LEGI
LABORATOIRE DES ECOULEMENTS GÉOPHYSIQUES ET INDUSTRIELS

OSUG
Observatoire des Sciences de l'Univers de Grenoble

Université Joseph Fourier
GRENOBLE

M2R Internship

Modelling the multiscale interactions of the coupled atmosphere-ocean-sediment system, induced by Marine Renewable Energy Devices

Supervisors : Julien CHAUCHAT & Achim WIRTH

LEGI Laboratory - Team MEIGE

Tim NAGEL

M2R SIM - UJF

Abstract

While the world assist to an increasing development of offshore wind energy, the environmental impact of offshore wind farms remains still unknown. If some recent studies have recently light up the impact of offshore wind farms piles on the seabed morphodynamic or the slight influence of wind farm presence on regional climate, no one has, as far as we know, ever considered the turbine wake impact upon both ocean and sediment layers. The purpose of this internship was to study this phenomenon an a local scale with an idealized 2D numerical model.
A large part of the work has consisted in the building and testing of the numerical model. Starting from nothing, the model consist in two modules, on for the ocean which solves the Shallow Water equation and one other for the sediment transport, solving the sediment mass conservation equation. The two modules are coupled with a quadratic friction law. Tests undertaken in one and two dimensions and comparisons with analytical solutions show the model accuracy and robustness.
The results light the fact that Kelvin-Helmholtz instabilities generated in the ocean by the wake presence seems to be controlled by the bottom friction. Sufficiently low friction induces a homogeneous turbulence state. Increasing the water depth leads to that turbulence state. Due to the turbulence, the morphological impact is thus lower. In the short time (days) main effects on the seabed are linked to the mean wake when in the long term, changes in the seabed could be dominated by turbulent or laminar nature of the wake. Nevertheless, more computations are necessary to ensure these first conclusions.

Résumé

Alors que l'on assiste à un développement croissant de l'éolien offshore au niveau mondial, l'impact environnemental des fermes eoliennes offshore demeure pour le moment assez peu connu. Si certaines études récentes on montré l'impact de ces constructions sur la dynamique climatique européenne ou encore celui des piles de turbines sur l'évolution de fond, aucune, ne fait à notre connaissance état de l'impact du sillage de ces structures sur la dynamique océanique et sédimentaire. Le but de ce stage était d'étudier ce phénomène à l'échelle locale grâce à un modèle numérique idéalisé en deux dimensions.
Une importante partie du stage à consisté en l'élaboration et la phase de tests du modèle numérique. Ce dernier se compose de deux modules, le premier résolvant les équations de Shallow Water pour l'océan et le second résolvant l'équation de conservation de la masse du sédiment pour le transport sédimentaire. Les deux modules sont couplés à l'aide d'une loi de friction quadratique. Les tests entrepris en 1D et 2D ainsi que les comparaisons avec des solutions analytiques ont permis de prouver l'efficacité et la précision du modèle.
Les résultats indiquent que la genération par le sillage d'instabilités de Kelvin-Helmholtz dans l'océan semble être contrôlée par la la friction du fond. Une friction suffisamment basse permet la génération une turbulence homogène dans l'océan. L'augmentation de la profondeur entraîne un tel état de turbulence. A cause de la turbulence, l'impact morphologique est donc plus faible. A court terme (de l'ordre du jour) les perturbations du fond sont du à la présence du sillage tandis qu'à long terme elles sont induites par la nature turbulente ou laminaire du sillage et de l'écoulement. Néanmoins, d'autres simulations sont nécessaires pour confirmer ces premières conclusions.

Acknowledgments

The first two persons I would like to thank are my two supervisors, Julien Chauchat and Achim Wirth, with whom I had the pleasure to work on a very interesting and profound subject. Their kindness and their availability immediately made me feel comfortable for my intership. I would specially thank them for their advices, for their precious help and knowledges and especially for the time they spend with me, always constructive. I really hope continue to work with them for my Ph.D Thesis!
As my internship was entirely numerical work, I couldn't manage it without the help of computer specialists: Cyrille Bonamy and Olivier De-Marchi, thank you.
The LEGI is a very pleasant place to work, I wish to thank everybody (interns, Ph. D. fellows, permanents) for the greats moments shared at the lunch-time or around a barbecue: thank you guys! I would just address a special thank to Thibault for his help and for the climbing sessions. Finally, I thank Charlotte for all that she brings to me.

Contents

Chapter 1

Introduction

Because of the rising need for sustainable energy, and because up to now wind energy is one of the few forms of renewable energy that can be harvested efficiently, many European countries are planning and building offshore wind farms to increase the proportion of renewable energy in their energy mix. According to the European Wind Energy Association 2013 annual report [*EWEA*, 2013], installed the European capacity was 5 GW at the end of 2012, and by 2020 it could be multiplied by 8 (40 GW), corresponding to 4% of the European electricity demand. By 2030, offshore wind capacity could totalize 150 GW, corresponding to 14% of the actual EU's total electricity consumption.
France has the ambition to install 6 GW off offshore wind turbines before 2020 and would become one of the world leaders in terms of offshore wind turbines capacity. In this context the French Institut for Renewable Marine Energy (France Energies Marines) has been created in order to support the research and the development on Marine Renewable Energy Devices.

In this context, the environmental impact of offshore wind farms has to be studied. Indeed, most the wind farms are localized in coastal areas (in 2012 the average water depth of offshore wind farms was 22m and the average distance to shore was 29km [*EWEA*, 2013]). It has been shown that wind farms have an impact on the seabed [*Van der Veen et al.*, 2007] but also on the atmosphere via the wake generation behind the rotor and on the ocean when the wakes impact the free surface [*Barbe*, 2013]. On a larger scale, impact of wind farms on the European regional climate [*Vautard et al.*, 2014] or on hurricanes [*Jacobson et al.*, 2014] have been studied recently, showing that the wind farms environmental impacts are an important questions nowadays.

The understanding of the impact induced by the construction of offshore wind farms in coastals areas requires the modeling of the ocean-atmosphere-sediment coupled system. However, if air-sea and ocean-sediment interaction have been the subject of an important literature, to the best of our knowledge, no study has been done on the multi-scale interactions in an atmosphere -ocean-sediment coupled system. This is even more true in the vicinity of Marine Renewable Energy Devices.
The aim of this internship is to build, starting from the mathematical model, an idealized 2D numerical model to study the impact of an offshore turbine wake on the ocean and the sediment dynamics.

Chapter 2

Physical and Mathematical Model

2.1 Physical Model

The physical model consists in two superposed layers (figure 2.1). The upper one is an homogeneous shallow water ocean layer. The lower one is the sediment bed layer, composed of cohesionless particles. The atmospheric layer is not considered in this internship work, the wind acts as the external forcing F.

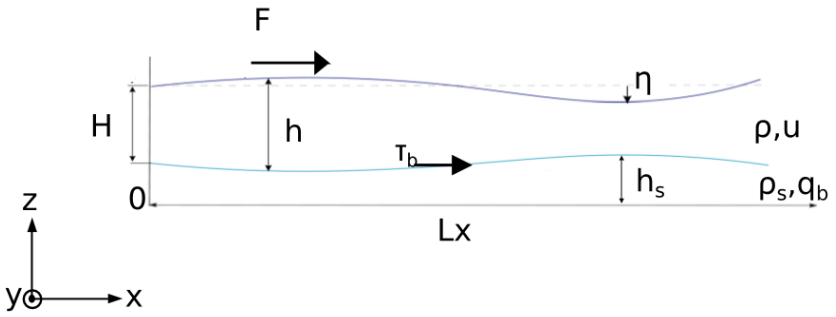


Figure 2.1 – Sketch of the physical model, exponent "s" refers to the sediment bed. From *Moulin* [2012]

The domain length in the x-direction is denoted as L_x and as L_y in the y-direction. The average depth of the ocean layer is denoted as H. The local thickness of the ocean layer and the seabed elevation, are denoted as $h(x, y, t)$ and $h_s(x, y, t)$ respectively. The ocean layer elevation can be described by the following equation:

$$h(x, y, t) = H + \eta(x, y, t), \tag{2.1}$$

where η represents the free sea surface perturbations, depending on space and time. The dimensional variables are consigned in the table 2.1. Densities are denoted as ρ, ρ_s and ρ_a for the ocean, the sediment and the atmosphere, respectively. The ocean layer is forced by the local wind stress at its upper surface. The spatial variation of the wind stress incorporates the wake-profile of a wind turbine. The wind stress is a quadratic function of the wind-speed following the classical drag law: $f = C_D\ \rho_a \mid u_a \mid u_a$
A large scale forcing acting on the ocean (G) can be added in the model. Physically it represents an horizontal pressure gradient ($\nabla_h P$), that is large scale or tidal currents. The oceanic motion induces a shear stress τ_b on the sediment bed layer. This stress is responsible for the coupling

Table 2.1 – Domain parameters.

ρ (kg.m^{-3})	ρ_s (kg.m^{-3})	ρ_a (kg.m^{-3})	L_x (m)	L_y (m)	H (m)
1025	2650	1.2	10^3	10^3	1-10

between the ocean and the sediment bed layers. It is also parametrizes by a quadratic friction law (see section 2.2.2). For a bed composed of cohesionless grains (equivalent to sand), the sediment start to move when the drag force exerted by the flow is higher than the friction force between the grains. This is characterized by the dimensionless Shields parameter,[*Buffington*, 1999] which gives the threshold of motion for sediments at the bed:

$$\theta = \frac{\tau_b}{(\rho_s - \rho)gd}, \tag{2.2}$$

where g is the gravitational acceleration and d is the characteristic sediment particle diameter. The sediment starts to move as soon as the Shields parameter exceeds a typical critical value ($\theta > \theta_c$). The critical Shields parameter, θ_c, depends on several sediment properties as the density or the grain size.
The Shields parameter represents the ratio between the drag force and the apparent submerged grains weight.

2.2 Mathematical Model

The physical model presented in the previous section can be represented by a mass conservation for the sediment, and by the Shallow Water (SW) equations for the ocean. Their coupling is done by a turbulent drag law, this law allow also to compute the bed shear stress. Friction forces are the link between the three phases and are therefore one of the most important parts of the model, this laws have thus to be well parametrized in order to ensure the model accuracy.

2.2.1 Morphodynamic Model

Bed motion results from a local flux balance. The mass conservation equation, also called Exner equation, allows to calculate time evolution of the seabed elevation (h_s):

$$\partial_t h_s(x,y,t) + \partial_x q_x(x,y,t) + \partial_y q_y(x,y,t) = 0 \ , \tag{2.3}$$

where $q \begin{pmatrix} q_x \\ q_y \end{pmatrix}$ is the total sediment flux. mainly two types of sediment transport exists, bed-load transport and suspended load transport. In this work, only bed-load will be considered, this type of transport is generally dominating for rather low bed shear stress, i.e when the Shields number of the flow is closed to the critical one
Total sediment flux can be written as:

$$\begin{aligned} q_x(x,y,t) &= \frac{1}{1-p}\left(q_{bx}(x,y,t) + q_{sx}(x,y,t)\right) \\ q_y(x,y,t) &= \frac{1}{1-p}\left(q_{by}(x,y,t) + q_{sy}(x,y,t)\right) \ , \end{aligned} \tag{2.4}$$

where p is the bed porosity, q_{bx} is the bed load transport rate and q_{by} is the suspended load transport rate.
If only bed-load transport is considered, the total sediment flux becomes:

$$\begin{aligned} q_x(x,y,t) &= \frac{1}{1-p}\left(q_{bx}(x,y,t)\right) \\ q_y(x,y,t) &= \frac{1}{1-p}\left(q_{by}(x,y,t)\right) \ , \end{aligned} \tag{2.5}$$

Many flux formulas have been proposed in the literature but the most widely used for bed-load transport is the *Meyer-Peter and Müller* [1948] (MPM) formula which relates the particle flux q_b to the excess Shields parameter:

$$\begin{cases} \dfrac{q_b}{d\sqrt{(\rho_s/\rho)gd}} = 8(\theta-\theta_c)^{3/2} & \text{if } \theta > \theta_{\mathrm{c}} \\ q_b = 0 & \text{otherwise} \end{cases} \tag{2.6}$$

The other transport formula implemented in the code is a simpler bed-load formula in which the total flux is proportional to the fluid velocity at a given power:

$$q_b = \alpha u^{\beta} \ , \tag{2.7}$$

where α, β are two constants.
The choice in the sediment transport law is therefore very important, given that the resulting fluxes are used in the seabed morphodynamic calculation. The MPM formula has always been implemented except for 1D model validation, because comparisons with analytical results were only possible with equation 2.7 (see chapter 4).

2.2.2 Shallow Water Equations

The Shallow Water equations are derived from the Navier-Stockes equations assuming several simplifications:

- ➔ The ocean is composed of an incompressible fluid.
- ➔ The vertical scale is very small compared to horizontal one (i.e ocean layer is very thin)
- ➔ The density variations within the ocean layer are very low.

In two dimensions, the SW equations are given by:

$$\partial_t u + u\partial_x u + v\partial_y u + g\partial_x \eta \ = \ \nu\nabla^2 u + F_x - G_x \tag{2.8}$$

$$\partial_t v + u\partial_x v + v\partial_y v + g\partial_y \eta \ = \ \nu\nabla^2 v + F_y - G_y \tag{2.9}$$

$$\partial_t h + \partial_x(hu) + \partial_y(hv) \ = \ 0, \tag{2.10}$$

Where F and G correspond to the forces applied on the ocean layer by the atmosphere and the sediment, respectively. The atmosphere acts the ocean layer thanks to the wind shear stress. It can be written as:

$$\begin{pmatrix} F_x \\ \\ F_y \end{pmatrix} = \frac{1}{\rho h}\begin{pmatrix} f_x \\ \\ f_y \end{pmatrix}, \tag{2.11}$$

where f_x and f_y are the surface forces depending on x and y axes for which a quadratic friction law is used:

$$\begin{pmatrix} f_x \\ \\ f_y \end{pmatrix} = \begin{pmatrix} C_D\ \rho_a \mid u_a \mid u_a \\ \\ C_D\ \rho_a \mid u_a \mid v_a \end{pmatrix}, \tag{2.12}$$

where $\mid u \mid = \sqrt{u^2 + v^2}$ and where C_D is the drag coefficient, *Wu* [1982] and *Smith* [1988] suggest:

$$C_D = (0.6 + 0.07u).10^{-3} \quad \text{for 6 m/s} < \text{u} < 26 \text{ m/s} \tag{2.13}$$

The range of availability of this friction law is in agreement with the good working conditions of wind turbines (from 3 m.s^{-1} to 30 m.s^{-1}) [1] A similar quadratic friction law is used to modelise the friction between the ocean and the sediment layer:

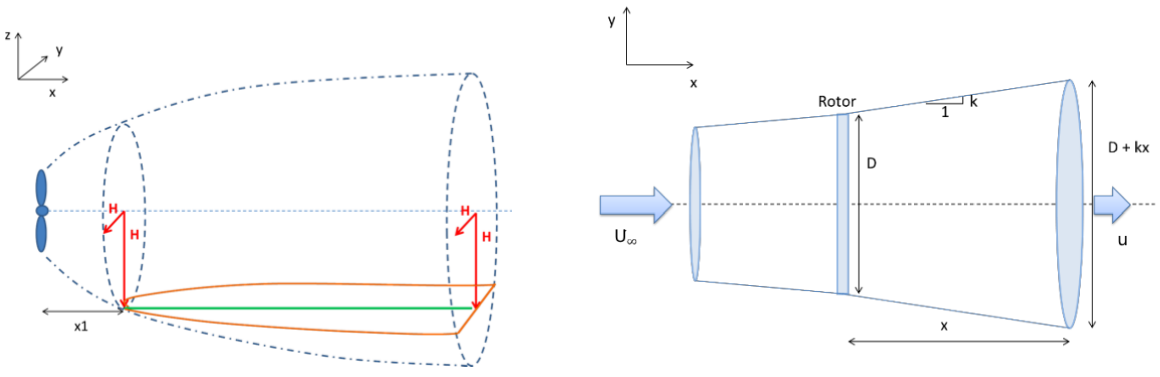
$$\begin{pmatrix} G_x \\ \\ G_y \end{pmatrix} = \frac{1}{h} \begin{pmatrix} \tau_{bx} \\ \\ \tau_{by} \end{pmatrix}, \tag{2.14}$$

$$\begin{pmatrix} \tau_{bx} \\ \\ \tau_{by} \end{pmatrix} = \begin{pmatrix} C_{Ds} \mid u \mid u \\ \\ C_{Ds} \mid u \mid v \end{pmatrix}, \tag{2.15}$$

where $C_{Ds} = 0.005$ is the friction coefficient between the sand and the ocean.
The ocean layer is thus reacting to the wind forcing via a quadratic friction law and in return, the ocean layer transmits this stress to the sediment layer via the a similar quadratic friction law.

2.2.3 Wake Induced by Turbines

As air flows through a wind turbine and energy is extracted from it, some of the properties of the flow are changed: the air is decelerated and turbulence intensity is increased. The region of the flow behind a turbine where these properties are changed is called the wake of a wind turbine and the effects are referred to as wake effects.



(a) (b)

Figure 2.2 – Illustration of a turbine wake (a) and of Jensen wake downstream the rotor (b). The red zone on (a) correspond to the ocean surface which is affected by the wake. From *Barbe* [2013].

1. http://www.france-energies-marines.org/Les-energies-marines-renouvelables/L-energie-eolienne-en-mer

When regarding wakes, a distinction can be made between the near and far wake regions. The near wake is the area just behind the rotor, where the properties of the rotor can be discriminated. It extends approximately up to one rotor diameter downstream. The far wake is the region beyond the near wake, where the focus is put on the bulk influence of wind turbines, so modeling the detailed rotor motion is less important [*Vermeer et al.*, 2003]. In the far wake the two main mechanisms determining the flow conditions are advection and turbulent diffusion and in many situations a parabolic approximation is appropriate to deal with this region. It is expected that sufficiently far downstream, the effects of momentum deficit and increased level of turbulence will vanish because of turbulent diffusion in the wake.

