weather forecasts and hydraulic. In fact, numerical models are nowadays commonly used in fluvial and maritime hydraulics as prevention tools for example. Parameter estimation, also called inverse problem, consists of retrieving data of a problem from its solution. For free surface flow hydraulics, it involves assessing hidden or difficult-to-access parameters, such as bathymetry, bed friction, inflow discharge, tidal parameter, initial state and so on. In this work, an optimization process has been carried out to find initial state of a TELEMAC-2D computation. The fidelity of the optimized initial state is investigated with numerically generated synthetic data from so-called "identical-twin-experiments", in which true state is known. Thus, the test case "gouttedo" is studied in this work. This TELEMAC-2D model simulates the circular spreading of a wave. The domain is square with a size of 20.1 m x 20.1 m with a flat bottom. It is meshed with 8,978 triangular elements and 4,624 nodes. Triangles are obtained by dividing rectangular elements on their diagonals. The mean size of obtained triangles is about 0.3 m. The boundary conditions of the model are considered as solid walls with perfect slip conditions. The observation data used to retrieve the initial state are water depth synthetic data generated numerically. The water is initially at rest with a Gaussian free surface in the centre of a square domain such as the water depth is given by (4).

$$h_0^t(x) = 2.4e^{-\frac{[(x-10)^2+(y-10)^2]}{4}} \qquad (4)$$

So, the objective of this test case is to recover this initial state h_0^t constituent the true state using an optimization chain. The observations considered in this case are the water depth on all mesh nodes at different times in second $T = \{0.4; 0.8; \dots; 3.6; 4\}$. The initial guess of the initial state is set to a constant water depth of 2.4 m on each computational nodes. This inverse problem is solved computing variational data assimilation algorithm 3D-VAR. However, this algorithm consumes an important CPU time due to the large dimension of the system and the need for running it several times during the minimization of the cost function. Hence, we conclude the need to reduce the dimension of the initial model in order to alleviate the computational cost of the data assimilation process. The idea of order reduction is to search an optimal increment, not in the initial space of large dimension (dimension of nodes number), but in a space of reduced dimension. More precisely, we will look for an initial state $h_0(x)$ in the form given by (5).

$$h_0\ (x) = h_0^b(x) + \sum_{i=1}^{r} \alpha_i \Phi_i \qquad (5)$$

Where, $h_0^b(x)$ is a fixed reference state, and $(\Phi_1, \dots, \Phi_r)$ is the basis of the reduced space.
Then, the minimization process is done using the cost function $J\left(h_0\ (x)\right) = \tilde{J}(\alpha_1, \dots, \alpha_r)$.

Thus, the minimization takes place in a space of dimension r with $r \ll x$. The vector basis $(\Phi_1, \dots, \Phi_r)$ represents the modes of the system variability. The computation of this basis has been done in this work based on the classical Proper Orthogonal Decomposition method. By evaluating the energy captured by the proper vectors, only 7 modes are required to capture 99.9% of the total system variability. Thus, the truncation order of the approximation based of the proper orthogonal decomposition is fixed at $r = 7$. Generally, optimisation methods are used to solve minimisation problems. Many deterministic optimisation methods are known as gradient descent methods, among which the well-known BFGS quasi-Newton method, which is the approach used in this work. The inverse problem is solved in about 10 minimization iterations. Fig 8. displays the inverse problem results obtained when considering the minimization in the POD reduced space (green curve with circle markers) and in the initial space of large dimension (dimension of nodes number) (blue curve with square markers) over a slice along the axis x in the middle of the computational domain.

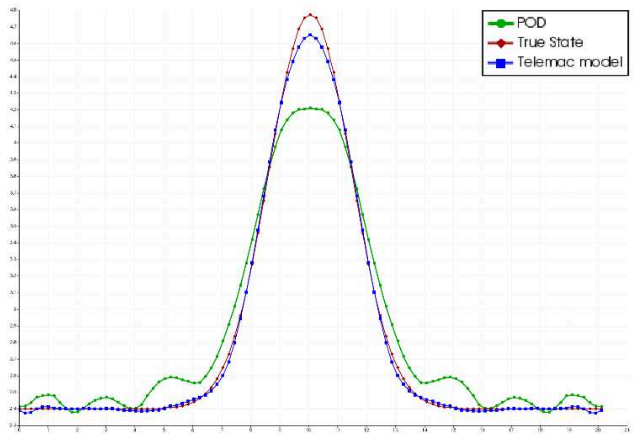


Figure 8. *Initial state results obtained by minimization process in the POD reduced space (green curve with circle markers) and in the initial space of large dimension (blue curve with square markers) over a slice along the x-axis in the middle of the computational domain. In comparison with the true state (red curve with diamonds)*

