Tsunami modelling: a link between partial differential equations and neural networks

Guillaume Steimer

M1 CSMI

UFR de Mathématiques et d'Informatique

Université de Strasbourg

Contents

1 Model and simulation **4**
- 1.1 Saint Venant equations . . . 4
- 1.2 1D finite volume method . . . 5
- 1.3 A well-balanced HLL scheme . . . 6

2 Numerical implementation **9**
- 2.1 Calculation script . . . 9
- 2.2 Validation . . . 10
  - 2.2.1 Lake at rest solution . . . 10
  - 2.2.2 The propagation a perturbation . . . 11
  - 2.2.3 Thacker test case for wet-dry transitions . . . 12
- 2.3 How to add some randomness . . . 13
  - 2.3.1 Add a random perturbation . . . 13
  - 2.3.2 Make random sea beds . . . 14
- 2.4 A numerical result . . . 15
- 2.5 Statistical insight . . . 17

3 About the project **18**
- 3.1 Road map . . . 18
  - 3.1.1 At the milestone of the V0 report (14 April) . . . 18
  - 3.1.2 At the milestone of the V1 report (22 May) . . . 19
  - 3.1.3 Between the V1 and the final report (28 May) . . . 19
- 3.2 Choices and resources . . . 19

4 Conclusion **20**

Introduction

All over the world, countries with maritime borders must prepare to face a common threat: tsunamis. From the Mediterranean Sea to the Pacific Ocean, every country must be able to react quickly to such a risk: within 15 minutes to an earthquake, every government must be able to predict whether there will be a tsunami and, if so, when the wave will hit the coast and how high it will be.

There are numerical models to simulate tsunamis, unfortunately these calculations take hours or days, so we should find another way to make good predictions.

In this semester project, we will try to demonstrate that a combination of one-dimensional partial differential equations (PDEs) and neural networks can produce tangible results in this type of problem.

Our objectives are to study and implement a one-dimensional scheme that reproduces the waves, taking into account the topography of the ocean floor. Then, we will validate it with test cases available in the article [1]. Finally, we will focus on the generation of random topographies. This project is a preparatory work that has to be done before the use of neural networks.

The project takes place in the MOCO (Modélisation et Contrôle) team, located at the IRMA (Institut de Recherche Mathématique Avancée) in Strasbourg. This team is specialised in PDEs analysis, control theory, scientific computing and high performance computing, statistics.

The project is supervised by Emmanuel Franck, Laurent Navoret and Vincent Vigon who are working on machine learning for simulation, numerical modelling... I would like to thank them for their guidance and help during this project.

1 Model and simulation

In this section, we will present the chosen model and develop a numerical scheme that preserves the well-balanced property based on finite volume methods.

1.1 Saint Venant equations

To describe the dynamic of tsunamis, we consider the Saint Venant equations, also called shallow-water equations to design this problem. Indeed, the ocean can be seen as a thin layer of water[1]: this is reasonable because the depth of the ocean ($\sim 1 - 10$ km) is small compared to the distance this type of wave travels ($\sim 100 - 1000$ km).

Let $h(t, x)$ be the water height and $u(t, x)$ the velocity which depend on time t and space x, $b(x)$ is a smooth topography and $g > 0$ refers to the gravitational acceleration. They satisfy the following system:

$$\begin{cases} \partial_t h + \partial_x(hu) = 0, \\ \partial_t(hu) + \partial_x\left(hu^2 + \frac{1}{2}gh^2\right) = -g\, h\partial_x b. \end{cases} \tag{1}$$

We will set $h_0(x)$ and $u_0(x)$ as initial conditions for the resolution.

If we consider a surface of water at rest, we have $u(x) = 0, \forall x$ and we obtain from equation (1):

$$\partial_t h = 0 \Rightarrow h(t, x) = h(x)$$
$$\partial_x\left(\frac{1}{2}gh^2\right) = gh\partial_x h = -gh\partial_x b \Rightarrow \partial_x(h + b) = 0 \Rightarrow b(x) + h(x) \text{ is constant}$$

Therefore, every couple (u, h) of this form:

$$\begin{cases} u(x) = 0, \\ b(x) + h(x) = c, \end{cases} \tag{2}$$

are usually called "steady states of the lake at rest" (indeed, without wind, a lake surface is plane). It is vital that our scheme preserve this type of solution (well-balanced property) as we will consider perturbations of this physical equilibrium to create simulations for our neural network. Otherwise, spontaneous waves could appear in the simulation and greatly disrupt the results.

[1] In fact, the shallow-water system can be derived from Navier-Stokes equations by integration in the vertical direction, see article [2].

1.2 1D finite volume method

Let's start by transforming our system into the standard form of a hyperbolic equation. Using (1), we obtain this equivalent form:

$$\partial_t w + \partial_x f(w) = r(w) \tag{3}$$

where $w = \begin{pmatrix} h \\ hu \end{pmatrix}$, $f(w) = \begin{pmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \end{pmatrix}$ is called the flux and $r(w) = \begin{pmatrix} 0 \\ -g(\partial_x b)h \end{pmatrix}$ the source term.

Then, we construct a uniform a mesh of $[0, L]$ space: we consider a set of points $(x_{i+\frac{1}{2}})_{i\in\{1,\dots,N\}}$ that form interfaces between different cells $(C_i)_{i\in\{1,\dots,N\}}$ such that $C_i = \left]x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}\right[$ with a constant space step $\Delta x_i = \Delta x = x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}}$. We note x_i the middle of C_i.

We also introduce a time step $\Delta t^n, n \in \mathbb{N}$ to define a sequence of intermediate times t^n with the relation $t^{n+1} = t^n + \Delta t^n$.

The main idea of the finite volume method (see article [3]) is to compute at each time step an approximation $(w_i^n)_i$ of the mean solution over each cell:

$$w_i^n \simeq \frac{1}{\Delta x_i}\int_{C_i} w(t^n, x)dx$$

and

$$w_i^0 = \frac{1}{\Delta x_i}\int_{C_i} w(t^0, x)dx \text{ where } w(t^0, x) = \begin{pmatrix} h_0 \\ h_0u_0 \end{pmatrix}(x)$$

As w is a solution of (3), we integrate (3) on a square $K := (t^n, t^{n+1}) \times C_i$:

$$\int_{C_i} w(t^{n+1}, x)dx - \int_{C_i} w(t^n, x)dx + \int_{t^n}^{t^{n+1}} f(w(t, x_{i+\frac{1}{2}}))dt - \int_{t^n}^{t^{n+1}} f(w(t, x_{i-\frac{1}{2}}))dt = \iint_K r(w)dtdx$$

We replace the integrals with their approximations:

$$\Delta x_i w_i^{n+1} - \Delta x_i w_i^n + \Delta t^n f_{i+\frac{1}{2}}^n - \Delta t^n f_{i-\frac{1}{2}}^n \simeq \Delta t^n \Delta x_i \begin{pmatrix} 0 \\ -g\{h\partial_x b\}_i \end{pmatrix}$$

where $f_{i+\frac{1}{2}}^n$ is the numerical flux:

$$f_{i+\frac{1}{2}}^n \simeq \frac{1}{\Delta t^n}\int_{t^n}^{t^{n+1}} f(w(t, x_{i+\frac{1}{2}}))dt,$$

and

$$\begin{pmatrix} 0 \\ -g\{h\partial_x b\}_i \end{pmatrix} \simeq \frac{1}{\Delta t^n \Delta x_i}\iint_K r(w)dtdx$$

is an approximation of the source term in the cell C_i.