Obviously, in the case of offshore wind turbines, the downstream ocean surface is affected by the presence of the wake, as illustrated in figure 2.2. According to *Barbe* [2013], the turbine wake affects the ocean by decreasing its surface velocity. This perturbation is maximal at the impact location and then decreases downstream. A distance of 40 to 60 times the rotor diameter D is necessary for the wake effects to vanish.

Several models describe the velocity deficit induced by the wake behind a turbine, but the two major ones are now as Jensen's [*Jensen*, 1983] and Larsen's [*Larsen et al.*, 1996] models. Both are kinematic models, *i.e* they only use the momentum equation to model the velocity deficit. In this work, I will use a modified exponential model similar to the Jensen's one. The later is the oldest and simplest wake model, assuming a linearly expanding wake with a velocity deficit which only depends of the distance to the rotor. Because the velocity in the wake is constant for a given downstream distance, the velocity profile is called 'hat-shaped'. This model is given by the following relationship:

$$u = U_\infty \left[1 - \frac{1 - \sqrt{1 - C_w}}{(1 + \frac{2kx}{D})^2} \right] \ , \tag{2.16}$$

where U_∞ is the wind velocity far from the turbine, C_w the drag coefficient between the turbine and the air and k the Wake Decay Constant, equivalent to the wake's linear slope (figure 2.2). For offshore turbines, the admitted value is equal to 0.04 [*Renkema*, 2007].
important in order to avoid interferences between the wake and the periodic boundaries. Such domain would be computationally too expensive in respect to the internship duration. The solution chosen is to implement a similar linear perturbation model using a wake decay modeled by an exponential function. The velocity deficit profile has the same shape as Jensen's model, but, for a same wind turbine size, perturbations can be easily parametrized in order to reduce the wake length. This model gives the velocity in the wake as:

$$u = U_\infty \left[1 - e^{-\frac{x-a}{b}} \right] \ , \tag{2.17}$$

where a and b are adjustable coefficients.

2.2.4 Wake Interaction with Ocean Surface

The wake impacts the ocean surface at a given distance downstream the rotor position (figure 2.16). Determination of this distance is done using the Jensen's model, indeed, usual trigonometry gives:

$$x_{impact} = \frac{H_r - D/2}{k} \ , \tag{2.18}$$

This impact distance depends on the wind turbine height and on the rotor diameter, increasing with the turbine size. Nevertheless, even for small turbines (H_r = 70 m and D = 80 m), the

impact distance is equal to 750 meters, several times the turbine height. Perturbations in the seabed induced by the turbine pile are localized in the pile vicinity, the seabed is thus perturbed on a local scale (10 times the pile diameter). As the pile diameter doesn't exceed 20 meters for the biggest turbines, seabed perturbations induced by the pile and the wake presence are uncorrelated. The effect of the pile is thus not considered here.

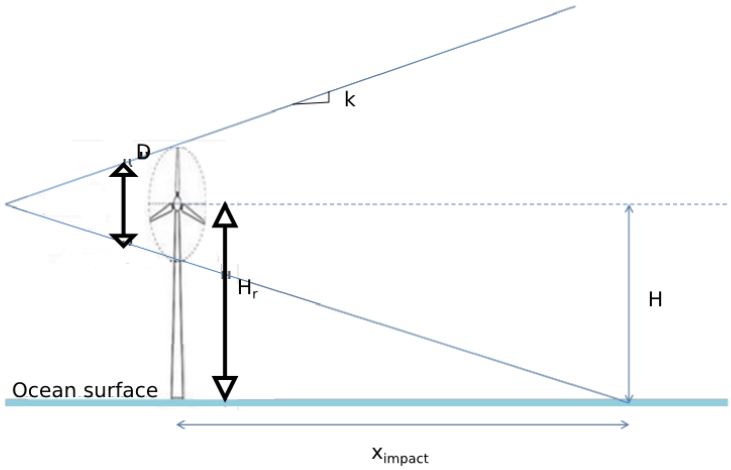


Figure 2.3 – Scheme of Jensen wake. From *Barbe* [2013].

Chapter 3

Numerical Model

Written in Fortran 90, the overall model can be divided in two coupled modules, the hydrodynamic and the morphodynamic modules, subject to input data, such as atmospheric forcing, bedform elevation and imposed currents, the hydrodynamic module solves the SW equations, using a second order Runge-Kutta scheme (see section 3.3). Output from this module are velocity fields u and v along with the free surface elevation η. These velocity fields are necessary to compute the Shields number. As morphological timescale is large compared to the to hydrodynamic one, two time-steps are used, one for the hydrodynamic module (Δt) and the other for the morphodynamic module (Δt_{morpho}) with $\Delta t_{morpho} = 1000 \times \Delta t$. The morphodynamic module is only called if Shields number exceeds the critical value in one or more of the grid points. The morphodynamic module solves the Exner equation using a NOCS scheme (see section 3.2).

3.1 Numerical Grid, Boundary Conditions and Stability Criterion

3.1.1 Numerical Grid

The numerical grid is two dimensional (x and y) and is the same for the two layers. Because of SW approximation, only the horizontal velocities u and v are considered explicitly.

The discretization domain is a rectangle of $Lx \times Ly$. As described in section 3.2, boundary conditions are periodic for all the layers in both x and y directions. The grid is regular, with identical spatial resolutions in the x and y directions, $\Delta x = Lx/nx = 1$ m and $\Delta y = Ly/ny = 1$ m where nx and ny are the number of grid points in the x and y direction, respectively.

3.1.2 Boundary Conditions

The periodic boundary condition is explained here for a one dimensional case in the x-direction, extension to two dimensions is straightforward. For the shallow water and the morphological modules, the value calculated at grid point i involves values at points $i-1$ and $i+1$. Thus, at the boundaries, values of each variable has to be given. Here, periodic boundary conditions are used. As shown in figure 3.2, every point which is coming out of the domain at a boundary reappears at its opposite side.

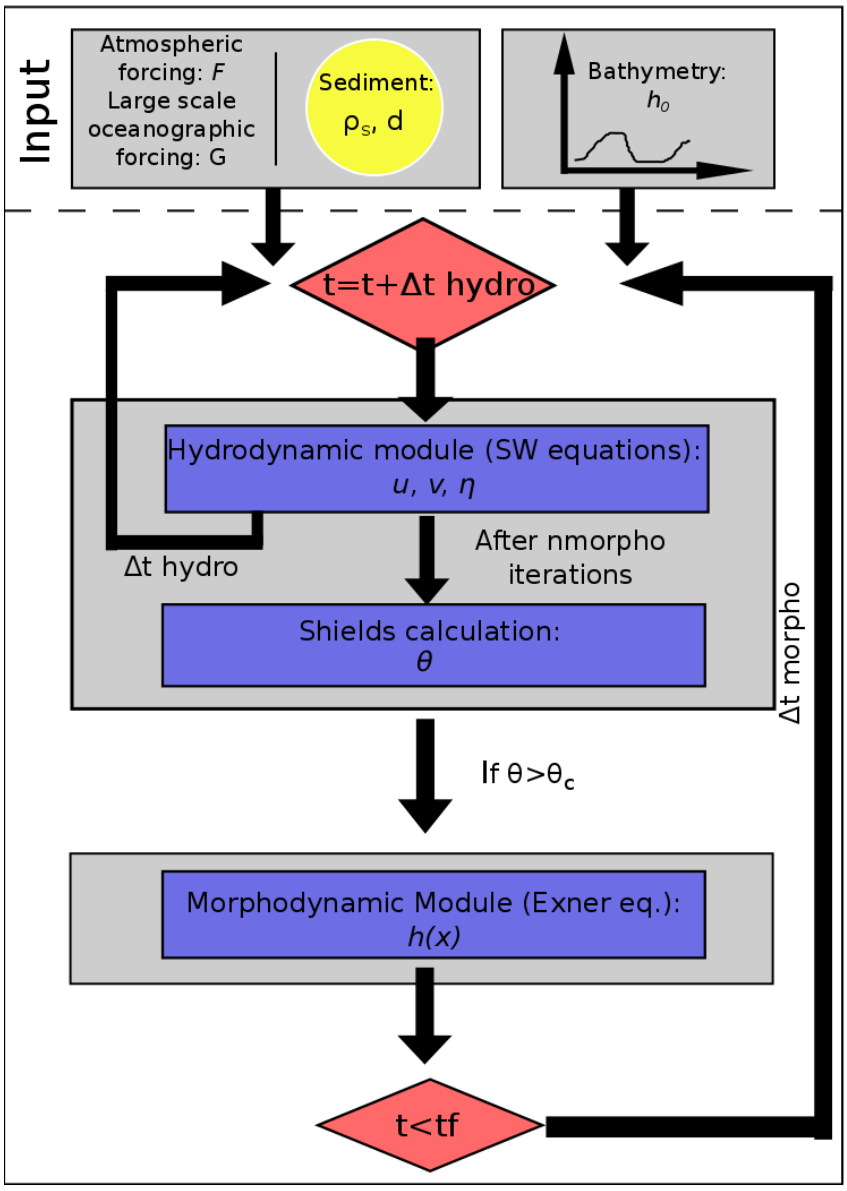


Figure 3.1 – Organization chart of the code, $nmorpho = \Delta t_{morpho}/\Delta t$

Figure 3.2 – Scheme of the periodic boundary conditions in the x-direction. Source: *Moulin* [2012].

The same process is applied in the y direction.

3.1.3 Stability Criterion

For the stability of the numerical scheme, the Courant-Friedrichs-Lewy criterion (CFL) has to be respected:

$$\frac{\mid c + u_{max} \mid \ dt}{dx} \leq 1 \ , \tag{3.1}$$

where c is the wave velocity.
The ratio on the left side of this condition is called the CFL number. The criterion has a simple interpretation show in figure 3.3. The domain of dependency, shown shaded, consists of all points in the past from which information can propagate at or slower than the wave speed c. For any differencing scheme, the differencing domain consists of the set of points used to determine the next value of the solution. If the differencing domain is wider in x than the domain of dependency, then the algorithm is CFL stable. If the differencing domain is narrower in x than the domain of dependency, then the algorithm is CFL unstable.
To summarize, the distance of wave propagation during one time step (Δt) must be smaller than the mesh grid size.

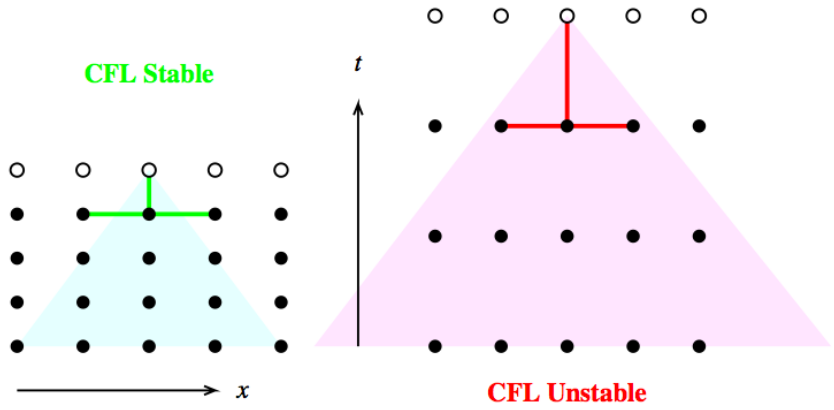


Figure 3.3 – Scheme of the CFL conditions. Source: *www.physics.buffalo.edu/phy410-505*

CFL condition must be filled for both ocean and sediment layers. While ocean waves propagation celerity is classically defined as $c_{ocean} = \sqrt{gh_o}$ (h_o is the ocean layer thickness), sand waves propagation celerity (c_{sand}) corresponds to dunes migration speed. As in the model $h \in [6; 20]$ m, $c_{ocean} \in [7.7; 14]$ m.s^{-1} and $c_{sand} \approx 10^{-4}$ m.s^{-1}, if the CFL conditions for the ocean is filled, it is also the case for the sedimentary CFL.

3.2 Morphodynamic Model

This section describes the morphodynamic module implemented into the code to solve equation (2.3). Because of the possible presence of steep slopes, conservative shock-capturing schemes have to be used. Here, a NOCS (Non-Oscillatory Central Scheme) collocated with the mesh has been chosen. This type of scheme is able to solve the conservative forms of equations and have a strong stability on shock areas [*LeVeque*, 2002].
In order to be closer to the granular flow physic, an avalanche management module has been added to the morphodynamic model. Indeed, for strong enough dune slopes, local avalanches can occur, this phenomenon has to be taken into account.

Two types of NOCS schemes have been tested in this work. In one dimension, I used a non-staggered NOCS scheme, developed by *Nessyahu and Tadmor* [1990] and successfully used by *Marieu* [2007] in his work to solve the Exner equation. For this scheme, and for the avalanche management, the module tests have been done on an simple advection case explained in chapter 4. Furthermore, comparisons of non-staggered NOCS scheme and Upwind scheme with analytical solution obtained from the characteristic method have also been undertaken.
Nevertheless, 2D extension of this non-staggered NOCS scheme has not been proposed in the literature, but such work have been done by *Jiang et al.* [1998] for the staggered NOCS scheme. Tests undertaken in (4.3) show that for a one dimensional case, the two schemes give similar results.

3.2.1 NOCS Non-Staggered Scheme

The NOCS scheme solves the Exner equation in two steps: a predictor-step, that gives a temporary bedform from which fluxes are recalculated and the corrector-step in which the definitive bedform is obtained. The NOCS used in this work is collocated with the mesh.
The numerical scheme presented here has been used by *Marieu* [2007].

The predictor step gives the bed elevation at grid point i after one half time-step calculation (n+1/2):

$$h_i{}^{n+\frac{1}{2}} = h_i{}^{n} - \frac{1}{2}\frac{\delta t}{\delta x}\, q_i{}' \ , \tag{3.2}$$

where $q_i{}'$ is the flux derivative approximation at grid point i. The temporary flux depends only on the bed elevation $h_i{}^{n+\frac{1}{2}}$:

$$q_i{}^{n+\frac{1}{2}} = q\left(h_i{}^{n+\frac{1}{2}}\right) \ , \tag{3.3}$$

The corrector-step gives the bed elevation at grid point i at time n+1:

$$h_i{}^{n+1} = \frac{1}{2}\left(h_{i+1}{}^{n} + h_{i-1}{}^{n}\right) + \frac{1}{4}\left(h_{i-1}{}' - h_{i+1}{}'\right) - \frac{\delta t}{2\delta x}\ \left(q_{i+1}{}^{n+\frac{1}{2}} - q_{i-1}{}^{n+\frac{1}{2}}\right), \tag{3.4}$$

where h' is the approximation of the bed elevation derivative.

The calculation of $q_i{}'$ and $h_i{}'$ involves a slope limiter, in order to ensure TVD (Total Variation Diminishing) properties of the solution. In the present work, $\beta-$limiter has been used. In order to compute the derivative approximation of a function ϕ, the $\beta-$limiters are define as follow:

$$\phi_i{}' = \mathrm{MinMod}\left[\beta(\phi_{\mathrm{i}} - \phi_{\mathrm{i-1}}),\ \frac{1}{2}(\phi_{\mathrm{i+1}} - \phi_{\mathrm{i-1}}),\ \beta(\phi_{\mathrm{i+1}} - \phi_{\mathrm{i}})\right] \ , \tag{3.5}$$

where β is the limiter parameter and $MinMod$ the function such as:

$$\text{MinMod}\{\text{x}_1, \text{x}_2, \text{x}_3\} = \begin{cases} min\{x_1, x_2, x_3\} & if\ x_k > 0;\ \forall k = 1, 2, 3, \\ max\{x_1, x_2, x_3\} & if\ x_k < 0;\ \forall k = 1, 2, 3, \\ 0 & \text{else} \end{cases} \tag{3.6}$$

when $\beta = 1$, the limiter is the so-called $MinMod$ and when $\beta = 2$ it is the *Superbee* limiter, the latter is less diffusive.