As shown in Fig. 8, the initial state obtained with the minimization in the initial space is close to the true state. Whereas the initial state obtained by solving the minimization in the reduced space presents more water depth oscillation, but the main behaviour is close to the true state. However, the computational cost is drastically reduced using the reduced space. In fact, the minimization process in the initial space takes about 5 days for 10 iterations. This is induced by the finite difference approximation of the observation operator adjoint. In the reduced space, this computational time is reduced to 17 minutes. This proves the efficiency of reduced order model when considering optimization in huge dimension problem.

D. *Monitoring tool*

The goal for that application was to create an autonomous software that will:

- Gather data to build the bathymetry of the model.
- Define the input parameters of the study.
- Run the TELEMAC-2D study.
- Display results informations.
- And a couple other functionalities such as automatic report generation, archives…

All these functionalities must be accessible through a user friendly GUI written in PyQt that could be used by someone not familiar with the TELEMAC-MASCARET system. The aim of this tool is to be able to control that the stream flows within the channels of the station are always fluvial. This taking into account the expected sedimentation and low water levels. If not, a dredging operation is required. So, the objective of this work is to provide a functional software able to analyse and forecast the water flow in order to anticipate the dredging operation. In order to evaluate this risk a simulation is ran to estimate the current number in the canal for the next months. Fig. 9 is an example of such a computation.

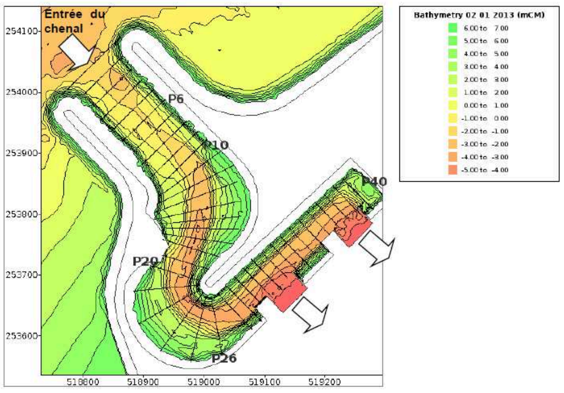


Figure 9. *Bathymetry estimation on the computational model*

TelApy is used here to have control on different computation parameters such as the date of the simulation, the sea level, the pump to take into account…

Fig. 10 displays a screenshot of the bathymetry built from the data available. Fig. 11 presents the GUI part where the study input parameters are defined. Finally, Fig. 12 displays simulation results of the Froude number on the computational model.

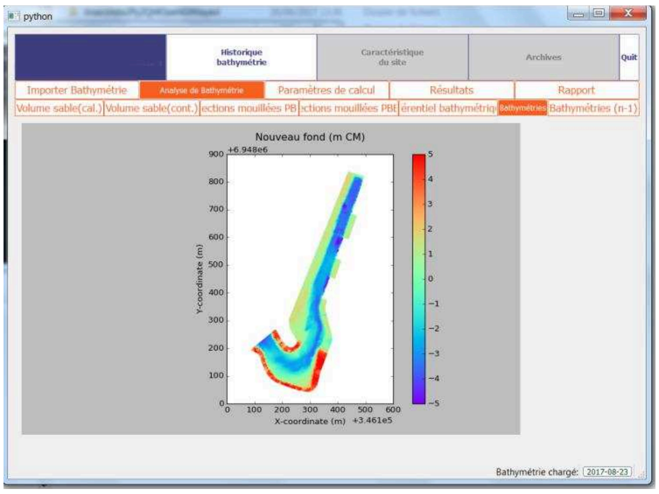
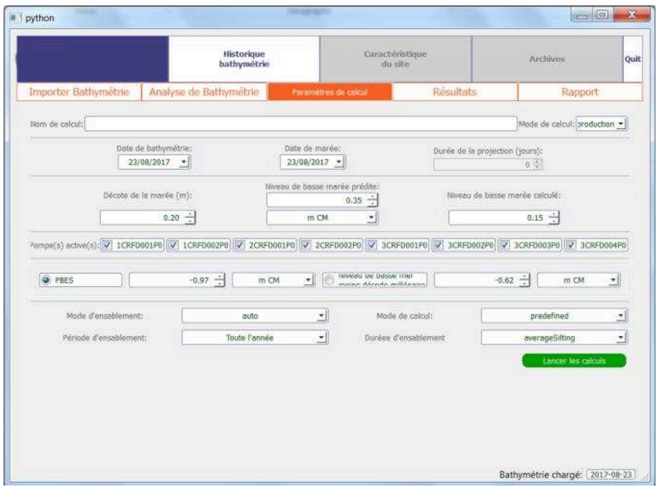