Finally, to compute the sequence $(w_i^n)_{i\in\mathbb{Z},n\in\mathbb{N}}$, we will use an update formula:

$$w_i^{n+1} = w_i^n - \frac{\Delta t^n}{\Delta x_i}\left(f_{i+\frac{1}{2}}^n - f_{i-\frac{1}{2}}^n + \begin{pmatrix} 0 \\ g\Delta x_i\{h\partial_x b\}_i \end{pmatrix}\right) \tag{4}$$

We have to rewrite $f_{i+\frac{1}{2}}^n - f_{i-\frac{1}{2}}^n + \begin{pmatrix} 0 \\ g\Delta x_i\{h\partial_x b\}_i \end{pmatrix}$ in (4) in order to create two flows from the cells C_{i-1} and C_{i+1} named F^L and F^R such that (4) become:

$$w_i^{n+1} = w_i^n - \frac{\Delta t^n}{\Delta x_i}\left(F_{i+\frac{1}{2}}^L - F_{i-\frac{1}{2}}^R\right) \tag{5}$$

Such flows have to preserve the well-balanced property (with $u = 0$, the quantity $h + b$ must be preserved in each cell between to time steps) and the positivity of water heights.

1.3 A well-balanced HLL scheme

In the following, we will present a 1D finite volume scheme. It is called the Harten-Lax-van Leer (HLL) [1] scheme. A lot of numerical schemes struggles to preserve the well-balanced property, the positivity of water heights and the entropy-preserving properties at the same time. This scheme preserve the first two and a weak form of the third.

The objective is to define F^L and F^R.

The main principle of this scheme is to use an HLL solver to approximate a solution of the Riemann problem[2] at each interface $x_{i+\frac{1}{2}}$ throughout each time step. It preserves the positivity of water heights. With a correct approximation of the source term, we will preserve the well-balanced property and allow the program to treat wet-dry transitions.

Our goal is to define $F_{i+\frac{1}{2}}^L$ and $F_{i+\frac{1}{2}}^R$, the numerical flux between C_i and C_{i+1}. To achieve that, we introduce some quantities which will be used in our scheme. Let X be the height h, the velocity u or the topography b.

In a cell C_i, we note X_i the mean of X in C_i.

$$X_i = \frac{1}{\Delta x}\int_{C_i} X(x)dx$$

At the interface between C_i and C_{i+1}, we note X_L the mean of X in the left cell C_i and X_R the mean of X in the right cell C_{i+1}:

$$X_L = \frac{1}{\Delta x}\int_{C_i} X(x)dx \text{ and } X_R = \frac{1}{\Delta x}\int_{C_{i+1}} X(x)dx$$

For instance, $u_L = \frac{1}{\Delta x}\int_{-\Delta x}^0 u(x)dx$.

Those two notations describe the same thing but the point of view differ: in fact the flows will be computed on an interface, that is why we use the indexes L and R.

[2] An initial value problem with a piecewise constant initial data (we will approximate w with a constant in each cell) and a single discontinuity.

We define F_{HLL}, the HLL flow:

$$F_{HLL}(w_L, w_R) = \begin{cases} f(w_L) & \text{if } 0 < \lambda_L, \\ \dfrac{\lambda_R f(w_L) - \lambda_L f(w_R)}{\lambda_R - \lambda_L} + \dfrac{\lambda_L \lambda_R}{\lambda_R - \lambda_L}(w_R - w_L) & \text{if } \lambda_L < 0 < \lambda_R, \\ f(w_R) & \text{if } \lambda_R < 0. \end{cases}$$

where $\lambda_L = \min_{w=w_L,w_R}\left(u - \sqrt{gh}, 0\right)$, $\lambda_R = \max_{w=w_L,w_R}\left(u + \sqrt{gh}, 0\right)$ are the waves velocity of the approximate solution of the Riemann problem. We can remark that F_{HLL} is continuous: it can deal with a λ equal to 0. It will be used in (6).

Finally, we define the numerical flows F^L and F^R of (5):

$$\begin{cases} F^L_{i+\frac{1}{2}}(w_L, w_R, b_L, b_R) = F_{HLL}(w_L, w_R) + \begin{pmatrix} \dfrac{\lambda_L \lambda_R \Delta b}{\lambda_R - \lambda_L} \\ -\dfrac{\lambda_L g \Delta x \{h\partial_x b\}_{i+\frac{1}{2}}}{\lambda_R - \lambda_L} \end{pmatrix} \\ F^R_{i+\frac{1}{2}}(w_L, w_R, b_L, b_R) = F_{HLL}(w_L, w_R) + \begin{pmatrix} \dfrac{\lambda_L \lambda_R \Delta b}{\lambda_R - \lambda_L} \\ -\dfrac{\lambda_R g \Delta x \{h\partial_x b\}_{i+\frac{1}{2}}}{\lambda_R - \lambda_L} \end{pmatrix} \end{cases} \tag{6}$$

where $\Delta b = b_R - b_L$ and $\{h\partial_x b\}_{i+\frac{1}{2}}$ is an approximation of the source term the interface $x_{i+\frac{1}{2}}$:

$$\{h\partial_x b\}_{i+\frac{1}{2}} = \begin{cases} \dfrac{h_L + h_R}{2\Delta x}\min(h_L, \Delta b) & \text{if } \Delta b \geqslant 0 \\ \dfrac{h_L + h_R}{2\Delta x}\max(-h_R, \Delta b) & \text{if } \Delta b < 0 \end{cases}$$

Now, we can define the update formulas taken from (5) for our scheme:

$$\begin{cases} w_i^{n+1} = w_i^n - \dfrac{\Delta t^n}{\Delta x}\left(F^L_{i+\frac{1}{2}} - F^R_{i-\frac{1}{2}}\right) \\ w_i^0 = \dfrac{1}{\Delta x}\left(\int_{C_i} h_0(x)dx, \int_{C_i}(h_0 u_0)(x)dx\right)^T \end{cases} \tag{7}$$

Its Courant-Friedrichs-Lewy (CFL) condition is:

$$\Delta t < \frac{\Delta x}{2\max\limits_{C_i}(|\lambda_L|, |\lambda_R|)} \tag{8}$$

It's a condition for the stability of our scheme. If we note:

$$\text{CFL} = 2\max_{C_i}(|\lambda_L|, |\lambda_R|)\frac{\Delta t}{\Delta x} \tag{9}$$

We must have CFL < 1 and we obtain a way to calculate every Δt^n in order to ensure the stability of our scheme, using (8) and (9):

$$\Delta t^n = \frac{\text{CFL} \cdot \Delta x}{2 \max\limits_{C_i}(|\lambda_L|, |\lambda_R|)} \tag{10}$$

Then, to calculate means, we will approximate the integral of a function by its value in the centre of the domain of integration. It will result in an error of Δx^2.