3.2.2 NOCS Staggered Scheme

One Dimensional Scheme

The NOCS staggered scheme is also composed of a predictor and a corrector step, the latter is on a staggered grid. It is based on the reconstruction of a piecewise-linear interpolant from the known staggered cell-averages at time t^n:

$$\bar{h}(x, t^n) = \sum_i \left[{h_i}^n - {h_i}' \left(\frac{x - x_i}{\Delta x} \right) \right] \chi_i(x), \tag{3.7}$$

where ${h_i}'$ is the discrete slope involving the slope limiter described in equations 3.5 and 3.6, and where $\chi_i(x)$ is the characteristic function of the cell $I_i := \mid x - x_i \mid \leq \Delta x/2$.
This interpolant is then projected on the staggered cell-averages on the next time step, t^{n+1} , resulting in the two-step predictor-corrector form:

$${h_i}^{n+\frac{1}{2}} = {h_i}^n - \frac{1}{2} \frac{\delta t}{\delta x}\ {q_i}'\ , \tag{3.8}$$

$${h_{i+\frac{1}{2}}}^{n+1} = \frac{1}{2} \left({h_i}^n + {h_{i+1}}^n \right) + \frac{1}{8} \left({h_i}' - {h_{i+1}}' \right) - \frac{\delta t}{\delta x}\ \left({q_{i+1}}^{n+\frac{1}{2}} - {q_i}^{n+\frac{1}{2}} \right), \tag{3.9}$$

The staggered corrector has to be reprojected on the non-staggered grid by using a piecewise-linear interpolant through the calculated staggered cell-averages at time t^{n+1}:

$$\bar{h}_{i+\frac{1}{2}}{}^{n+1} = {h_{i+\frac{1}{2}}}^{n+1} - {h_{i+\frac{1}{2}}}' \left(\frac{x - x_{i+\frac{1}{2}}}{\Delta x} \right), \tag{3.10}$$

where ${h_{i+\frac{1}{2}}}'$ is the staggered discrete derivative of the staggered corrector term.
Finally, the cell-averages at time t^{n+1} are obtained by averaging this interpolant,resulting in the non-staggered corrector scheme:

$$\begin{aligned} {h_i}^{n+1} = &\ \frac{1}{\Delta x} \left[\int_{x_{i-\frac{1}{2}}}^{x_i} \bar{h}_{i-\frac{1}{2}}{}^{n+1} + \int_{x_i}^{x_{i+\frac{1}{2}}} \bar{h}_{i+\frac{1}{2}}{}^{n+1} \right] \\ {h_i}^{n+1} = &\ \frac{1}{4} \left({h_{i-1}}^n - 2{h_i}^n + {h_{i+1}}^n \right) - \frac{1}{16} \left({h_{i+1}}' - {h_{i-1}}' \right) - \frac{1}{8} \left({h_{i+\frac{1}{2}}}' - {h_{i-\frac{1}{2}}}' \right) \\ &\ - \frac{\delta t}{2\delta x}\ \left({q_{i+1}}^{n+\frac{1}{2}} - {q_i}^{n+\frac{1}{2}} \right) \end{aligned} \tag{3.11}$$

Figure 3.5 shows the second order construction process leading to the non-staggered corrector scheme.
For staggered variables, boundary conditions slightly differ from the ones described in section 3.2. Indeed, as the grid is staggered extreme points are out of the domain. In 1D, the non staggered grid has the size n, while the staggered one has the size $n + 1$, starting from 0 to n. For a given f function, periodic boundary conditions are given by:

$$f(0) = f(n-2)$$
$$f(n) = f(2)$$
$$f(n-1) = f(1)$$

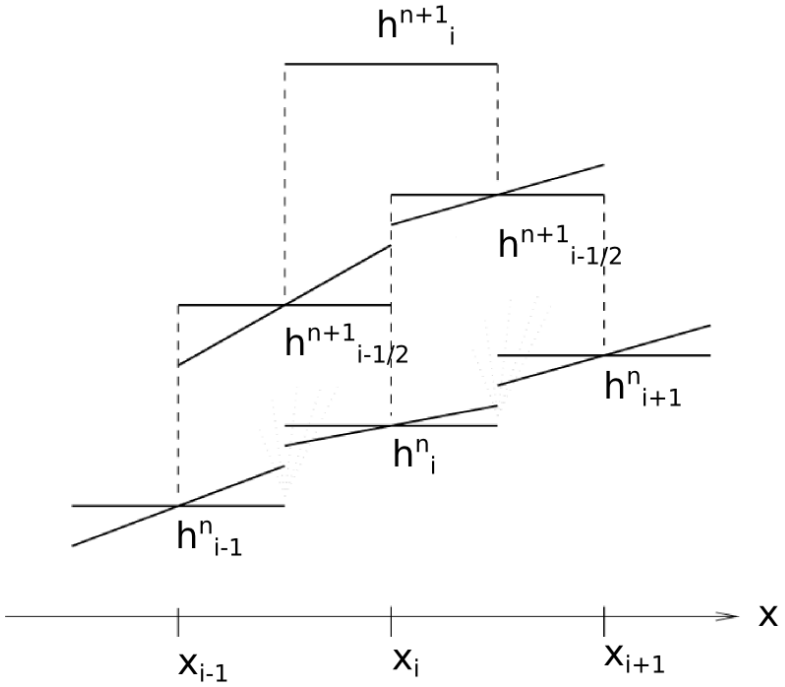


Figure 3.4 – Second order reconstruction. From *Jiang et al.* [1998].

Two Dimensional Extension

The arguments applied in the one-dimensional case can be easily extended to higher dimensions. This extension is straightforward and is based on exactly the same procedure described in section 3.2.2. A non-staggered scheme is created from a staggered scheme by averaging the interpolants constructed from the given staggered values. In two dimensions, predictor and staggered corrector become respectively:

$$h_{i,j}{}^{n+\frac{1}{2}} = h_{i,j}{}^{n} - \frac{1}{2}\frac{\delta t}{\delta x}\ qx_{i,j}{}' - \frac{1}{2}\frac{\delta t}{\delta y}\ qy_{i,j}{}' \ , \tag{3.12}$$

where $qx_{i,j}{}'$ and $qy_{i,j}{}'$ are the flux derivative approximation in the x and y directions respectively.

$$\begin{aligned} h_{i+\frac{1}{2},j+\frac{1}{2}}{}^{n+1} = \ & \frac{1}{4}\left(h_{i,j}{}^{n} + h_{i+1,j}{}^{n} + h_{i,j+1}{}^{n} + h_{i+1,j+1}{}^{n}\right) \\ & + \frac{1}{16}\left(h'_{i,j} + h'_{i+1,j} + h'_{i,j+1} + h'_{i+1,j+1}\right) \\ & + \frac{1}{16}\left(h`_{i,j} + h`_{i+1,j} + h`_{i,j+1} + h`_{i+1,j+1}\right) \\ & - \frac{\delta t}{2\delta x}\ \left(qx_{i+1,j}{}^{n+\frac{1}{2}} - qx_{i,j}{}^{n+\frac{1}{2}} + qx_{i+1,j+1}{}^{n+\frac{1}{2}} - qx_{i,j+1}{}^{n+\frac{1}{2}}\right) \\ & - \frac{\delta t}{2\delta y}\ \left(qy_{i,j+1}{}^{n+\frac{1}{2}} - qy_{i,j}{}^{n+\frac{1}{2}} + qy_{i+1,j+1}{}^{n+\frac{1}{2}} - qy_{i+1,j}{}^{n+\frac{1}{2}}\right) \end{aligned} \tag{3.13}$$

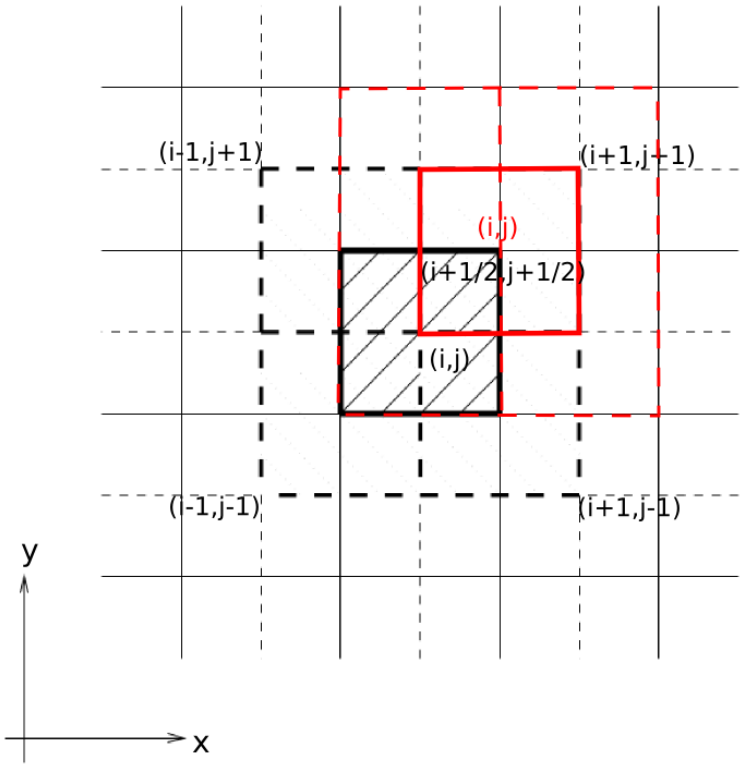


Figure 3.5 – Two-dimensional staggered (red) and non stagerred (black) grids. From *Jiang et al.* [1998].

The prime and back-prime notation denote the discrete derivatives in the x and y directions, respectively. The piecewise-linear interpolant reconstruction is then averaged, resulting in the non-staggered corrector at time step t^{n+1} and in cell $I_{i,j}$:

$$
\begin{aligned}
{h_{i,j}}^{n+1} = \quad & \frac{1}{4\Delta x\Delta y}\left[\int\int_{I_{i+\frac{1}{2},j+\frac{1}{2}}} {\bar{h}_{i+\frac{1}{2},j+\frac{1}{2}}}^{n+1} + \int\int_{I_{i-\frac{1}{2},j+\frac{1}{2}}} {\bar{h}_{i-\frac{1}{2},j+\frac{1}{2}}}^{n+1}\right] + \\
& \frac{1}{4\Delta x\Delta y}\left[\int\int_{I_{i-\frac{1}{2},j-\frac{1}{2}}} {\bar{h}_{i-\frac{1}{2},j-\frac{1}{2}}}^{n+1} + \int\int_{I_{i+\frac{1}{2},j-\frac{1}{2}}} {\bar{h}_{i+\frac{1}{2},j-\frac{1}{2}}}^{n+1}\right] \\
{h_{i,j}}^{n+1} = \quad & \frac{1}{4}\left({h_{i+\frac{1}{2},j+\frac{1}{2}}}^{n+1} + {h_{i-\frac{1}{2},j+\frac{1}{2}}}^{n+1} + {h_{i-\frac{1}{2},j-\frac{1}{2}}}^{n+1} + {h_{i+\frac{1}{2},j-\frac{1}{2}}}^{n+1}\right) \\
& + \frac{1}{16}\left[\left(h'_{i-\frac{1}{2},j-\frac{1}{2}} - h'_{i+\frac{1}{2},j-\frac{1}{2}}\right) + \left(h'_{i-\frac{1}{2},j+\frac{1}{2}} - h'_{i+\frac{1}{2},j+\frac{1}{2}}\right)\right] \\
& + \frac{1}{16}\left[\left(h`_{i-\frac{1}{2},j-\frac{1}{2}} - h`_{i-\frac{1}{2},j+\frac{1}{2}}\right) + \left(h`_{i+\frac{1}{2},j-\frac{1}{2}} - h`_{i+\frac{1}{2},j+\frac{1}{2}}\right)\right]
\end{aligned}
\tag{3.14}
$$

3.2.3 Avalanche Management

As the NOCS scheme is able to capture sharp shocks, it can thus simulate bottom sand dunes with strong slopes. The computed slope can become too steep to be realistic. In such

cases, considering local avalanches is necessary to ensure a realistic shape of the bedforms. Indeed, if the slope angle of a non-cohesive bed α_i becomes larger than the angle of repose α_r (between 28° and 33° for submerged sand), the bed material will slide (avalanche) to form a new slope approximately equal to α_r [*Sánchez and Wu*, 2011].

The avalanche management used here is inspired by *Marieu* [2007] and *Sánchez and Wu* [2011] works. The process of avalanching is simulated by calculating the local slope between the grid points and by enforcing $\alpha_i \leq \alpha_r$ while maintaining mass continuity between adjacent mesh points when the slope is supercritical.
The equation for bed change due to avalanching (eq. (4.3)) is obtained by combining the equation of angle of repose and the continuity equation between two adjacent mesh points and summing over the previous and the next grid points compared to the considered location:

$$\Delta h_i = \begin{cases} R\ (1/2)dx\ (\lambda_{i+1} - sgn[\lambda_{i+1}]\lambda_r) & \text{if } \lambda_{\mathrm{i+1}} > \lambda_{\mathrm{r}} \\ R\ (1/2)dx\ (\lambda_{i-1} - sgn[\lambda_{i-1}]\lambda_r) & \text{if } \lambda_{\mathrm{i-1}} > \lambda_{\mathrm{r}} \\ 0 & \text{else} \end{cases} \tag{3.15}$$

where λ_{i-1}, λ_{i+1}, λ_r and R are respectively the slope between the grid points i and $i-1$, between the grid points $i+1$ and i and the repose slope and the under-relaxation factor (approximately 0.25-0.5). The under relaxation allows to stabilize the avalanching process.

Equation (4.3) is applied by sweeping through all computational grid points to calculate the local bed increase or decrease and then modifying the bathymetry as:

$$h_i{}^{n+1} = h_i{}^{n} + \Delta h_i \tag{3.16}$$

Because avalanching between two grid points may induce new avalanching at neighboring grid points, the sweeping process is repeated until no supercritical slope is detected. This module has been implemented in the 1D model only, in order to allow the MPM formula validation (see section 4.2). A 2D extension does not appear necessary in regards to the smooth shapes obtained for 2D simulations. The avalanche management module test is described in section 4.1.

3.3 Discretization of the Shallow Water Module

The SW module is spatially discretized using the centered finite difference method. Such discretization is based on the Taylor series development, for a function $f(x, y)$, first and second order space derivative are written by:

$$\partial_x f(x,y) = \frac{f(x+\Delta x, y) - f(x-\Delta x, y)}{2\Delta x} + O(\Delta x^2), \tag{3.17}$$

$$\partial_x{}^2 f(x,y) = \frac{f(x+\Delta x, y) - 2f(x,y) + f(x-\Delta x, y)}{\Delta x^2} + O(\Delta x^2), \tag{3.18}$$

A second-order Runge-Kutta time scheme is used for the time discretization. As the morphological ones, this scheme is also a predictor-corrector scheme, the function derivative is evaluated for one half time-step calculation $(t + \Delta t/2)$ as shown in equation (3.19), the complete iteration time step $(t + \Delta t)$ in then calculated from this evaluation (equation (3.20)).

$$f(t + \Delta t/2) = f(t) + \frac{\Delta t}{2} F(t, f(t)), \tag{3.19}$$

where $F(t, f(t)) = \partial_t f(t)$

$$f(t + \Delta t) = f(t) + \Delta t F(t + \frac{\Delta t}{2}, f(t) + \frac{\Delta t}{2}), \tag{3.20}$$

More details about this numerical discretization process are available in *Moulin* [2012].

Chapter 4

Morphodynamic Module Validation

In order to validate the numerical schemes implemented in the morphodynamic module, a 1D test case on the migration of an isolated dune is presented. The system and the variables are described on figure 4.1. The test case consist in a unidimensionnal flow with a constant imposed discharge: $Q_f = 10$ m^2.s^{-1}.

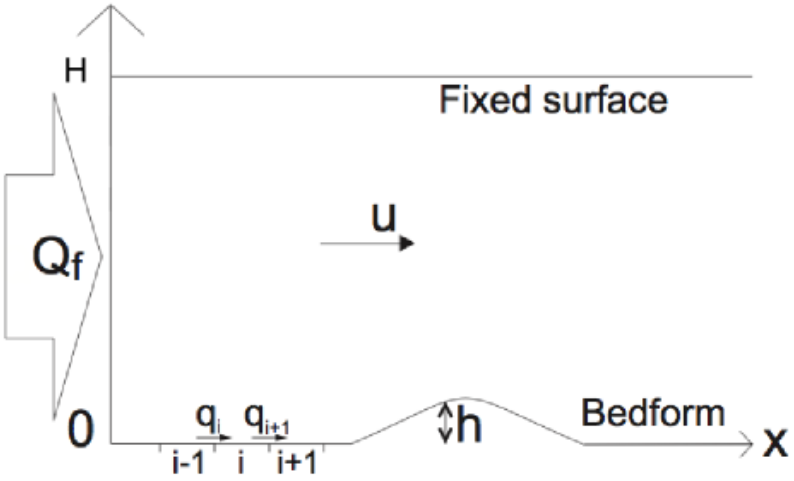


Figure 4.1 – Descriptive scheme of the advection test. Source: *Marieu* [2007].

The free-surface is fixed (sliding wall with a depth of $H = 6$ m), so fluid velocity variations depend on bathymetry variations only. The fluid velocity u can written as a function of the fluid discharge:

$$u(x) = \frac{Q_f}{H - h(x)} \tag{4.1}$$

where h is the bottom height.

The initial bedform consist in a Gaussian sand dune:

$$h(x) = 2e^{-\gamma(x-x_c)^2} \tag{4.2}$$

where $\gamma = 0.01$ m^{-2}, $0 \leq x \leq 300$ m and $x_c = 150$ m.

An analytical solution for the dune migration can be obtained by solving the sediment mass

conservation equation by using the simple bed-load formula (eq. (2.7)). Under such assumption, the sediment flux can be linked to the discharge and the water depth only:

$$q(x) = \alpha \left(\frac{Q_f}{H - h(x)}^{\beta} \right) , \tag{4.3}$$

The chain rule derivation allows to rewrite the horizontal sediment flux gradient as:

$$\frac{\partial q}{\partial x} \approx \frac{\partial q}{\partial h} \frac{\partial h}{\partial x} , \tag{4.4}$$

Therefore mass conservation can be rewritten under non-conservative form as:

$$\frac{\partial h}{\partial t} + a \frac{\partial h}{\partial x} = 0 , \tag{4.5}$$

where a is the bedform celerity:

$$a(h) = \frac{dq}{dh} = \frac{\alpha \beta {Q_f}^{\beta}}{(H - h)^{\beta+1}} , \tag{4.6}$$

where $\alpha = 0.001 \mathrm{s}^2\mathrm{m}^{-1}$ and $\beta = 3.0$, following *Marieu* [2007].

In this simple case, the solution of equation (4.5) can be calculated analytically using the characteristic method. Indeed, the height is conserved on characteristic lines given by the following equation:

$$h(x,t) = h(x,0) \text{ on } \mathrm{x(t)} = \mathrm{x_0} + \mathrm{a(h)t} \tag{4.7}$$

4.1 Simple Bed-load Formulae

Figure 4.2 shows the bottom evolution at different time computed with the Upwind scheme (panel a), the NOCS collocated using the $MinMod$ limiter (panel b) and the NOCS collocated using the *Superbee* limiter (panel c). For all simulations space and time steps are the same, $dt = 1$ s and $dx = 0.5$ m.
The sand dune is advected downstream and the lee side of the dune becomes steeper and steeper, a shock is formed at $t = 2000$ s. After this time, the dune spreads out and its amplitude decreases.

Figure 4.2.d shows that the numerical scheme closest to the analytical solution is the NOCS collocated with the *Superbee* limiter. Indeed, for the time and space step chosen, the $Minmod$ limiter presents a strong numerical diffusion that is too strong (figure 4.2.b), the shock is even not predicted. The Upwind scheme is worse than the *Superbee*, however the shock is predicted.

In order to validate the avalanche management module, two cases have been considered. The first one is a "slump test" currently used for concrete, the second one concern the application to dune migration.

The "slump test" consist in the construction of an immerged sand heap at initial angle α_i greater than the angle of repose ($\alpha_i = 35° > \alpha_r = 28°$). As observed in figure 4.3.a, the local avalanche processing leads to a final heap state where the maximal slope angle is lower or equal than the angle of repose ($\alpha_f = 27.9°$).