Figure 10. *Bathymetry on the model window*



Nom de calcul: Mode de calcul: production

Date de bathymétrie: 23/08/2017 Date de marée: 23/08/2017 Durée de la projection (jours): 0

Décote de la marée (m): 0.20 Niveau de basse marée prédite: 0.35 m CM Niveau de basse marée calculé: 0.15

Pompe(s) active(s): ✓ 1CRFD001P0 ✓ 1CRFD002P0 ✓ 2CRFD001P0 ✓ 2CRFD002P0 ✓ 3CRFD001P0 ✓ 3CRFD002P0 ✓ 3CRFD003P0 ✓ 3CRFD004P0

[✓ PBES] -0.97 m CM Niveau de basse mer moins décote millénaire -0.62 m CM

Mode d'ensablement: auto Mode de calcul: predefined

Période d'ensablement: Toute l'année Durée d'ensablement averageSilting

Bathymétrie chargé: 2017-08-23

Figure 11. *Input parameters window*

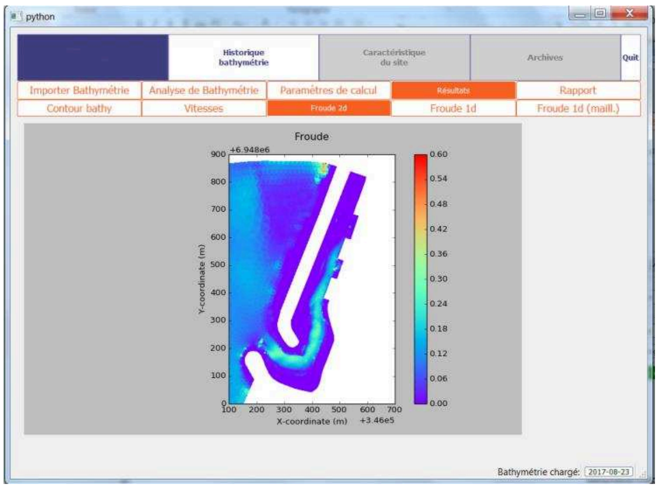


Figure 12. *Froude on the model window*

Conclusions and outlooks

Interoperability has become an important factor for codes to evolve and interact with others. With this module we are beginning to scratch the surface of what it is possible to do. We have here four applications in multiple domains: Uncertainty Quantification, Optimization, Reduced Order Model and Surveillance System. Interaction with outside software is now facilitate with TelApy.

The documentations (Doxygen and user documentation) and examples will be available in the v7p3 release of TELEMAC-MASCARET. Examples can be found within the source code in notebook format, a practical interactive format to manipulate Python scripts[a].

This new tool will allow us, in the future, to remove "user fortran" because all the modifications can be done directly via the APIs. Also we could rewrite the coupling between modules using the APIs. In the next versions of the TELEMAC-MASCARET SYSTEM more complex examples will be added to the ones already available in notebook format.

References

[1] A. Bernard and N. Goutal, "Comparison between 1D and 2D models for hydraulic modeling of a floodplain: case of Garonne river",proceedings of River Flow conference, 2008, in press.

[2] C. Goeury, Y.Audouin, F. Zaoui, "User documentation v7p3 of TelApy module", 2017

[3] J-M. Hervouet, "Hydrodynamics of Free Surface Flows", Wiley, 2007, pp. 83–130.

[4] EDF-EADS-PHIMECA, "Reference guide", OpenTURNS version 1.1, 2013.

[5] C. Goeury, T. David, R. Ata, S. Boyaval, Y. Audouin, N. Goutal, A-L. Popelin, M. Couplet, M. Baudin, R. Barate "Uncertatiny quantification on a real cas with TELEMAC-2D" Proceeding of the 2015 Telemac User Conference, October 2015

[6] C. Goeury, A. Ponçot, J.-P. Argaud, F. Zaoui, R. Ata, Y. Audouin, "Optimal calibration of Telemac-2D models based on a data assimilation algorithm", XXIVth Telemac & Mascaret User Club, Graz, Austria, October 2017

[7] PETERSON P. F2py: a tool for connecting fortran and python programs. International Journal of Computational Science and Engineering, 4(4):296–305, January 2009

[a]www.jupyter.org