A last important remark is that intermediates (during the calculation of flows) heights h_L, h_R can be negative. It is crucial to take that into account: looking at the remark 2.1 in the article [1]: we have to impose a threshold $\epsilon \ll 1$ (10^{-12} in practice) such that whenever $h_L < \epsilon$ (resp. h_R), we set $u_L = 0, h_L = 0$ (resp. $u_R = 0, h_R = 0$).

2 Numerical implementation

In this section, we will discuss two main points: the implementation of the numerical scheme (6)(7) in C++ and how to add randomness in the topography.

2.1 Calculation script

In figure 1, the program is described in a UML diagram:

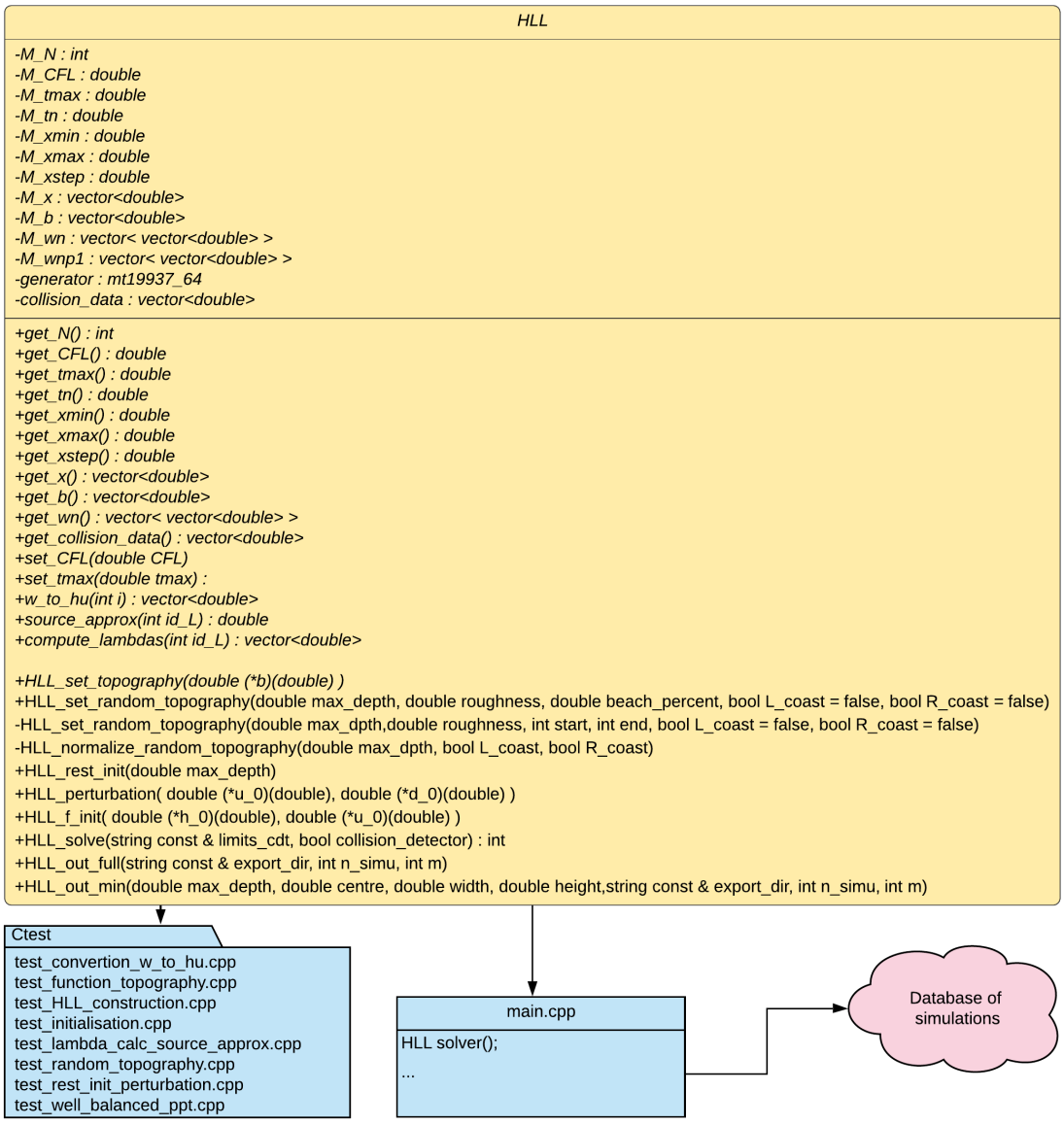

Figure 1: UML diagram of the calculation program

The program is based on the object-oriented dimension of C++. A more complete explanation can be found reading `README.md` in the repository.

2.2 Validation

We will validate our program through test cases, some of them are available in the article [1].

2.2.1 Lake at rest solution

We take "lake at rest" type initial conditions and we observe the well-balanced property. The result of figure 2 does not seem impressive nevertheless it shows that this scheme works. There are no spontaneous wave, $h + b$ remain constant at all time.