Application to the dune migration (figure 4.3.b) shows that the avalanche management module ensure that the maximal value of the downstream slope is always lower than the critical

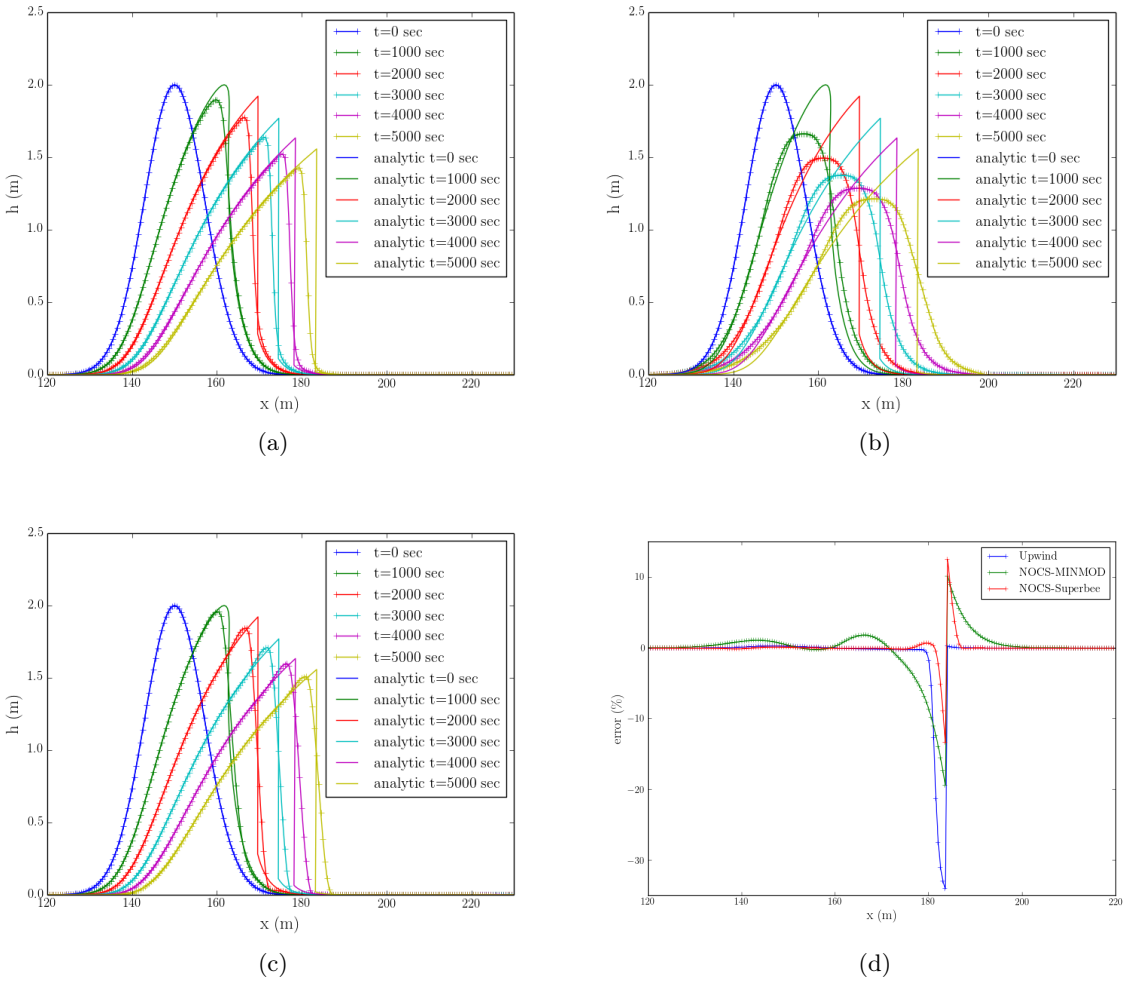
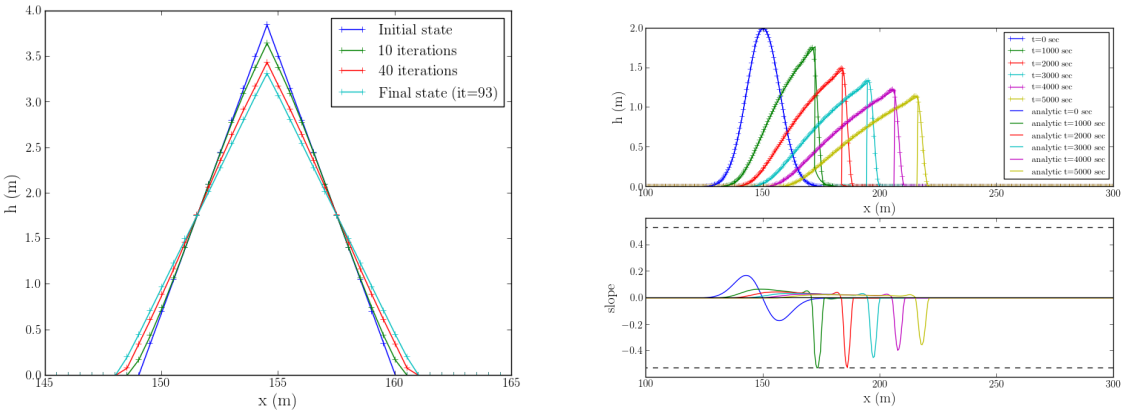


Figure 4.2 – Morphodynamic module test using Upwind scheme (a), NOCS collocated with the *MinMod* limiter (b) NOCS collocated with the *Superbee* limiter (c) and errors between the different schemes used and the analytical solution (d).

stability (dashed lines).
The two tests described here validate the avalanche management module.



(a) (b)

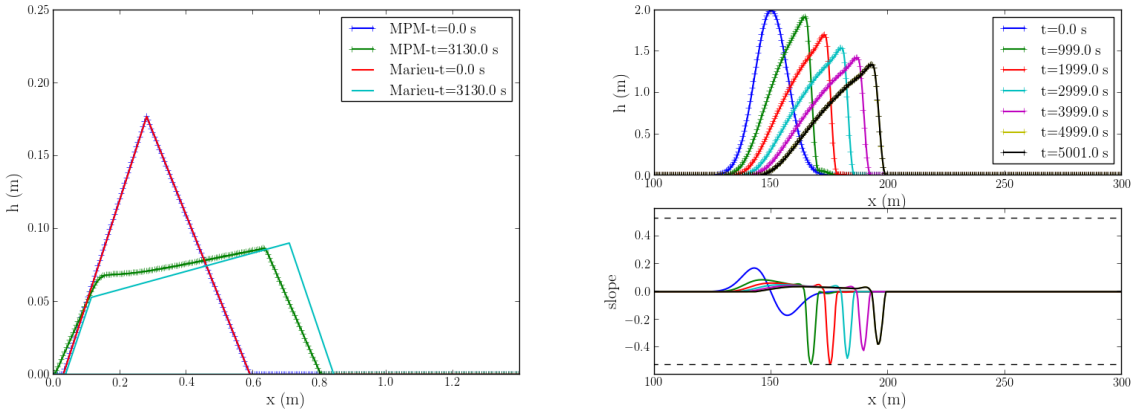
Figure 4.3 – Avalanche management module test on a sand heap (a) and on the dune migration case, with $Q_f = 14$ m^2.s^{-1} (b).

Parameters have the following values: $\rho_s = 2650$ kg.m^3, $\rho = 1025$ kg.m^3, $\nu = 10^{-6}$ m^2.s^{-1}, $d = 0.2$ mm. According to *Liu* [2001], for the parameters set used here, the critical Shields number is equal to 0.052.

4.2 MPM Formulae

For the MPM formula, no analytical solution exists. In order to validate the MPM transport formula, the present model solution is compared with experimental date obtained at LEGI by *Rossi et al.* [2003]. This test case has been proposed by *Marieu* [2007]. In his work Marieu used a flow law based on one dimension SW (Barré de Saint Venant) equations because of the non-uniform flow and empirical sediment transport law based on *Rossi et al.* [2003] results, as the ones compared in figure 4.4 are obtained with H and Q_f fixed and the MPM transport formula. The studied scale, as the dune size, are quite small, with a maximum dune height of 0.177 m and a maximal water depth of 0.297 m. Figure 4.4.a shows that the MPM transport law gives qualitatively closed results compared to the Marieu ones. The sediment motion doesn't occurs everywhere on the dune, which confirms that the morphodynamic module is able to reproduced the threshold sediment motion as described by *Meyer-Peter and Müller* [1948]. For the input flux used here, the local Shields stays always subcritical on the bottom part of the upstream slope, so this part remains steady in the time, showing that, locally, the system is under the transport threshold. The MPM transport law seems to well reproduced a real dune migration.

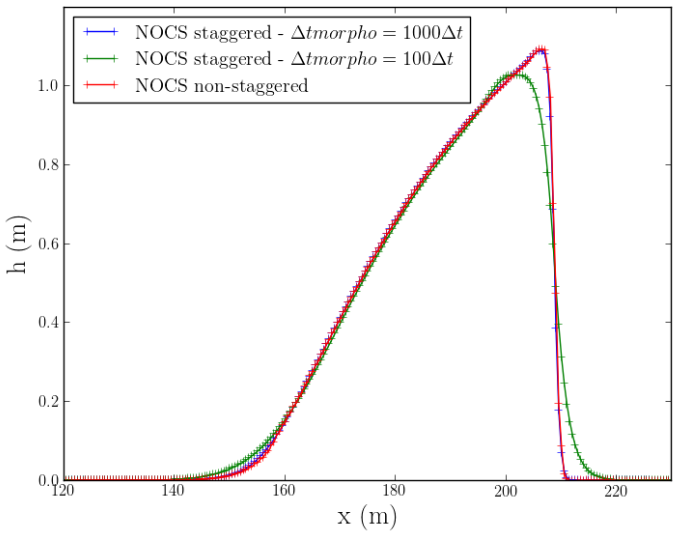
Figure 4.4.b, present the bed elevation profile along the x-direction at different times (upper panel) and the local bed slope along the x-direction (lower panel). The predicted dune migration is similar to the one obtained with the simple bedload formula (section 4.1). Therefore, the implementation of the morphological module with the MPM formula is validated.



(a) (b)

Figure 4.4 – Morphodynamic module tests with MPM formulae for the flume dune case $Q_f = 0.07$ m^2.s^{-1} (a), and the section 4.1 case ($Q_f = 20$ m^2.s^{-1}).

4.3 NOCS Staggered and 2D Extension



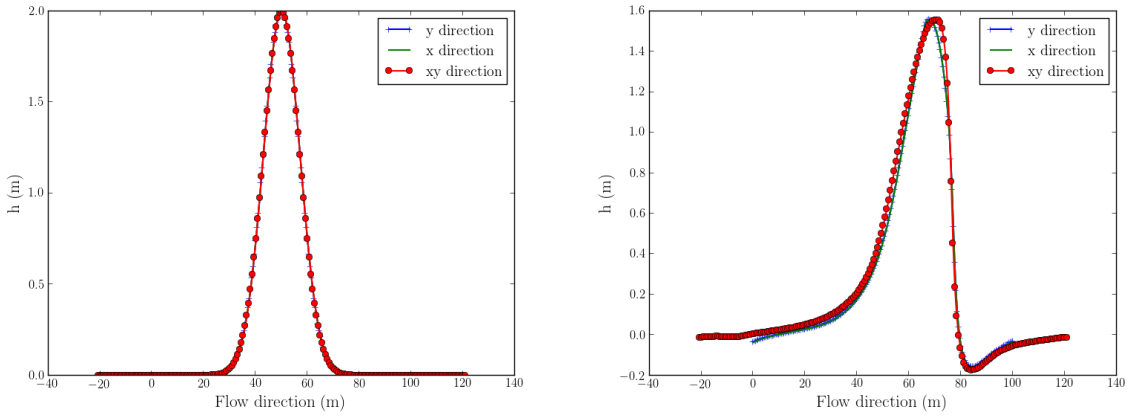
(a)

Figure 4.5 – Comparison between the NOCS staggered and non-staggered scheme for a 1D simulation with an imput flux of 20 m^2.s^{-1} and $\Delta t = 0.02$ s.

All the 1D test cases presented above were done using non-staggered NOCS scheme. As the 2D extension of this scheme was not found in the literature, the 2D morphodynamic model has been implemented based on a staggered scheme. On a 1D case in the x direction, comparisons between both schemes have been done (figure 4.5). As described by *Marieu* [2007], because of the double interpolant projection necessary to obtain the corrector step on the non-staggered grid (see 3.2.2), the staggered scheme is more diffusive than the non-staggered one. Introducing a second time-step Δt_{morpho} and calling the morphodynamic module only when the critical Shields

is exceeded at some location in the grid allows to reduce the NOCS natural numerical diffusion. The choice of the morphodynamic time-step is also important. As the changes in sediment are very slow compared to the oceanic motion, a morphodynamic time step 1000 times higher than the hydrodynamic one appears to be a good compromise.

The 2D extension is validated by comparing the Gaussian dune migration in three different directions (x, y as well as the diagonal $x - y$ direction). Dune migration is induced by an imposed flow discharge. Figure 4.6 shows that the dune migration and deformation is absolutely identical in the x and y directions. Furthermore, the dune behavior in the diagonal directions is almost identical to the axial ones, showing that the cross terms in the morphodynamic model are correctly implemented. Slight variations can be imputed to the TVD estimator and to the fact that sediment fluxes are computed in the x and y direction only and then projected on the diagonal. Nevertheless, variations are small enough to be ignored, and the 2D extension of the NOCS scheme is validated.



(a) (b)

Figure 4.6 – dune motion with the input flux variation direction ($Q_f = 20$ m^2.s^{-1}).

2D plots of the dune motion (figure 4.7) confirm that dune deformation in the flux direction are extremely similar for the three given directions.

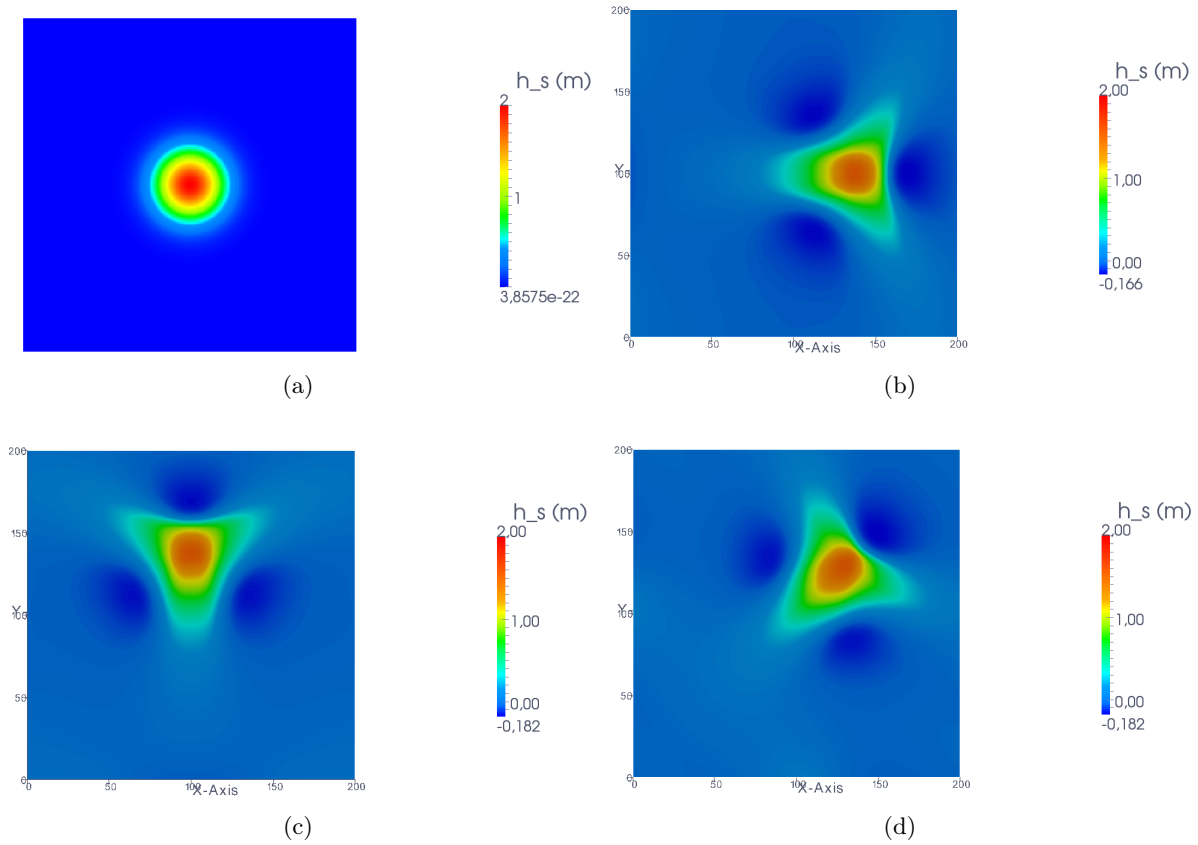
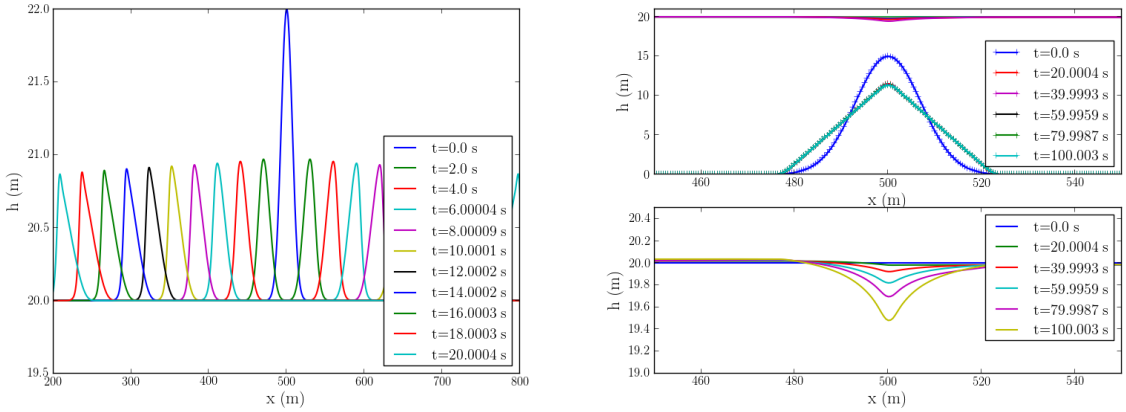


Figure 4.7 – 2D plot dune motion with the input flux variation direction ($Q_f = 20$ m^2.s^{-1}), initial state (a), $x-$direction (b), $y-$direction (c) and $xy-$direction (d).

Chapter 5

Shallow Water Module Validation

The Shallow Water module and its coupling with the morphodynamic module have also to be validated. The classical test case for the SW equations consists in a gravity wave celerity verification. In fact, the speed of the waves is given by $c = \sqrt{gh_o}$, where h_o is the water depth. Perturbation travel at a speed c in the positive or negative x direction (in 1D).
The wave is generated by a bump of the free surface at the initial time (figure 5.1.a). In this test case, the water depth is equal to 20 m, the corresponding waves celerity is $c = 14$ m.s^{-1}, which is the one observed on figure 5.1.a. For this test, the friction coefficient between the sediment bed layer and the ocean layer and the atmospheric forcing are set equal to zero.



(a) (b)

Figure 5.1 – SW module tests for waves celerity (a) and coupling with the morphodynamic module (b).

Figure 5.1.b shows a result of the coupled SW and morphodynamic modules, the free surface is affected by the dune presence. The slight dissymetry of the free surface inflection, compared to the dune shape, is due to the bed friction. More precisely, the gap between the dune shape and the free surface deformation is induced by the drag force.
It can also be observed that the dune slope is grater than the angle of repose at the initial time, so avalanches occur.

A third test case has been done to confirm the accuracy of the shallow water module, it concerns the non linear part of the equation: without the presence of friction, the free surface deformation has to follow the Bernoulli solution (Energy conservation). For a flow with a free

surface the Bernoulli equation can be applied to that free surface, which is a streamline with constant pressure, the equation becomes:

$$h_o + h_s + \frac{u^2}{2g} = const. \iff h_o(x) + h_s(x) + \frac{u(x)^2}{2g} = h_1 + \frac{{u_1}^2}{2g}, \tag{5.1}$$

where h_s is the bed elevation, h_o is the water depth and u is the fluid velocity. Far from the obstacle, the fluid velocity and the water depth are respectively equal to u_1 and h_1.
The free surface deformation induced by the presence of the bedform can be written as:

$$\eta = h_1 - h_o - h_s, \tag{5.2}$$

So from equation (5.1), the free surface deformation η could be rewritten as:

$$\eta = \frac{u^2 - {u_1}^2}{2g}, \tag{5.3}$$

where, for small free surface variations, $u = q/(h_1 - h_s)$ and $u_1 = q/h_1$, with q the flow discharge. So the free surface deformation becomes:

$$\eta = \frac{1}{2g}\left(\frac{q^2}{(h_1 - h_s)^2} - \frac{q^2}{(h_1)^2}\right), \tag{5.4}$$

Periodic boundary conditions imposed in the SW equations have to be changed in this case. As in subcritical free surface hydraulic problem an input flux and an output depth need to be imposed.

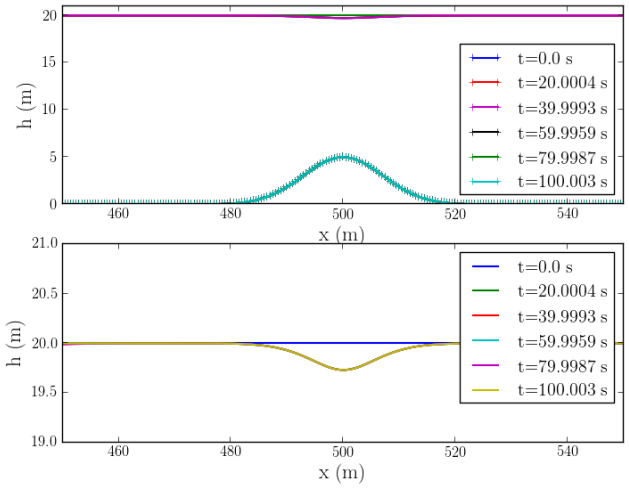


Figure 5.2 – SW module test with no friction and an imposed input flow discharge of $Q_f = 50$ m^2.s^{-1}.

Figure 5.2 shows that the free surface deformation induced by the presence of the bedform follow the Bernoulli solution in a qualitative way, the deflection is symmetric and exactly opposed to the bedform elevation. The deformation is also quantitatively closed to the Bernoulli solution as shown in table 5.1. The maximum free surface deflection values obtained from the numerical model (η_{model}) tends to differ from the solution given by equation 5.3 (η_{approx}) when the input flow rate Q_f is increased. The comparison of numerical results with the analytical solution

Table 5.1 – Maximal values of free surface deflexion for five different input fluxes.

Q_f ($m^2.s^{-1}$)	η_{model}(m)	η_{approx}(m)	error (%)	$\eta_{analytical}$(m)	error (%)
2	$3.966.10^{-4}$	$3.964.10^{-4}$	0	$3.964.10^{-4}$	0
10	$9.941.10^{-3}$	$9.910.10^{-3}$	0.8	$9.941.10^{-3}$	0
20	0.0401	0.0396	1.2	0.0401	0
50	0.2686	0.2477	7.8	0.2686	0
100	1.5377	0.9684	37	1.5384	0.04

shows a very good agreement for all flow discharges. This shows that the non linear terms, non accounted for the approximate solution, becomes dominants as the flow discharge is increased. These non-linear terms are well implemented and solved in the proposed numerical model. Tests have also been carried in two dimensions. For this case, the conservation of the Bernoulli Potential has been checked. From equation 5.1, the Bernoulli Potential has to be constant along the surface streamline, in 2D it can thus been written as:

$$B_{pot} = h_o + h_s + \frac{u^2 + v^2}{2g} = const, \tag{5.5}$$

As shown in figure 5.3.a and 5.3.b, tests have first been done with input fluxes in the x and y directions only for a sandbar (the dune is the same in all the x or y direction), then for a 2D Gaussian sand dune with an imput flux in the x direction (figure 5.3.d). For all these cases, a steady state is represented, the simulation time is t=1000 seconds. The Bernoulli potential is recovered in all the three configurations, even if, as shown on figure 5.3.d, small variations can be observed. These variations are acceptable with a maximal relative error of 0.015%.

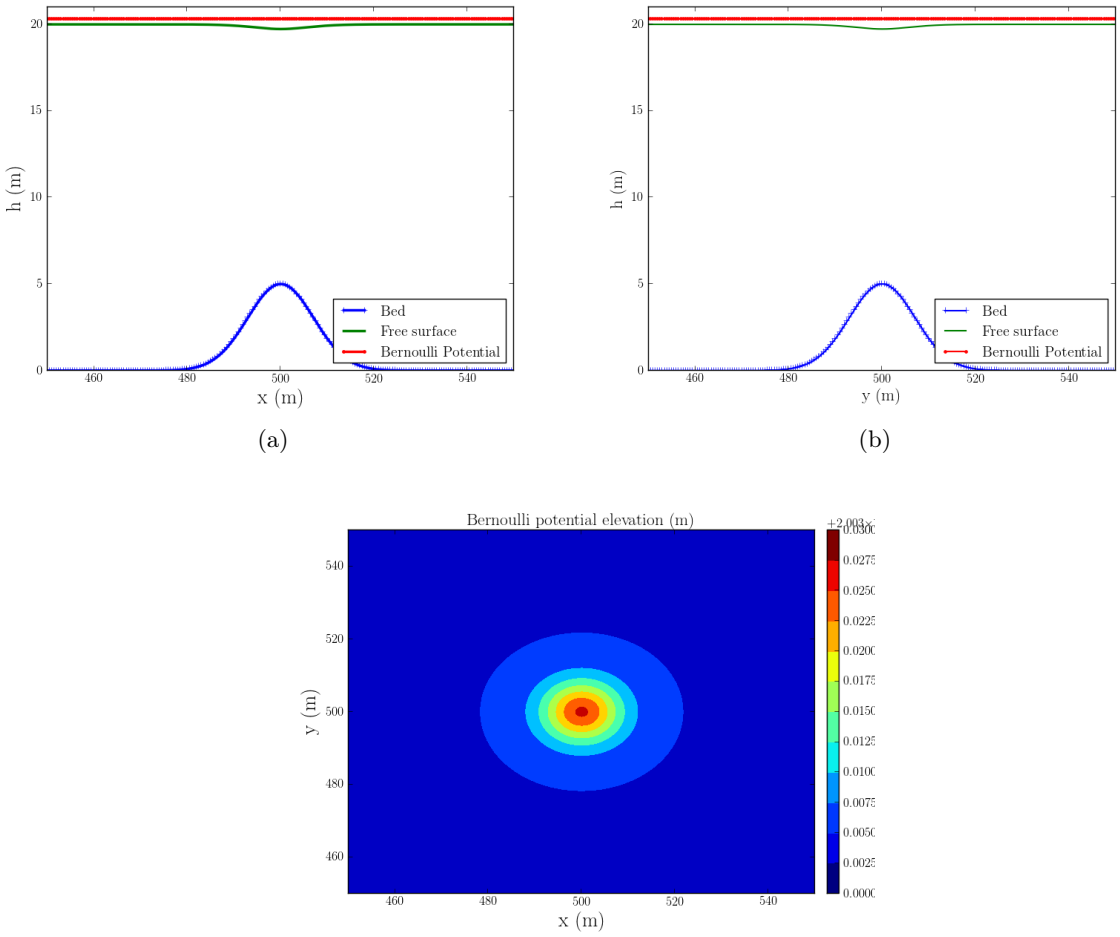


Figure 5.3 – Bernoulli potential conservation test in the x (a) and y (b) directions. Bernoulli potential (c) for a 2D gaussian sand dune with a input fluxe in the x direction only.

Chapter 6

Wake Effects

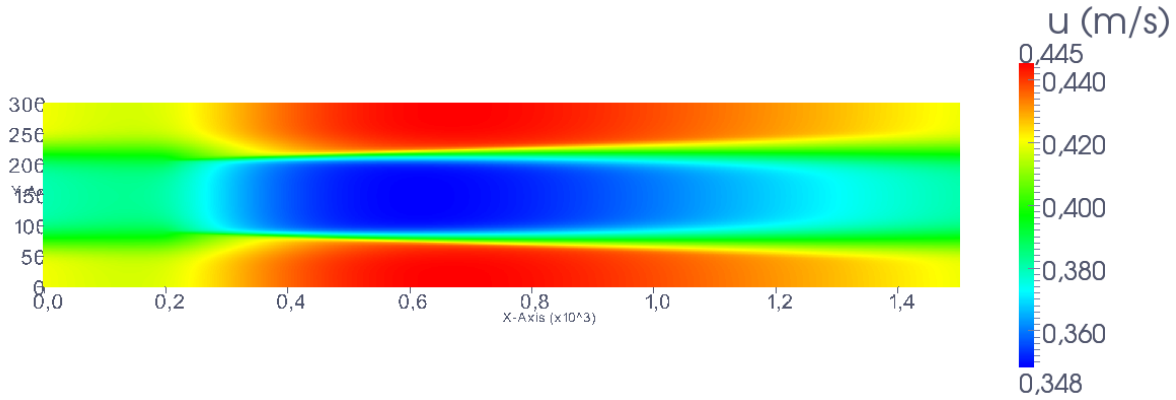
All the numerical simulations presented in this work involve the same wake parameters, such as rotor diameter or wind velocity. The rotor diameter is small ($D = 40$ m) in order to reduce the domain size and the computation time. Forcing wind has been chosen as 20 m.s^{-1}, corresponding to the high range part of turbine good working (from 3 m.s^{-1} to 30 m.s^{-1}). Concerning the time-step, $\Delta t_{morpho} = 1000\Delta t$ for all the simulations. In order to respect the CFL condition, when water depth H is equal to 6 meters, $\Delta t = 0.02$ s and when $H = 20$ m, $\Delta t = 0.01$ s. Simulations with different horizontal viscosity ν, water depth and a large scale flow of varying strength and angle have been performed. An overview of the different simulations parameters is given in table 6.1:

Table 6.1 – Wake simulations and their parameters.

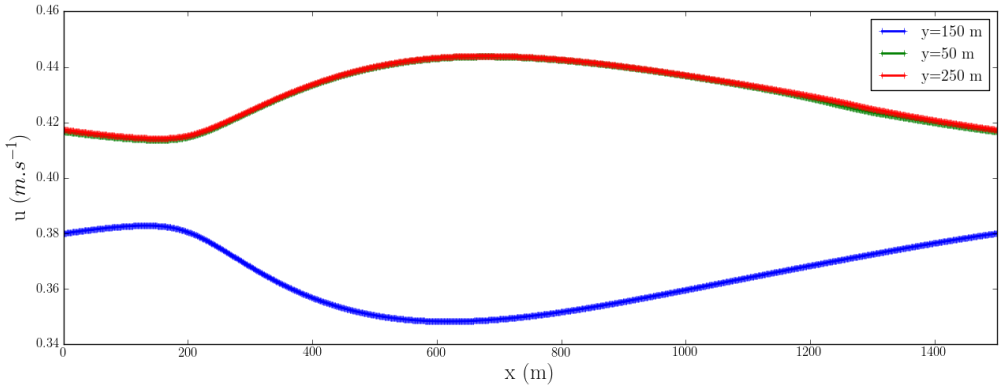
Simulation name	**Viscosity ($m^2.s^{-1}$)**	L_x **(m)**	L_y **(m)**	**Water depth (m)**	**Turbulent**
H6NU5E-2	5.10^{-2}	1500	300	6	No
H6NU1E-2	1.10^{-2}	1500	300	6	Yes
H6NU5E-3	5.10^{-3}	1500	300	6	Yes
H6NU1E-3	1.10^{-3}	1500	300	6	Yes
H20NU5E-3	5.10^{-3}	1500	300	20	Yes

6.1 Wake Effects on the Ocean Free Surface

As shown on figure 6.1.a, for simulation H6NU5E-2, the wake creates a velocity decrease in the ocean, the perturbation intensity is maximum a few hundred meters after the wake impact and decreases downstream. Outside the wake, the flow is not subjected to the velocity reduction and a strong velocity gradient is generated, across the wake boundaries, at the transition between the perturbed and the unperturbed zones. The forcing is symmetric with the $y = 150$ m plane (figure 6.1.b), periodic boundary conditions are used. Indeed, in the results presented here, the study domain is surrounded by an infinity of other identical domains in both x and y directions. This is particularly conspicuous in the x direction, were the wake that is coming out of the domain at the right boundary reappears at the left one. The wake present in the study domain is thus perturbed by an upstream wind wake.
Simulation H6NU5E-2 is taken as the referent one, corresponding to a laminar wake case.



(a)



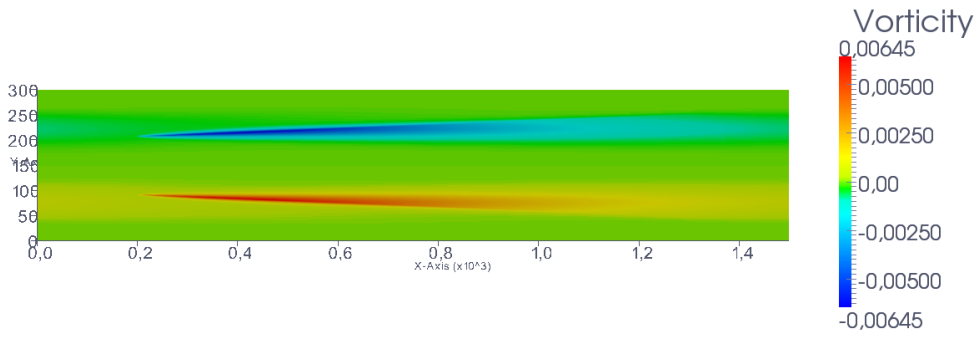
(b)

Figure 6.1 – 2D x-velocity field (a) and 1D transect in the x direction (b) for a simulation time $t = 50$ hours.

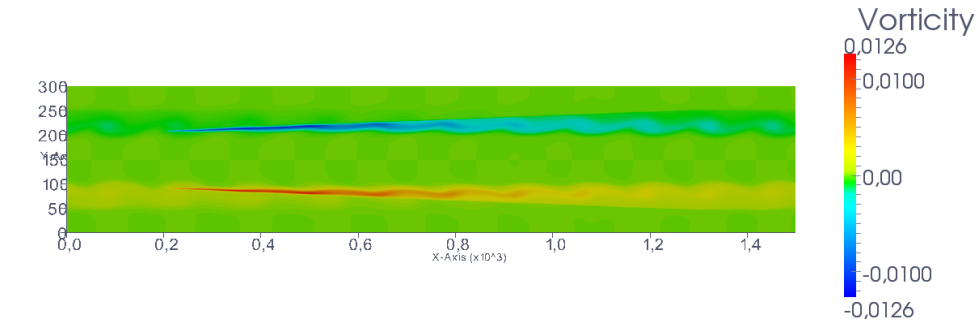
Decreasing the ocean viscosity leads to instabilities in the wake, the ocean flow becomes turbulent, starting from a vortex formation at the wake boundaries. Figure 6.2 shows the 2D vorticity field at the same simulation time and for different ocean viscosities. While the viscosity is decreased, the vorticity anomalies of the flow are increased, *i.e* the vortex intensity becomes stronger. Decreasing the viscosity leads thus to the appearance of a Kelvin-Helmhotz instability. Here, this instability is induced by a strong velocity shear in the ocean layer, inside and outside the wake, generating strong vorticity.

Figure 6.3 shows that for a water depth equal to 20 m, vorticity picture is completely different to the 6 m case. For $h = 20$ m, vortex spacing is higher, approx. 400 m against approx. 100 m for $h = 6$ m and the size of the vortices is higher too. Contrary to the 6 m water depth case, the two vortex streets interact, generating a broader turbulent domain. An increase of approximately a factor 4 in water depth leads to a change of the same order in the vortex spacing. It appears that vortex size and spacing seems to depends on the water depth. This water depth appears in the bottom friction terms $G = \tau_b/h$ in the SW equations. As the simulations are undertaken at a imposed stress, the bottom shear stress doesn't depend on the water depth. The influence of the bottom friction force is thus decreasing when the water depth is increasing, allowing for stronger instabilities.