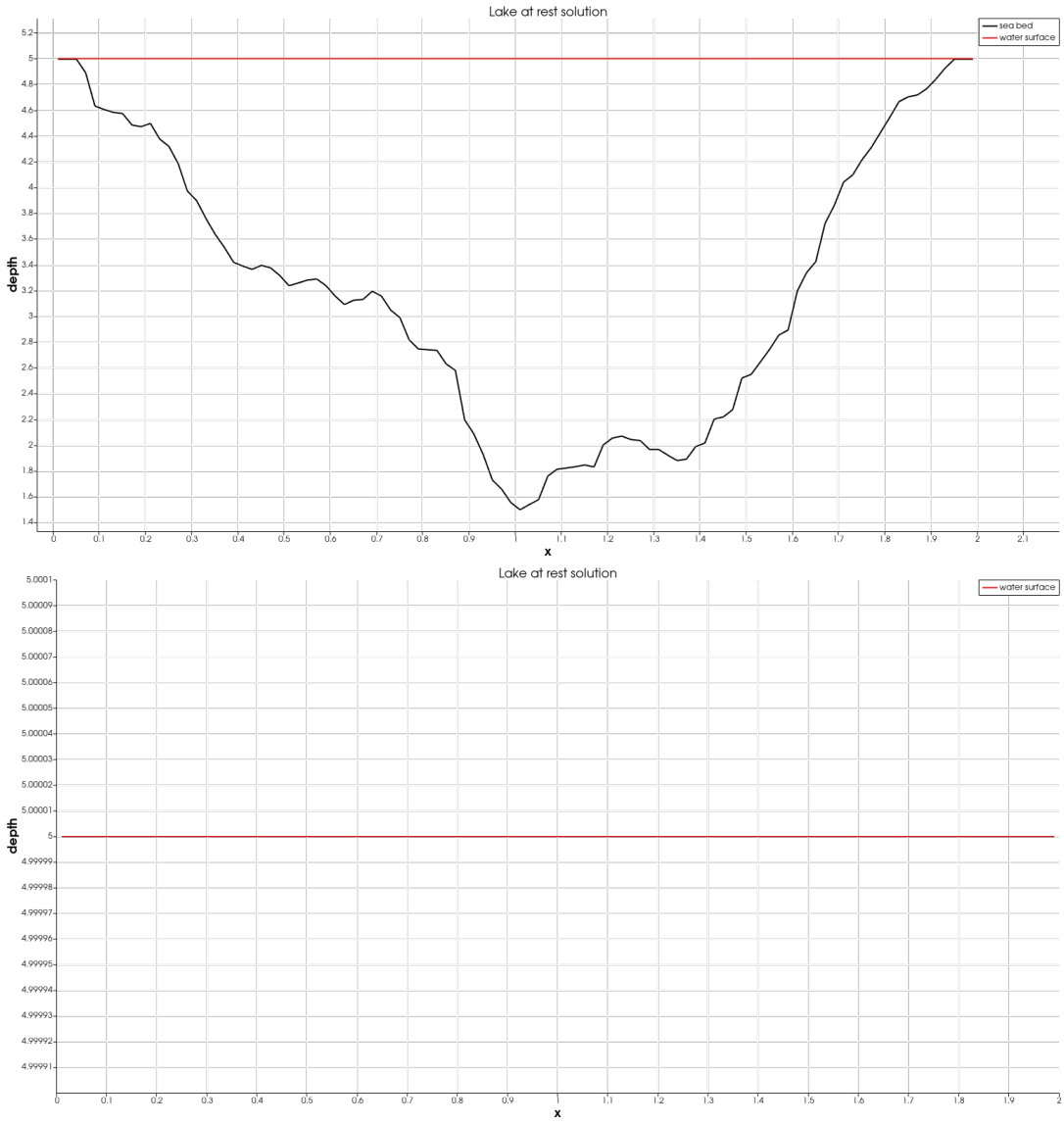

Figure 2: Lake at rest solution (the whole problem and a zoom on the water surface)

2.2.2 The propagation a perturbation

This simulation is based on the section 3.1 from the article [1]. We have the domain $[0, 2]$, $\Delta x = \frac{1}{40}$, a final time of $t = 0.2$, a CFL condition of 0.9 and the bottom topography is defined by:

$$b(x) = \begin{cases} 2 + 0.25(cos(10\pi(x - 0.5)) + 1) & \text{if } 1.4 < x < 1.6 \\ 2 & \text{otherwise.} \end{cases}$$

The initial conditions are $u_0(x) = 0$ and

$$h_0(x) = \begin{cases} 3 - b(x) + 0.001 & \text{if } 1.1 < x < 1.2 \\ 3 - b(x) & \text{otherwise.} \end{cases}$$

It gives us similar results on the graph 3:

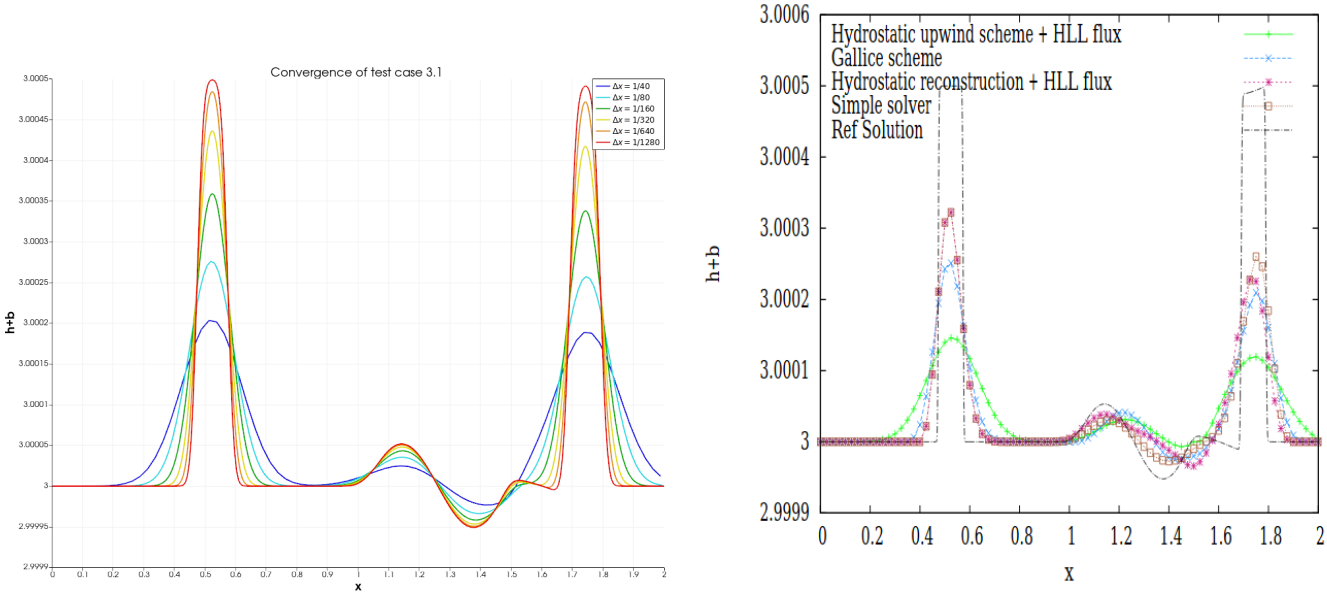

Figure 3: Case 3.1 in [1], our program (left) and the reference (right)

We can see the convergence of our scheme to the reference solution described in [1] and called "Ref Solution" in the previous figure.