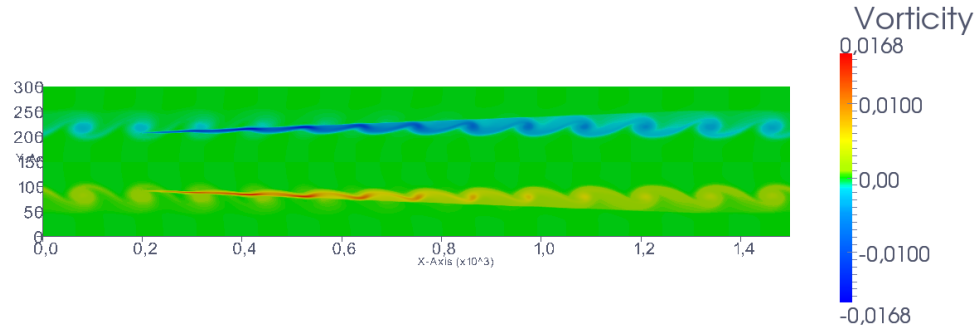
The water depth and the bottom friction seems to be the controlled parameters for the



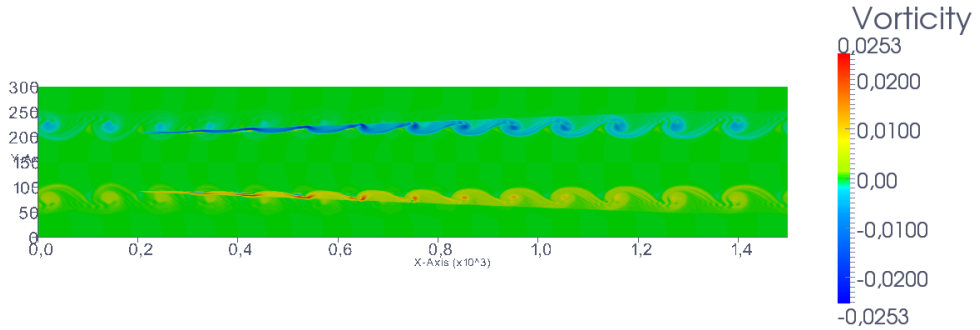
(a)



(b)



(c)



(d)

Figure 6.2 – 2D vorticity field for H6NU5E-2 (a), H6NU1E-2 (b), H6NU5E-3 (c) and H6NU1E-3 (d). Viscosity is decreasing from top to the bottom. Simulation time is the same for each snapshot, $t = 50$ hours.

size and the spacing between vortices. Nevertheless, to the best of our knowledge, no study has been done on Kelvin-Helmhotz instability formation in presence of bottom friction. As more and more deep water wind farms are planned [*EWEA*, 2013], a better understanding of the local impact on the circulation is important. An analytical development could be undertaken in order to give insights into this open question.

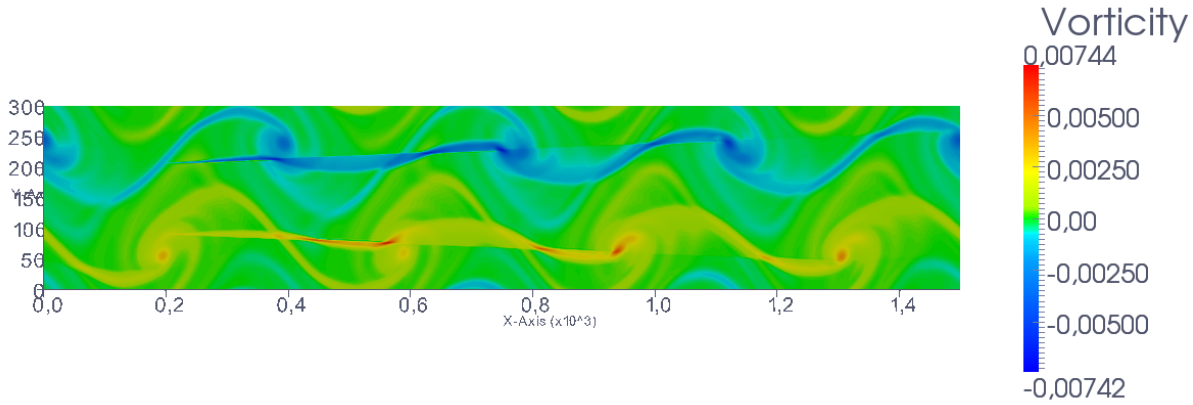


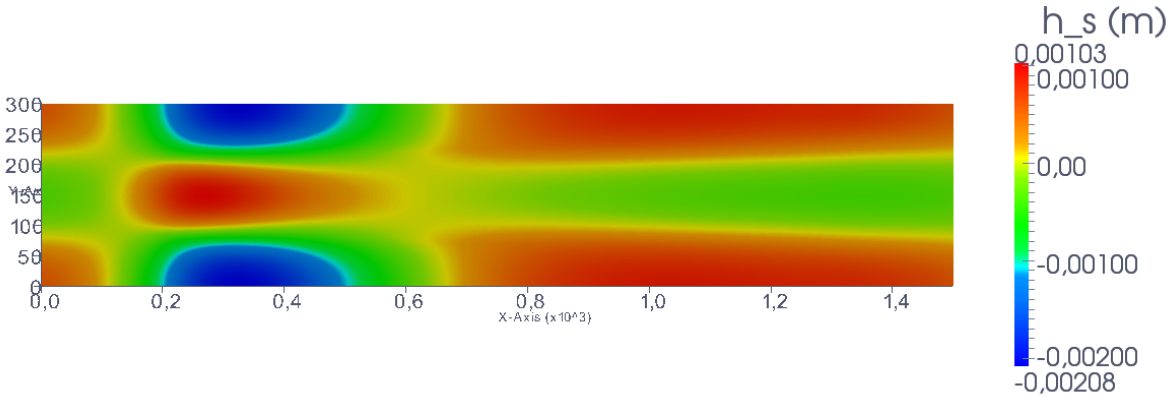
Figure 6.3 – 2D vorticity field for H20NU5E-3. Simulation time is $t = 50$ hours.

6.2 Wake Effects on the Seabed

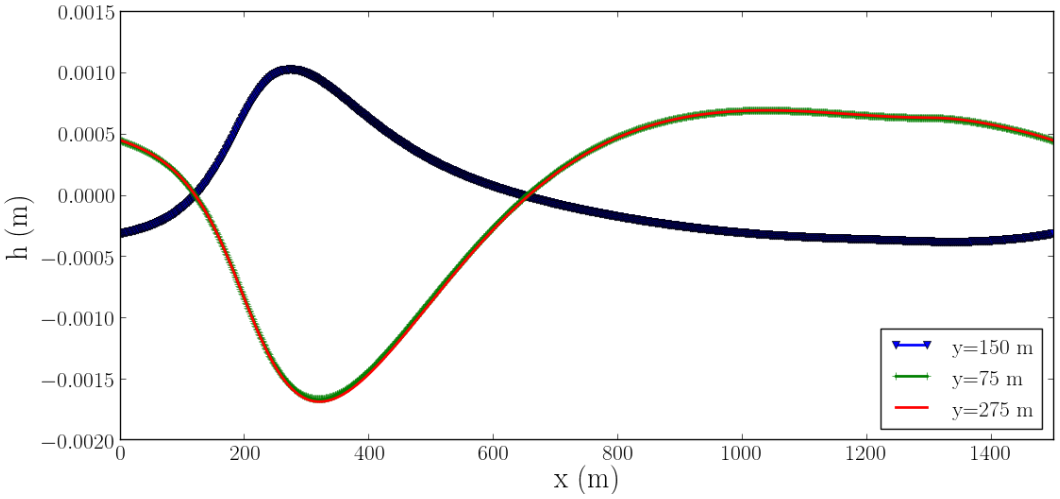
6.2.1 Spatial Evolution

For all the simulations undertaken, spatial qualitative impact on the wake upon the seabed is similar. Figure 6.4 shows the impact for the laminar (H6NU5E-2) case. The velocity deficit induced by the wake leads to a sediment accumulation, *i.e* a dune formation close to the impact abscissa (between $x = 200$ and $x = 300$ m). Downstream in the wake, after the $x = 650$ m abscissa point, the velocity deficit becomes less important, corresponding to a bottom shear stress that induces bed erosion. Outside the wake, the flow accelerates by bypassing the dune, such acceleration increases the bottom shear stress and thus the sediment transport, leading to a hole formation on each side of the dune. The hole depth is much important than the dune height (0.0017 m against 0.0010 m for the dune) after 80 hours of simulation. After $x = 650$ m, the velocity and the bottom shear stress decrease along the x-direction, producing sediment accumulation in this region.
The dune width corresponds to the local wake width, showing that the local velocity deficit induced by the wake is the main cause of the seabed evolution. Sediment deposition is also much more important because of the periodic boundary conditions. Sediment transport inside the wake which is coming out at the right boundary reappears at the left one, increasing the amount of sediment available to be deposited in the velocity deficit zone.

The dune shape is inversed compared to the classical one, here a strong slope appears upstream and a smooth slope downstream. The velocity deficit generated by the wake just after the impact point is strong enough to induce a sharp sediment drop off. This deposit maximum occurs at the dune upstream position, where the longitudinal velocity gradient is the stronger.



(a)

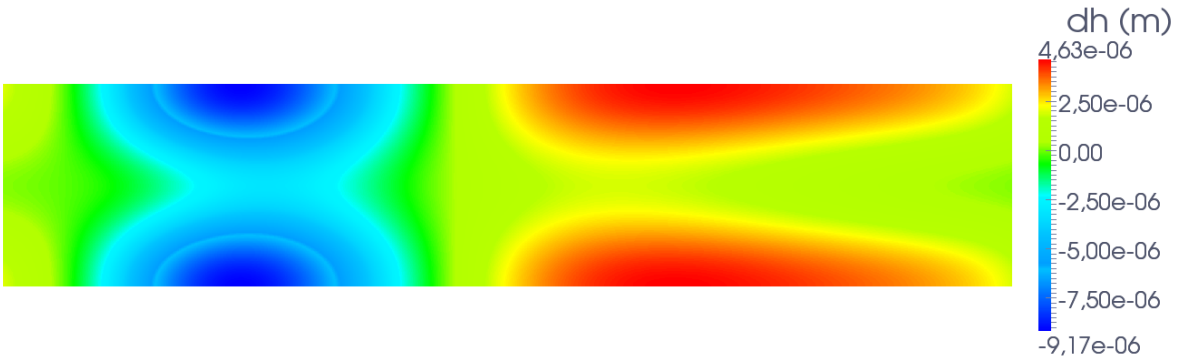


(b)

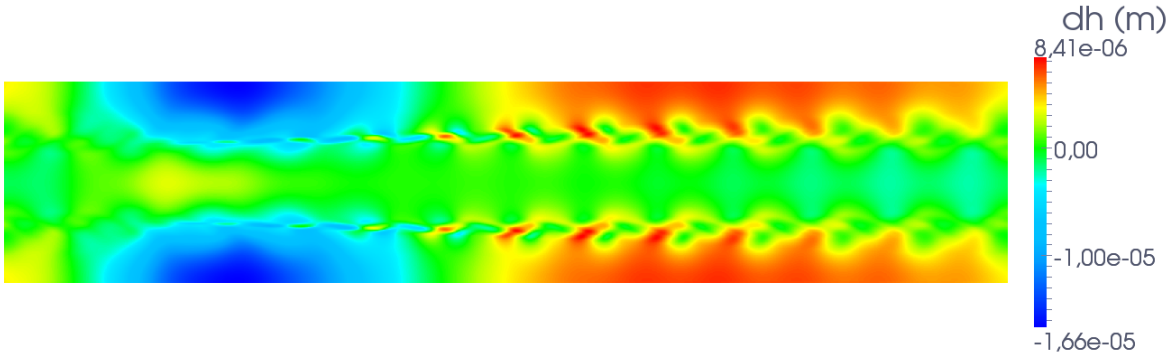
Figure 6.4 – 2D seabed height field (a) and x-direction transect (b) for $H6NU5E-2$. Simulation time is $t = 80$ hours.

6.2.2 Time Evolution

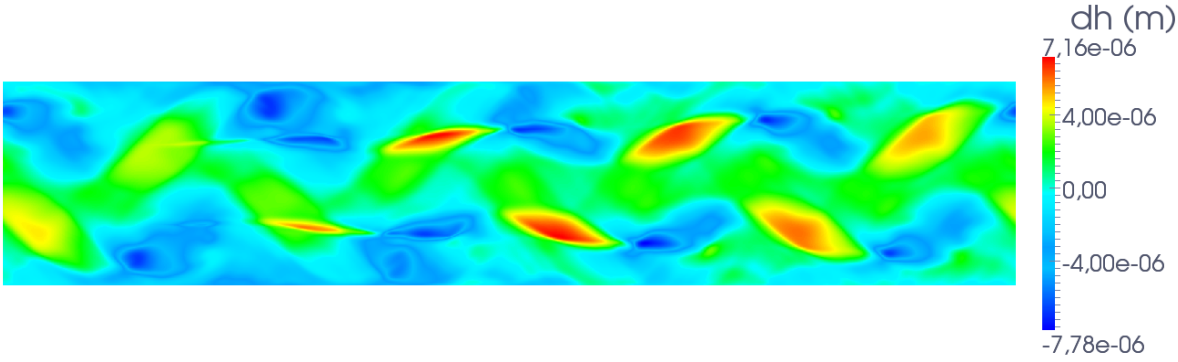
Figure 6.5 shows that the bathymetry variation between two consecutive output ($\Delta t = 1800$ s) of the same simulation. The morphodynamical evolution between turbulent and laminar cases is completely different. The vortices strongly affect seabed morphodynamics. For the 6 m water depth case, the wake boundary imprint in the seabed becomes unstable and for the 20 m water depth case, the wakes imprint is totally annihilate by large scale vortices. Difference in vortex scale observed in the ocean is found again in the seabed, showing that vortex formation on the free surface has a significant impact on the seabed morphodynamics. However, even if this impact appears clearly, its intensity remains quite small. Indeed, the order of change in height correspond to 2.10^{-6} m for an hour. Changes of the order of the mm will only appear after one month of simulation. Long term simulations should be carried out in order to better modelize the morphodynamic impact of the ocean vortices.



(a)



(b)

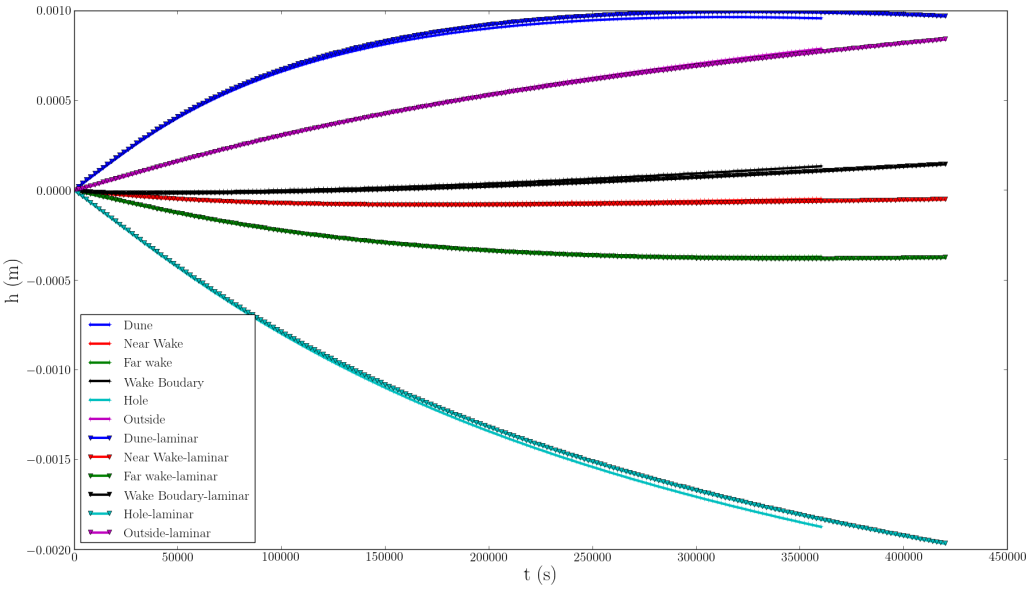


(c)

Figure 6.5 – 2D bathymetry difference field between two consecutive output for H6NU5E-2 (a), H6NU5E-3 (b) and H20NU5E-3 (c) after 70 hours of simulation.

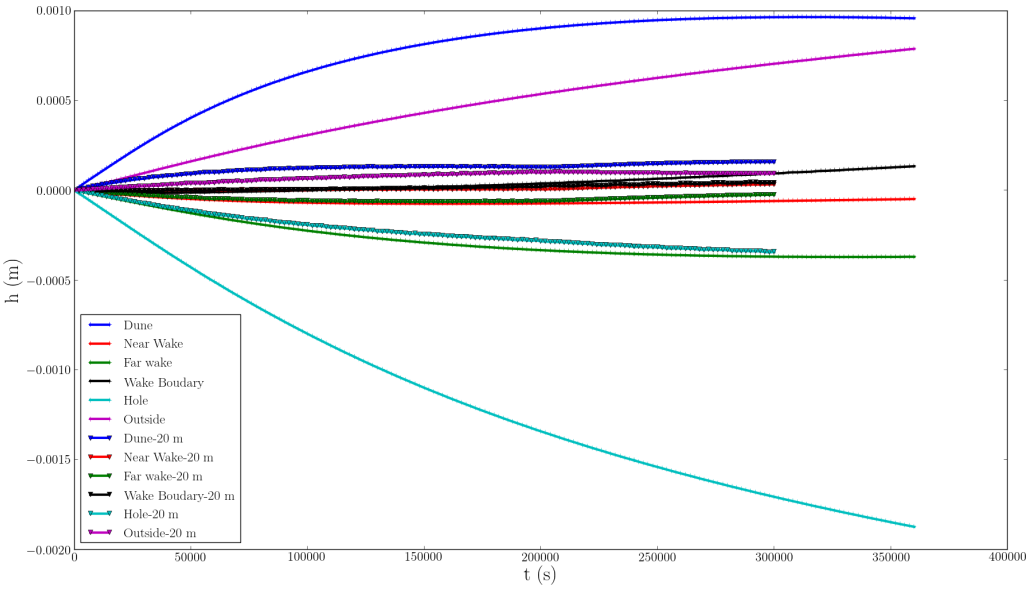
Changes of seabed evolution in specific seabed points for H6NU5E-2, H6NU5E-3 and H20NU5E-3 are presented on figures 6.6 and 6.7. A first important result is that, even for the laminar case, seabed evolution is not linear in time. A very common method to increase time simulation in morphological numerical model consists in using a morphological prefactor: sediment fluxes are multiplied by a prefactor χ ($\chi >> 1$) so that for a given simulation time correspond finally to χ times this simulation time. Because of the non-linear seabed behavior observed herein such method seems to be not justified. For a given xate depth ($H = 6$ m), the seabed evolution is not affected by the vortices at short-time. However, for a greater water depth ($H = 20$ m), the

seabed evolution is much lower (one order of magnitude), even if the instantaneous bed shear stress magnitude and the Shields number are similar in both cases. This difference in the sediment transport can be explained by the homogeneous turbulence for the 20 m water depth case. This turbulence induces a non local relation between the momentum input and the momentum outtake, positive and negative vorticity alternate in time resulting in an average transport close to zero. In conclusion, the seabed pattern is similar for both water depth but the magnitude of this disturbances is one order of magnitude lower for the 20 m water depth case. This result is extremely important, showing that the sediment transport physic induced by the wake presence seems to be completely different depending on the water depth. This recent and astonishing result needs further confirmations.



(a)

Figure 6.6 – Time evolution of specific seabed points, comparison between H6NU5E-2 and H6NU5E-3. Maximal simulation time is equal to 116 hours, *i.e* ≈4.5 days.



(a)

Figure 6.7 – Time evolution of specific seabed points, and between H6NU5E-3 and H20NU5E-3. Maximal simulation time is equal to 100 hours, *i.e* ≈4 days.

Chapter 7

Conclusion and Perspectives

During this internship an idealized 2D numerical model has been proposed to study the impact of an offshore wind turbine wake on the ocean and sediment dynamics. To the best of our knowledges no study has been done on this subject.
A simplified physical model has been proposed and a mathematical model has been consistently built. My main task during the internship has been to write a numerical model, starting from nothing. An important effort has been dedicated to the careful validation of the sediment transport and the hydrodynamic modules. Classical test cases and comparisons with analytical solution have been undertaken, showing the model accuracy and robustness.

The present results show that the turbine wake has an impact on both ocean and sediment bed layers. Turbine wake impact on ocean surface can generates Kelvin-Helmholtz instabilities and vortex streets formation. Size and spacing between these vortices seems to be controlled by the friction between the bed and the ocean layer. When friction is decreased, large scale instabilities are easily generated, leading to an homogeneous turbulence state in the ocean. As the in model the wind stress is imposed such state appears when water depth increases. Two morphodynamics evolutions are noticeable, in the short-time (days), main effects on the seabed are linked to the mean wake and not to its turbulent or laminar nature. In the long term, change in seabed elevation could be dominated by the turbulent nature of the wake. When the turbulence is homogeneous long-term and short-term impacts on the wake seems to be dominated by the turbulence. At short-term, the homogeneous turbulence decreases the average sediment transport. Wake imprint in the seabed is thus quantitatively lower than less turbulent or laminar cases. Therefore, when the water depth is increased, the morphological impact are lower and the turbulent ocean dynamic is increased. This phenomenon can have a significant feed back on the atmospheric dynamic, that should be taken into account as more and more offshore wind farms are planned at important water depth.

The present numerical model shows very important and promising results. Nevertheless, many improvements have to be implemented into the numerical model. If the addition of an large scale oceanic forcing is under construction, an other important point would be, as mentioned above, to consider the atmospheric dynamic differently by introducing a shallow water module for the atmosphere. Furthermore, the wake model used herein is extremely simplified, using more realistic model such as the parabolic Larsen's one would provide more realistic simulations. As impact of turbine wake on the ocean is, as far as we know, still understudied, data assimilation from natural sites are essential in order to better parametrize the numerical models.

Bibliography

Barbe, J. B., *Etude de l'impact des eoliennes offshores sur l'ocean et l'atmosphere - Internship report*, 2013.

Buffington, J. M., The legend of af shields, *Journal of Hydraulic Engineering*, *125*(4), 376–387, 1999.

EWEA, *Deep water, the next step for offshore wind energy*, 2013.

Jacobson, M. Z., C. L. Archer, and W. Kempton, Taming hurricanes with arrays of offshore wind turbines, *Nature Climate Change*, *4*(3), 195–200, 2014.

Jensen, N., *A note on wind generator interaction*, 1983.

Jiang, G.-S., D. Levy, C.-T. Lin, S. Osher, and E. Tadmor, High-resolution nonoscillatory central schemes with nonstaggered grids for hyperbolic conservation laws, *SIAM Journal on Numerical Analysis*, *35*(6), 2147–2168, 1998.

Larsen, G., J. Hojstrup, and H. Madsen, Wind fields in wakes, in *EWEC 1996 Proceedings, Goteborg (Sweden)*, 1996.

LeVeque, R. J., *Finite volume methods for hyperbolic problems*, vol. 31, Cambridge university press, 2002.

Liu, Z., Sediment transport, *Laboratoriet for Hydraulik og Havnebygning, Institut for Vand, Jord og Miljøteknik, Aalborg Universitet*, 2001.

Marieu, V., Modélisation de la dynamique des rides sédimentaires générées par les vagues, Ph.D. thesis, Bordeaux 1, 2007.

Meyer-Peter, E., and R. Müller, Formula for the bedload transport, in *3rd Meeting of the International Association of Hydraulic Research*, 1948.

Moulin, A., *Air-sea interaction at the synoptic and the meso-scale - Internship report*, 2012.

Nessyahu, H., and E. Tadmor, Non-oscillatory central differencing for hyperbolic conservation laws, *Journal of computational physics*, *87*(2), 408–463, 1990.

Renkema, D. J., Validation of wind turbine wake models, *Master of Science Thesis, Delft University of Technology*, 2007.

Rossi, L., H. Michallet, P. Bonneton, et al., Morphodynamique d'une dune, couplage avec l'écoulement, *Acte du Congrès Français de Sédimentologie*, 2003.

Sánchez, A., and W. Wu, A non-equilibrium sediment transport model for coastal inlets and navigation channels, *Journal of Coastal Research*, pp. 39–48, 2011.

Smith, S., Coefficients for sea surface wind stress, heat flux, and wind profiles as a function of wind speed and temperature, *Journal of Geophysical Research: Oceans (1978–2012)*, *93*(C12), 15,467–15,472, 1988.

Van der Veen, H. H., S. Hulscher, and B. Perez Lapena, Seabed morphodynamics due to offshore wind farms, in *River, Coastal and Estuarine Morphodynamics: RCEM*, 2007.

Vautard, R., F. Thais, I. Tobin, F.-M. Bréon, J.-G. Devezeaux de Lavergne, A. Colette, P. Yiou, and P. M. Ruti, Regional climate model simulations indicate limited climatic impacts by operational and planned european wind farms, *Nature communications*, *5*, 2014.

Vermeer, L., J. N. Sørensen, and A. Crespo, Wind turbine wake aerodynamics, *Progress in aerospace sciences*, *39*(6), 467–510, 2003.

Wu, J., Wind-stress coefficients over sea surface from breeze to hurricane, *Journal of Geophysical Research: Oceans (1978–2012)*, *87*(C12), 9704–9706, 1982.