2.2.3 Thacker test case for wet-dry transitions

This test is based on the section 3.3 from the article [1]. The scheme have to handle wet-dry transitions. The domain is $[-2, 2]$.

The bottom topography is defined by:

$$b(x) = \frac{x^2}{2} + \frac{3}{2}$$

The initial conditions are $u_0(x) = 0$ and:

$$h_0(x) = \begin{cases} -\frac{1}{2}\left(\left(x + \frac{1}{2}\right)^2 - 1\right) & \text{if } -\frac{3}{2} < x < \frac{1}{2} \\ 0 & \text{else.} \end{cases}$$

The final time is 16, $\Delta x = \frac{1}{80}$ and the CFL condition is 0.5. Lets compare our results to those of the article in the figure 4:

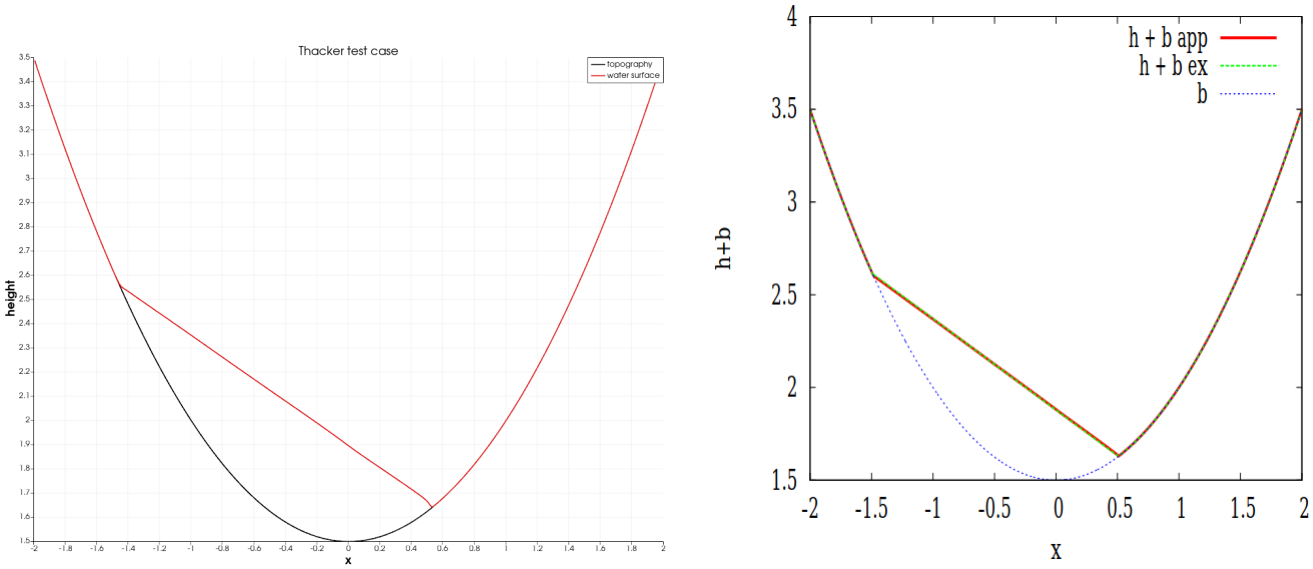

Figure 4: Case 3.3 in [1], our program (left) and the reference (right)

We obtain the same results as in the article. It is reassuring, indeed wet-dry transitions are managed and it should give proper results when the water depth is very low and the wave could create dry surfaces.

2.3 How to add some randomness

We can easily randomise parameters such as the length of the domain, the maximum depth... This part will explain how to set a random perturbation in the simulation and how to make random sea beds. Throughout the simulation, a pseudo random generator, called twister engine, ensure the generation of all the random variables.

2.3.1 Add a random perturbation

Starting from a lake at rest type solution, we add a window shape disorder with a random height, width and position with the method `HLL_perturbation`().

The typical form used is illustrated by the figure 5:

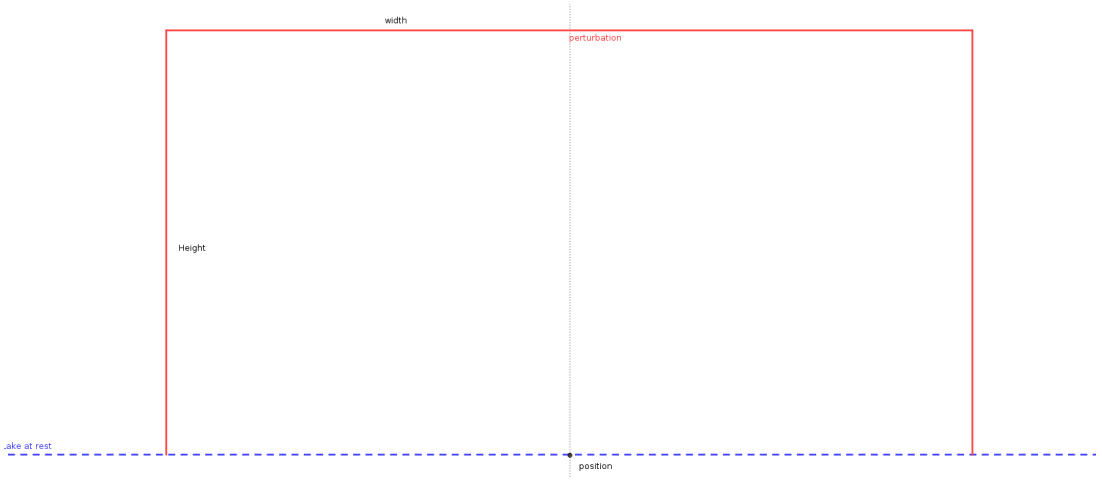

Figure 5: Typical perturbation from the lake at rest state

2.3.2 Make random sea beds

Generating random topographies is more complicated than generating disturbances. In order to accomplish that, a technique similar to the "midpoint displacement"[3] algorithm is used.

The main principle is simple: we are going to explain it using the figure 6.

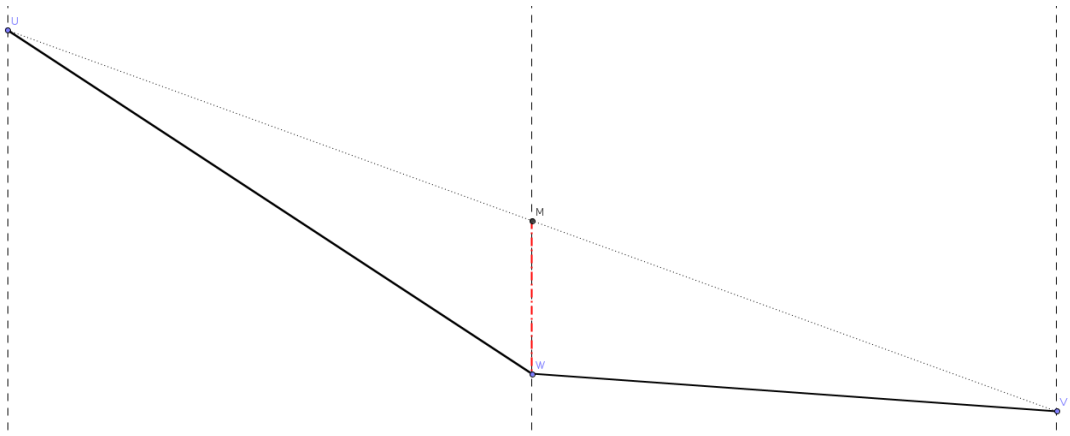

Figure 6: Midpoint displacement in 1D

We work an given interval $[U, V]$ of two points with a given abscissa and height. Then, we look for the middle of the segment: M. We keep the abscissa of M and add a random value within the range $[-a, a], a \in \mathbb{R}_+$ to its height: we obtain the point W.

We restart the algorithm using $[U, W]$, $[W, V]$ and a range of $[-\frac{a}{2}, \frac{a}{2}]$ for the random height. until the intervals are small enough. All the created segments form a random topography. We can make it more or less steep by choosing a bigger or smaller. Is the code, a in represented by the parameter `roughness` which is a percentage of the maximal depth.

In our situation, we already have a breakdown of the domain so we will use those values for the abscissa of the points considered. In practice, we will initialise this algorithm with coasts and beaches at the endpoints and we have to normalise our results so as not to exceed a maximum depth. The first midpoint will also be chosen differently. Here is a draft in the algorithm 1:

Algorithm 1: random_topography(*max depth*, *roughness*, *start*, *end*)

Data: A set of N abscissa in M_x
Result: A set of N heights in M_b

```
middle ← ( start + end ) / 2
M_b[ middle ] ← ( M_b[ start ] + M_b[ end ] ) / 2 + random
if middle − start > 1 then
    random_topography(max depth,roughness / 2,start,middle)
end
if end − middle > 1 then
    random_topography(max depth,roughness / 2,middle,end)
end
```

[3] Common technique used in the video-game industry to create random landscapes.

Here are a few examples on figure 7 for a maximum depth of 5, a domain length of 10 and 400 points and different steeps using the method `HLL_set_random_topography()`:

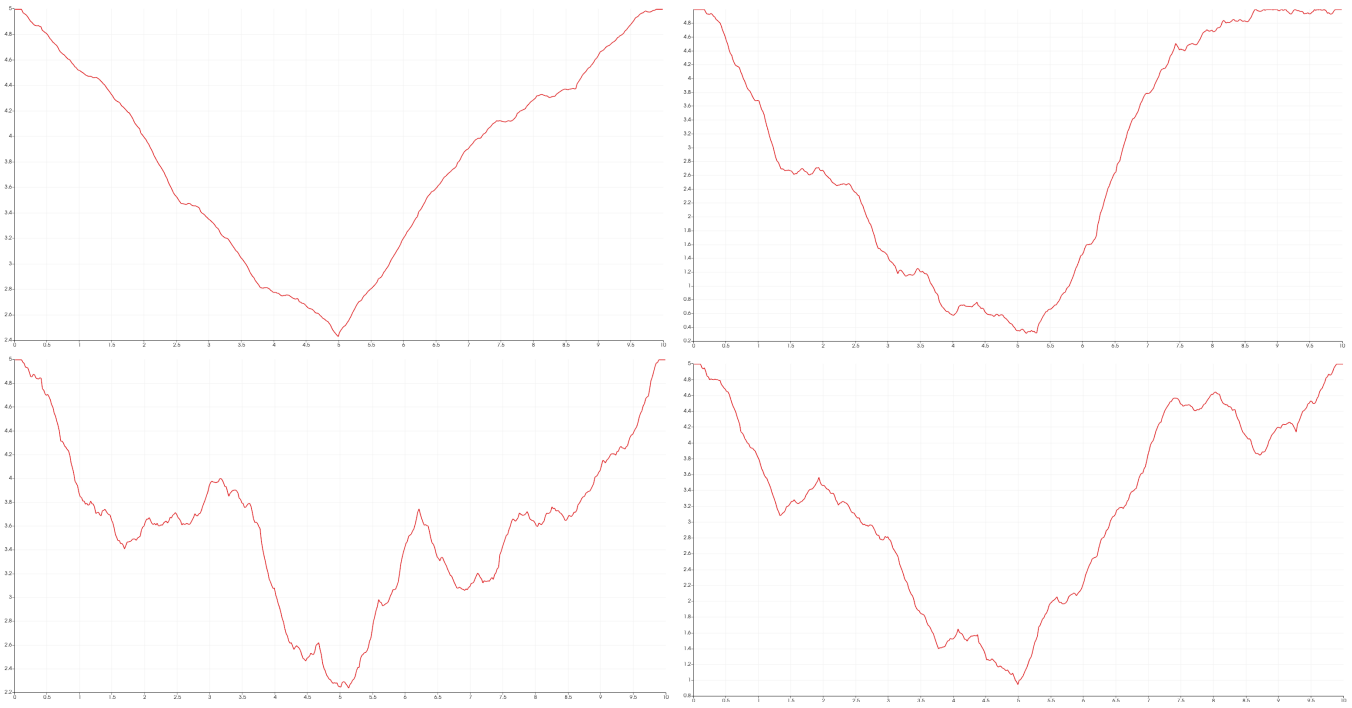

Figure 7: Random sea beds

2.4 A numerical result

We can observe a few phenomena on a simulation with a random topography: the height of the wave increase in shallow waters and its speed diminish. We create a decomposition of $[0, 4]$ with $\Delta x = \frac{1}{80}$, a CFL condition of 0.8 and a sufficient time to make our observations. We use a perturbation 0.001 tall and 0.1 wide centred of 1.65. The maximal depth is set to 4, the roughness to 50% and the beach proportion to 2%. In the figure 8, we can see the results of the simulation. Each dotted curve is a representation of the wave at regular time intervals ($\Delta t = 0.15$). We remark that the greater is the depth, the faster is the wave (which is normal because its speed is of the order of $\sqrt{gh}$). It is not obvious here, but in the simulation it is remarkable that a diminution in water depth tends to cause a rise in the wave height.

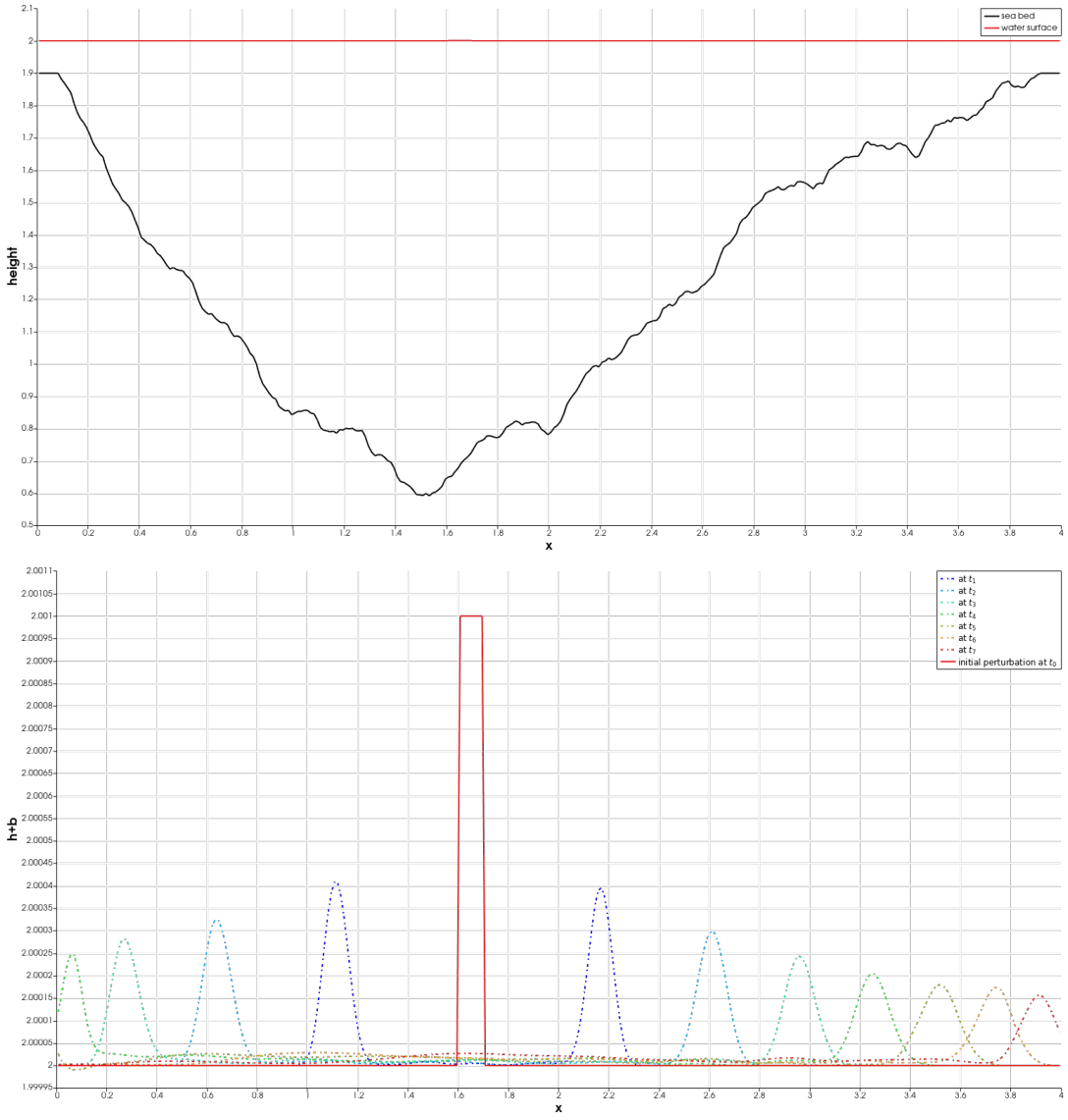

Figure 8: Random topography (left) and the propagation of a perturbation (right)

2.5 Statistical insight

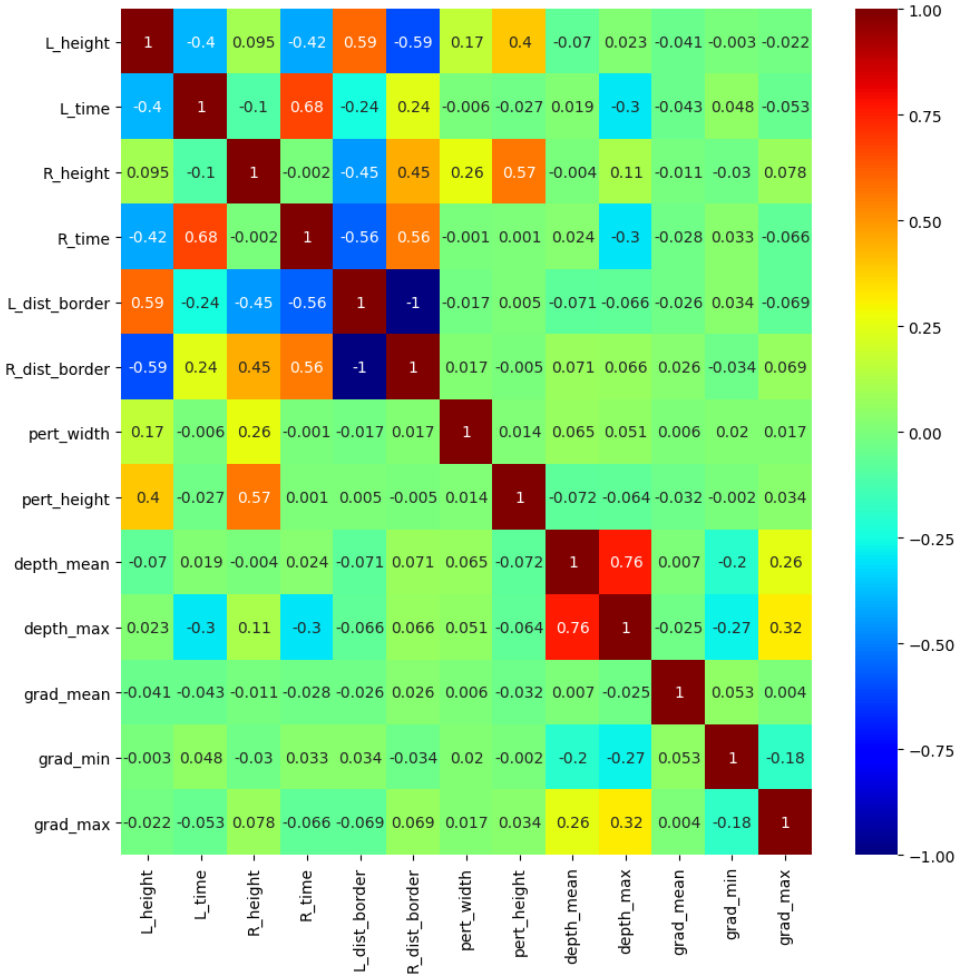

Figure 9: Correlation matrix with 1000 simulations

All the chosen parameters are set with a uniform distribution on the following intervals:

- $[0.0001, 0.001]$ for the perturbation height,
- $[0.1, 0.2]$ for the perturbation width,
- $[0.5, 0.6]$ for the roughness of the topography (I should have chosen a longer interval looking at the matrix).

The domain of simulation is $[0, 20]$, the max_depth is 1.0 and there are 2% of beaches at each border. The CFL is set on 0.8 and $\Delta x = \frac{1}{50}$. It will be necessary to refine these choices but this just gives an idea of the work to follow.

We remark that the height of the wave is highly correlated to the perturbation parameters (especially its height), its distance from the border and its arrival time.

3 About the project

3.1 Road map

3.1.1 At the milestone of the V0 report (14 April)

This is what has already been achieved:

1. get acquainted with the one-dimensional finite volume method ($\sim$ 2-3h),
2. reading the article about the HLL scheme [1] and other articles to be more familiar with the subject ($\sim$ 5-6h),
3. programming the scheme in C++ starting from scratch. ($\sim$ 10-11h),
4. V0 report redaction ($\sim$ 4-5h).

And the next steps are:

1. Test the code on basic cases and correct every issue($\sim$ 2-4h),
2. Run the code on benchmarks, test its efficiency ($\sim$ 1-2h),
3. Complete the code by adding different outputs, automate it($\sim$ 4-6h),
4. prepare data relatively to the AI problem ($\sim$ 3-4h),
5. continuous writing of the report and the presentation ($\sim$ 11-13h),

I already spent 21 to 25 hours building this project. In total, I would spend 42 to 54 hours on it. The deadlines are:

- 14 April: submit the first version of this report
- 1 May: have a functional script for the simulations
- 22 May: submit an almost finished report
- 29 May: final report and the PDF file of the presentation
- 30-31 May: defence of the project

My managers advised me to focus on producing a functional and efficient simulation program for the project and to make some descriptive statistics on the results. Then I will be able to focus on AI during my internship.

3.1.2 At the milestone of the V1 report (22 May)

At this time, I continue to write the report and improve the code. The random simulations took much longer than expected and there were problems with the wet/dry transitions that had to be resolved. Here is an overview of the work done since the previous phase:

1. correct every bug and a new output method ($\sim$ 8h)
2. add unitary test and validate the code with test cases ($\sim$ 5h)
3. learn about pseudo-random generators and add method to create correct random topographies ($\sim$ 6h)
4. write the report ($\sim$ 12h)
5. learn markdown basics and write the user manual ($\sim$ 3h)

I spent 55-59 hours on this project. I haven't had time to make statistical inferences using simulations. In fact, I preferred to focus on having a valid code with all the necessary features and to make the statistics during the internship.

3.1.3 Between the V1 and the final report (28 May)

1. finish the code automation ($\sim$ 1h)
2. write the Python script that read the output, create a data-frame and the correlation matrix ($\sim$ 2h)
3. write the slides for the final presentation ($\sim$ 14h)
4. write the report ($\sim$ 2h)

In total, I recorded 74-78 hours spent on this project (which is a raw estimation considering I didn't kept a schedule). I began an outline of the data work with Python in order to produce a correlation matrix. Everything is set for the beginning of the internship.

3.2 Choices and resources

I chose to write the simulation program in C++ for its speed and adaptability. It requires more work than Python, but it is worth it. I use VScode in conjunction with Github and a Docker environment to develop and run it. I rely on Cmake and Ctest to handle compilation and unit testing. For the moment, my personal computer seems to be sufficient to perform the simulations (it took around 2 seconds for one complete simulation with 1000 cells in a domain of $[0, 20]$ that run until collisions). A more powerful computer might be needed for the next part of the project, mainly to train the neural networks.

I'm writing my report on Overleaf, an online LaTeX editor.

Due to the COVID-19 pandemic, my supervisors and I heavily used Google meeting and emails to communicate.

4 Conclusion

On the whole work done, the assimilation of the knowledge required for this project and the production of a valid code took much longer than expected. The road map was corrected early enough to focus on the most important thing: a valid means of simulation. It was better to spend more time on the latter in order to be able to work on a healthy foundation during the internship. The first draft of the link between the calculation code and the Python code for the learning machine is also being put in place, which will save time for the rest.

Concerning the questions raised at the beginning of the project, the idea of coupling scientific computation with machine learning is recent and there are certainly many new techniques that will come out of it. Throughout this project, I have laid the foundations for this exploration, I have indeed a valid and efficient way to create a database of simulations.

We have to keep in mind that the shallow water system I've been using is a simplified 1D model that assumes the ocean is a thin layer of water and doesn't account for wave dispersion, so the idea of this project is to provide some initial evidence.

During the internship that follows, I plan to train and test neural networks using those simulations. It might gives us some tangibles evidence on the use of machine learning in scientific computing. Then, I would like to implement a new scheme that consider dispersing effects. It will be based on the article [4] and called the Green-Nagdhi model.

References

[1] Emmanuel Audusse, Christophe Chalons, Philippe Ung. A simple well-balanced and positive numericalscheme for the shallow-water system. Communications in Mathematical Sciences, International Press,2015, 13 (5), pp.1317-1332. 10.4310/CMS.2015.c13.n5.a11. hal-01083364v2. https://hal.archives-ouvertes.fr/hal-01083364/file/simple_wellbalanced_positive_ARS.pdf.

[2] Equations de Saint Venant et application aux mouvements de fonds érodables." Écoulements en milieux naturels" Cours MSF12, M1 SU, P.-Y. Lagrée. http://www.lmm.jussieu.fr/~lagree/COURS/MFEnv/MFEnv.pdf.

[3] Méthodes de volumes finis pour les fluides compressibles, Nicolas Seguin, Université Pierre et Marie Curie - Paris 6. http://seguin.perso.math.cnrs.fr/DOCS/cours.pdf.

[4] Philippe Bonneton, Florent Chazel, David Lannes, Fabien Marche, Marion Tissier. A splitting ap-proach for the fully nonlinear and weakly dispersive Green-Naghdi model. Journal of ComputationalPhysics, Elsevier, 2010, 230 (4), pp.1479-1498. 10.1016/j.jcp.2010.11.015. hal-00482564. https://hal.archives-ouvertes.fr/hal-00482564/document.