List of Figures

Appendix A

Numerical Model in Fortran

```
program main

  !*************************************************************************
  ! Program by Tim NAGEL - LEGI                                            *
  !                                                                        *
  ! date: 31/03/2014                                                       *
  !                                                                        *
  ! This program tends to modelise a seabed evolution in time, by solving Exner *
  ! equation coupled with the SW equation for the ocean. Equations are solved   *
  ! in 2D                                                                  *
  !                                                                        *
  !   All units sgi                                                        *
  !*************************************************************************

  use netcdf
  implicit none
  include 'common.inc'

 !variables definitions
  real(8) :: t                        !time
  real(8) :: tf                       !simulation final time
  real(8) :: phi,maxslope,maxutest,maxeta,maxs,theta_c,tramp,q0max

  integer :: i,j,iprint !elements and iterations number

  character(len=80):: outputfile
  character(len=80):: restartfile

  real(8), dimension(nx) :: x,cfl
  real(8), dimension(ny) :: y
  real(8), dimension(nx,ny) :: u,v,s,u_c,u_test,s_b,eta,eta_c,h,hmean,h_n,theta,v_c,
      u_dip,v_dip,bpot
  real(8), dimension(nx,ny) :: u_half,v_half,qbx,qby,qbx1,qby1,sigma,sigma_a,W,Wu,omega
  real(8), dimension(nx) :: c,u_abs
  real(8) :: maxc,maxhn,maxh,maxtheta,maxv,M,Wxmax,Wymax,maxsigma,W0,dx1,dy1,maxu
  real(8) :: delta_t,t_iter,va0
  CHARACTER(len=4),dimension(1) :: restart_option
  CHARACTER(len=10) :: parm
  logical :: restart
!*********************************************************************
!                Restart option
!*********************************************************************
call getarg(1,parm)
read(parm,*)restart_option(1)
if (restart_option(1)=="RES") then
 restart=.true.
 print *, "Restart=true"
elseif (restart_option(1)=="NORE") then
 irestart_name=0
 restart=.false.
 print *, "Restart=flase"
else
 print *, "Reminber: Arg1 = NORE/RES (NORE=NoRestart, RES=Restart)"
endif
!*********************************************************************
!                Read the input file
!*********************************************************************
  open (12,file='input_morpho.txt')
  read (12,*)  !skip a line
  read (12,*)  !skip a line
  read (12,*) !calculation parametrisation
  read (12,*) dt
  read (12,*) nmorpho
  read (12,*) niter
  read (12,*) nprint
  read (12,*) !outputfile name
```

```
  read (12,"(a80)") outputfile
  read (12,"(a80)") restartfile
  read (12,*)  !domain parametrisation
  read (12,*) xlim
  read (12,*) ylim
  read (12,*)  !ocean
  read (12,*) rho
  read (12,*) h0
  read (12,*) nu
  read (12,*) cd1
  read (12,*) Wx
  read (12,*) Wy
  read (12,*) q0x
  read (12,*) q0y
  read (12,*) x0_oc
  read (12,*) y0_oc
  read (12,*) amp_oc
  read (12,*) kappa
  read (12,*)  !sediment
  read (12,*) rho_s
  read (12,*) d
  read (12,*) x0
  read (12,*) y0
  read (12,*) amp
  read (12,*) cd
  read (12,*) alpha
  read (12,*) beta
  read (12,*) slope_lim
  read (12,*) slope_lim_post
  read (12,*) !wake and turbines
  read (12,*) kj
  read (12,*) dr
  read (12,*) ht
  read (12,*) ua0
  read (12,*) drag
  read (12,*) rho_a
  read (12,*) !numerical scheme
  read (12,*) ibedflux
  read (12,*) ischeme
  read (12,*) icase
  read (12,*) icase_oc
  read (12,*) iav
  read (12,*) lim
  read (12,*) ibc
  read (12,*) ibcew
  read (12,*) ibcns
  read (12,*) idip
  read (12,*) iwake
  close(12)

  outputfile=trim(outputfile)//'.nc'
  print *,'Fichier de sortie -->',outputfile


  dx=xlim/float(nx-1)   !space grid size
  dy=ylim/float(ny-1)   !space grid size
  tf=float(niter)*dt   !final time calculation
  dtmorpho=nmorpho*dt          !morphodynamic timescale
  tramp=1000.
  pi=3.1415926
  maxu=maxval(u)
!**********************************************************************
!                creation of the x and y abcsisse gridspaces
!**********************************************************************
  !x
  x(1)=0
  do i=2,nx
     x(i)=x(i-1)+dx
  enddo
  !y
  y(1)=0
  do j=2,ny
     y(j)=y(j-1)+dy
  enddo
!**********************************************************************
!       creation of the initial bedform and free surface elevation
!**********************************************************************
  call init(x,y,h,eta)
  !! read the restart file here : t,h0,x,dx,u,eta,h
!**********************************************************************
!                        initial conditions
!**********************************************************************
  hmean=h0
  s=hmean+eta
  if (ibc==1) then
     u=(q0x/(s-h))!*cos(pi/4)
     v=(q0y/(s-h))!*sin(pi/4)
  else
     u=0.
```

```
      v=0.
      eta=0.
      h=0.
   endif
   u_test=0
   va0=0.
   u= sqrt((rho_a/rho)*(cd1/cd))*ua0!-(0.1)!*(1.41421356237/2)
  ! u= (1.0)*(1.41421356237/2)
  ! v= sqrt((rho_a/rho)*(cd1/cd))*va0+(0.05)*(1.41421356237/2)
   t=0     ! initial time
   omega=0 !Vorticity initialisation
   iter=0  !time loop initialization
   iprint=1
   delta_t=0
!*********************************************************************
!                Read Restart
!*********************************************************************
if (restart) then
    Call restart_sub(1,t,eta,u,v,h) !read the last state
    tf=t+tf
endif
print *, 't', t, 'maxu', u(10,10), 'maxv', v(10,10)
!*********************************************************************
!                        Atmospheric forcing
!*********************************************************************
   call atm_forcing(x,y,sigma_a)
!*********************************************************************
!              Bpot calculation: Bernoulli potential
!*********************************************************************
   bpot=s+(u**2+v**2)/(2*g)
!*********************************************************************
!                       Netcdf creation
!*********************************************************************
   call netcdf_creation_ocean(outputfile,x,y,h,s,u,v,t,bpot,W,Wu,omega,eta)
!if (.not. restart) then
   call netcdf_save_ocean(t,h,s,u,v,iprint,bpot,W,Wu,omega,eta)
!endif
!*********************************************************************
!                       Time loop
!*********************************************************************
   do while (t.LE.tf)
      !time and iteration advancing
      t=t+dt
      iter=iter+1
      if  (mod(iter,nmorpho)== 0) then
         delta_t=t-t_iter
         print *,'t=',t,'s - iter=',iter
        ! print *,'dt=',dt
        ! write(99,'(2(1x,i8),3(1x,e15.8))') iter, iprint, t, dt, delta_t
         t_iter=t
      endif

     !Atmospheric stress ramp
     ! if (t.le.tramp) then
     !    sigma=(t/tramp)*sigma_a
     ! else
         sigma=sigma_a
     ! endif

     ! Wu=(sigma/(rho*(hmean-h+eta)))

      !Ocean SW calculation
      call ocean(u,v,eta,hmean,h,sigma,u_c,v_c,eta_c,s)

      maxeta=maxval(abs(eta))
      if  (mod(iter,nmorpho-1)== 0) then
          call sedimentary_fluxes(u,v,qbx,qby)
          qbx1=qbx
          qby1=qby
      endif
      !Seabed morphodynamic
      theta_c=0.052e0
      if  (mod(iter,nmorpho)== 0) then
         do j=1,ny
            do i=1,nx
               theta(i,j)=0.5*rho*cd*(u(i,j)**2+v(i,j)**2)/((rho_s-rho)*g*d)
            enddo
         enddo
         maxtheta=maxval(theta)
        ! print *,'maxtheta', maxtheta
         if (maxtheta.ge.theta_c) then
            call morpho(eta,hmean,u,v,qbx1,qby1,h,h_n)
            h=h_n
         endif
         !mass conservation
         M=sum(h)*dx*dy
       !  print *,'Mass', M
      endif

      !updating h, u and eta
      u=u_c
      eta=eta_c
```

```
      v=v_c
      s=hmean+eta

      !Vorticity calculation
      dx1=dx**(-1)
      dy1=dy**(-1)
      do i=2,nx-1
         do j=2,ny-1
            omega(i,j)=0.5*dx1*(v(i+1,j)-v(i-1,j))-0.5*dy1*(u(i,j+1)-u(i,j-1))
         enddo
      enddo
      !BC
      do i=1,nx
         omega(i,1)=omega(i,ny-1)
         omega(i,ny)=omega(i,2)
      enddo
      do j=1,ny
         omega(1,j)=omega(nx-1,j)
         omega(nx,j)=omega(2,j)
      enddo

      !Bpot calculation: Bernoulli potential
      bpot=s+(u**2+v**2)/(2*g)

      maxeta=maxval(abs(eta))

      !save solution every nprint iteration
      if (mod(iter,nprint)== 0) then
         iprint=iprint+1
         call netcdf_save_ocean(t,h,s,u,v,iprint,bpot,W,Wu,omega,eta)
         print *, 't', t, 'maxu', u(10,10), 'maxv', v(10,10)
        ! call netcdf_close_ocean
      endif
  enddo
!**********************************************************************
!                  Write Restart
!**********************************************************************
irestart_name=irestart_name+1
Call restart_sub(2,t,eta,u,v,h)
!**********************************************************************
!                 Close the netcdf file
!**********************************************************************
call netcdf_close_ocean

print *, 't', t, 'maxu', u(10,10), 'maxv', v(10,10)
end program main
```

```
 !************************************************************************
! Program by Tim NAGEL - LEGI                                           *
!                                                                       *
! date: 28/02/2014                                                      *
!                                                                       *
! This subroutine computes initial conditions for the morphodynamic model, and*
! the free surface elevation. Choice is done in the input_morpho.txt file.    *
!                                                                       *
!   All units sgi                                                       *
!************************************************************************
subroutine init(x,y,h,eta)

implicit none
include 'common.inc'

!integer,intent(in) :: icase
real(8),dimension(nx),intent(in) :: x
real(8),dimension(ny),intent(in) :: y
real(8),dimension(nx,ny),intent(out) :: h,eta
real(8), parameter :: sigma = 1.5
real(8) :: x1,x2,x3,h1,h2,h3
integer :: i,j

x1=0.03
x2=0.28
x3=0.59
h1=0.
h2=0.177
h3=0.
!**********************************************************************
!                creation test case 0: slump test of a sand heap
!**********************************************************************
if (icase==0) then
   do i=1,nx
      if (x(i).ge.x1.and.x(i).le.x2) then
         h(i,j)=(x(i)-x1)*(h2/(x2-x1))
      elseif (x(i).ge.x2.and.x(i).le.x3) then
         h(i,j)=h2*(1-((x(i)-x2)/(x3-x2)))
      else
         h(i,j)=0
      endif
    enddo
endif
!**********************************************************************
```

```
!              creation test case 1: of the initial sand dune
!*********************************************************************
if (icase==1)then
  do j=1,ny
     do i=1,nx
        h(i,j)=amp*exp(-0.01*(x(i)-x0)**2)
     enddo
  enddo
elseif (icase==2) then
  do j=1,ny
     do i=1,nx
        h(i,j)=amp*exp(-0.01*(y(j)-y0)**2)
     enddo
  enddo
    ! h(i)=(1/(x(i)*sigma*sqrt(2*pi)))*exp(-10*((log(x(i))-nu)**2)/(2*sigma**2)) !
       dissymetric gaussian
endif
!*********************************************************************
!              creation test case 2: erosion pit in LEGI or test SW
!*********************************************************************
if (icase==3)then
   do j=1,ny
      do i=1,nx
         h(i,j)=0
      enddo
   enddo
endif
!2D gaussian
if (icase==4)then
   do j=1,ny
      do i=1,nx
          h(i,j)=amp*exp(-(0.01*(x(i)-x0)**2+0.01*(y(j)-y0)**2))
      enddo
   enddo
endif
!*********************************************************************
!              creation test case sw: initial gaussian at the free surface
!*********************************************************************
if (icase_oc==0) then
    do j=1,ny
      do i=1,nx
         eta(i,j)=0
      enddo
   enddo
elseif (icase_oc==1)then
   do j=1,ny
      do i=1,nx
         eta(i,j)=amp_oc*exp(-0.01*(x(i)-x0_oc)**2)
      enddo
   enddo
elseif (icase_oc==2) then
   do i=1,nx
      do j=1,ny
         eta(i,j)=amp_oc*exp(-0.01*(y(j)-x0_oc)**2)
      enddo
   enddo
elseif (icase_oc==3) then
   do j=1,ny
      do i=1,nx
         eta(i,j)=amp_oc*exp(-(0.001*(x(i)-x0_oc)**2+0.001*(y(j)-y0_oc)**2))
      enddo
   enddo
endif

end subroutine init
```

```
 !*********************************************************************
! Program by Tim NAGEL - LEGI                                         *
!                                                                     *
! date: 21/05/2014                                                    *
!                                                                     *
! This subroutine computes the atmospheric forcing induce by the wind on the  *
! ocean free surface, including the presence of one/several offshore wind     *
! turbines.                                                           *
! Wind turbine wake is take into account using the exponential model close to.*
! the Jensen one                                                      *
!                                                                     *
!   All units sgi                                                     *
!*********************************************************************
subroutine atm_forcing(x,y,sigma_a)

  implicit none
  include 'common.inc'

  real(8),dimension(nx),intent(in) :: x
  real(8),dimension(ny),intent(in) :: y
  real(8),dimension(nx,ny),intent(out) :: sigma_a
  real(8),dimension(nx,ny) :: dua,ua
  real(8),dimension(nx) :: delta_y
  real(8) :: x1,x2,x3,nyd2,maxdua,Deff,i0,Rnb,R95,c1,x_origin
  integer :: i,j
```

```
26
27 !**********************************************************************
28 !                              wake creation
29 !**********************************************************************
30   if (iwake==0) then
31      !parameters.
32      x2=200.      !distance from the boundary to the impact point of the wake with the
           ocean surface
33      x1=((ht-0.5*dr)/kj)  !distance from the turbine pile to the boundary
34      print *,'x1',x1
35      x3=1300.
36      nyd2=150.
37      rho_a=1.2
38
39      dua=0
40      do i=x2,x3
41         delta_y(i)=int(kj*(x(i)+x1))
42      enddo
43
44      do i=x2,x3
45         do j=nyd2-delta_y(i),nyd2+delta_y(i)
46          !  dua(i,j)=(1-sqrt(1-drag))/(1+(2*kj*(x(i)+x1)/(dr))**2)
47           dua(i,j)=exp(-(x(i)-200)/300)
48         enddo
49      enddo
50   endif
51
52   maxdua=maxval(dua)
53   print *, 'maxdua', maxdua
54   ua=ua0*(1-dua)
55 !**********************************************************************
56 !                            Atmospheric shear stress
57 !**********************************************************************
58   sigma_a=rho_a*cd1*(ua**2)
59
60 end subroutine atm_forcing
```

```
1
2  !**************************************************************************
3  ! Program by Tim NAGEL - LEGI                                             *
4  !                                                                         *
5  ! date: 31/03/2014                                                        *
6  !                                                                         *
7  ! Restart subroutine (reading (1) and writing(2))                         *
8  !                                                                         *
9  !   All units sgi                                                         *
10 !                                                                         *
11 !**************************************************************************
12 SUBROUTINE restart_sub(irestart,t,eta,u,v,h,omega)
13 implicit none
14 include 'common.inc'
15 INTEGER, intent(inout) :: irestart
16 real(8), intent(inout) :: t
17 real(8), intent(inout),dimension(nx,ny) :: eta,u,v,h,omega
18 CHARACTER(2) :: filename_irestart
19 CHARACTER(75) :: filename_restart
20 WRITE(filename_irestart,FMT='(I2)')10+irestart_name
21
22 print *, 'restart ok'
23 !**********************************************************************
24 !                                reading
25 !**********************************************************************
26 if (irestart==1) then
27 filename_restart="restart_rs.data"
28 Open(unit=16,status='old',Form='Unformatted',file=filename_restart)
29  read(16) irestart_name
30  read(16) eta
31  read(16) u
32  read(16) v
33  read(16) h
34  read(16) t
35         !read(16) omega
36
37 !**********************************************************************
38 !                                writing
39 !**********************************************************************
40 elseif (irestart==2) then
41 filename_restart="restart_"//trim(filename_irestart)//".data"
42 Open(unit=16,status='new',Form='Unformatted',file=filename_restart)
43  write(16) irestart_name
44  write(16) eta
45  write(16) u
46  write(16) v
47  write(16) h
48  write(16) t
49         !write(16) omega
50 endif
51 close(16)
52 Return
53 END SUBROUTINE restart_sub
```

```
 subroutine ocean(u,v,eta,hmean,h,sigma,u_c,v_c,eta_c,s)

  !*************************************************************************
  ! Program by Tim NAGEL - LEGI                                            *
  !                                                                        *
  ! date: 26/03/2014                                                       *
  !                                                                        *
  ! This subroutine is the ocean part of the main program, 2D SW equations for  *
  ! ocean are solved.                                                      *
  !                                                                        *
  !   All units sgi                                                        *
  !*************************************************************************
  use netcdf
  implicit none
  include 'common.inc'

  real(8),dimension(nx,ny),intent(in) :: u,v,eta,hmean,h,sigma
  real(8),dimension(nx,ny),intent(out) :: u_c,v_c,eta_c,s
  real(8),dimension(nx,ny) :: cfl,u_p,eta_p,v_p,u_bc,v_bc,eta_bc
  real(8) :: maxu,maxeta,maxcfl,maxup,maxetap,maxvp,maxvc,dx1,dx2,dy1,dy2
  integer :: i,j

  u_c=u
  eta_c=eta
  v_c=v

  dx1=dx**(-1)
  dx2=dx**(-2)
  dy1=dy**(-1)
  dy2=dy**(-2)

  call boundary_conditions(hmean,u_c,v_c,eta_c)

!((0.005)/(hmean(i,j)-h(i,j)+eta_c(i,j))) &!
!************************************************************************
!            SW equation, second order RK time scheme
!       predictor step u_p=u_predictor and u_c=u_corrector
!************************************************************************
  do j=2,ny-1
     do i=2,nx-1
        u_p(i,j)=u_c(i,j)+0.5*dt*((sigma(i,j)/(rho*(hmean(i,j)-h(i,j)+eta_c(i,j)))) &!
            forcing
        -((u_c(i,j)*cd*sqrt(u_c(i,j)**2+v_c(i,j)**2))/((hmean(i,j)-h(i,j)+eta_c(i,j))))
            &!friction
        -dx1*u_c(i,j)*0.5*(u_c(i+1,j)-u_c(i-1,j)) &!advection x
        -dy1*v_c(i,j)*0.5*(u_c(i,j+1)-u_c(i,j-1)) &!advection y
        +dx2*nu*(u_c(i+1,j)-2*u_c(i,j)+u_c(i-1,j)) &!diffusion x
        +dy2*nu*(u_c(i,j+1)-2*u_c(i,j)+u_c(i,j-1)) &!diffusion y
        -dx1*g*0.5*(eta_c(i+1,j)-eta_c(i-1,j))) !pressure gradient

        v_p(i,j)=v_c(i,j)+0.5*dt*(&!sin(pi/4)*(&!(0.0000125)/(hmean(i,j)-h(i,j)+eta_c(i,
            j)))&!(sigma(i,j)/(rho*(hmean(i,j)-h(i,j)+eta_c(i,j)))) &!forcing
        -((v_c(i,j)*cd*sqrt(u_c(i,j)**2+v_c(i,j)**2))/((hmean(i,j)-h(i,j)+eta_c(i,j))))
            &!friction
        -dx1*u_c(i,j)*0.5*(v_c(i+1,j)-v_c(i-1,j)) &!advection x
        -dy1*v_c(i,j)*0.5*(v_c(i,j+1)-v_c(i,j-1)) &!advection y
        +dx2*nu*(v_c(i+1,j)-2*v_c(i,j)+v_c(i-1,j)) &!diffusion x
        +dy2*nu*(v_c(i,j+1)-2*v_c(i,j)+v_c(i,j-1)) &!diffusion y
        -dy1*g*0.5*(eta_c(i,j+1)-eta_c(i,j-1))) !pressure gradient

        eta_p(i,j)=eta_c(i,j)+0.5*dt*( &
        dx2*kappa*(eta_c(i+1,j)-2*eta_c(i,j)+eta_c(i-1,j)) & !artificial diffusion x (
            stabilisation)
        +dy2*kappa*(eta_c(i,j+1)-2*eta_c(i,j)+eta_c(i,j-1)) & !artificial diffusion y (
            stabilisation)
        -(u_c(i+1,j)*(hmean(i+1,j)-h(i+1,j)+eta_c(i+1,j)) &
        -u_c(i-1,j)*(hmean(i-1,j)-h(i-1,j)+eta_c(i-1,j)))*0.5*dx1 &
        -(v_c(i,j+1)*(hmean(i,j+1)-h(i,j+1)+eta_c(i,j+1)) &
        -v_c(i,j-1)*(hmean(i,j-1)-h(i,j-1)+eta_c(i,j-1)))*0.5*dy1)
     enddo
  enddo

  call boundary_conditions(hmean,u_p,v_p,eta_p)

  !corrector step
  do j=2,ny-1
     do i=2,nx-1
        u_c(i,j)=u_c(i,j)+dt*((sigma(i,j)/(rho*(hmean(i,j)-h(i,j)+eta_c(i,j)))) &!
            forcing
        -((u_p(i,j)*cd*sqrt(u_p(i,j)**2+v_p(i,j)**2))/((hmean(i,j)-h(i,j)+eta_p(i,j))))
            &!friction
        -dx1*u_p(i,j)*0.5*(u_p(i+1,j)-u_p(i-1,j)) &!advection x
        -dy1*v_p(i,j)*0.5*(u_p(i,j+1)-u_p(i,j-1)) &!advection y
        +dx2*nu*(u_p(i+1,j)-2*u_p(i,j)+u_p(i-1,j)) &!diffusion x
        +dy2*nu*(u_p(i,j+1)-2*u_p(i,j)+u_p(i,j-1)) &!diffusion y
        -dx1*g*0.5*(eta_p(i+1,j)-eta_p(i-1,j))) !pressure gradient

        v_c(i,j)=v_c(i,j)+dt*(&!sin(pi/4)*(&!(0.0000125)/(hmean(i,j)-h(i,j)+eta_c(i,j)))
             &!(sigma(i,j)/(rho*(hmean(i,j)-h(i,j)+eta_c(i,j)))) &!forcing
```

```
81          -((v_p(i,j)*cd*sqrt(u_p(i,j)**2+v_p(i,j)**2))/((hmean(i,j)-h(i,j)+eta_p(i,j)))) &!friction
82          -dx1*u_p(i,j)*0.5*(v_p(i+1,j)-v_p(i-1,j)) &!advection x
83          -dy1*v_p(i,j)*0.5*(v_p(i,j+1)-v_p(i,j-1)) &!advection y
84          +dx2*nu*(v_p(i+1,j)-2*v_p(i,j)+v_p(i-1,j)) &!diffusion x
85          +dy2*nu*(v_p(i,j+1)-2*v_p(i,j)+v_p(i,j-1)) &!diffusion y
86          -dy1*g*0.5*(eta_p(i,j+1)-eta_p(i,j-1))) !pressure gradient
87
88          eta_c(i,j)=eta_c(i,j)+dt*( &
89          kappa*dx2*(eta_p(i+1,j)-2*eta_p(i,j)+eta_p(i-1,j)) & !artificial diffusion x ( stabilisation)
90          +dy2*kappa*(eta_p(i,j+1)-2*eta_p(i,j)+eta_p(i,j-1)) & !artificial diffusion y ( stabilisation)
91          -(u_p(i+1,j)*(hmean(i+1,j)-h(i+1,j)+eta_p(i+1,j)) &
92          -u_p(i-1,j)*(hmean(i-1,j)-h(i-1,j)+eta_p(i-1,j)))*0.5*dx1 &
93          -(v_p(i,j+1)*(hmean(i,j+1)-h(i,j+1)+eta_p(i,j+1)) &
94          -v_p(i,j-1)*(hmean(i,j-1)-h(i,j-1)+eta_p(i,j-1)))*0.5*dy1)
95        enddo
96    enddo
97
98    call boundary_conditions(hmean,u_c,v_c,eta_c)
99
100 !**********************************************************************
101 !                        updating eta and u
102 !**********************************************************************
103   s=hmean+eta_c
104 !**********************************************************************
105 !                          CFL calculation
106 !**********************************************************************
107   cfl=sqrt(g*s)*(dt/dx)+sqrt(g*s)*(dt/dy)
108   maxcfl=maxval(cfl)
109   if (maxcfl.gt.1) then
110       print *,'CFL>1 ==> CFL:',maxcfl
111       stop
112   endif
113
114 end subroutine ocean
```

```
1   subroutine boundary_conditions(hmean,u_bc,v_bc,eta_bc)
2
3     !*****************************************************************************
4     ! Program by Tim NAGEL - LEGI                                                *
5     !                                                                            *
6     ! date: 22/04/2014                                                           *
7     !                                                                            *
8     ! This subroutine establish the BC for the SW model.                         *
9     !                                                                            *
10    !   All units sgi                                                            *
11    !*****************************************************************************
12    use netcdf
13    implicit none
14    include 'common.inc'
15
16    real(8),dimension(nx,ny),intent(in) :: hmean
17    real(8),dimension(nx,ny),intent(inout) :: u_bc,v_bc,eta_bc
18    integer :: i,j
19
20  !**********************************************************************
21  !                         East_West BC
22  !  Bernoulli: imposed input velocity and imposed output height
23  !**********************************************************************
24    if (ibcew==1) then
25    !East
26       do j=1,ny
27          u_bc(nx,j)=u_bc(nx-1,j)
28          v_bc(nx,j)=v_bc(nx-1,j)
29          eta_bc(nx,j)=0
30    !West
31          eta_bc(1,j)=eta_bc(2,j)
32          u_bc(1,j)=q0x/(hmean(1,j)+eta_bc(1,j))
33          v_bc(1,j)=v_bc(2,j)
34       enddo
35  !**********************************************************************
36  !                         East_West BC
37  !                           Periodic
38  !**********************************************************************
39    elseif (ibcew==0) then
40       do j=1,ny
41    !East
42          eta_bc(nx,j)=eta_bc(2,j)
43          u_bc(nx,j)=u_bc(2,j)
44          v_bc(nx,j)=v_bc(2,j)
45    !West
46          eta_bc(1,j)=eta_bc(nx-1,j)
47          u_bc(1,j)=u_bc(nx-1,j)
48          v_bc(1,j)=v_bc(nx-1,j)
49       enddo
50    endif
51  !**********************************************************************
52  !                         South_North BC
53  !  Bernoulli: imposed input velocity and imposed output height
```

```
!**********************************************************************
  if (ibcns==1) then
  !North
     do i=1,nx
        u_bc(i,ny)=u_bc(i,ny-1)
        v_bc(i,ny)=v_bc(i,ny-1)
        eta_bc(i,ny)=0
  !South.
        eta_bc(i,1)=eta_bc(i,2)
        u_bc(i,1)=u_bc(i,2)
        v_bc(i,1)=q0y/(hmean(i,1)+eta_bc(i,1))
     enddo
!**********************************************************************
!                       South_North BC
!                         Periodic
!**********************************************************************
  elseif (ibcns==0) then
     do i=1,nx
  !North
        u_bc(i,ny)=u_bc(i,2)
        v_bc(i,ny)=v_bc(i,2)
        eta_bc(i,ny)=eta_bc(i,2)
  !South
        eta_bc(i,1)=eta_bc(i,ny-1)
        u_bc(i,1)=u_bc(i,ny-1)
        v_bc(i,1)=v_bc(i,ny-1)
     enddo
  endif

end subroutine boundary_conditions
```

```
 subroutine morpho(eta,hmean,u,v,qbx1,qby1,h,h_n)

  !*****************************************************************************
  ! Program by Tim NAGEL - LEGI                                                *
  !                                                                            *
  ! date: 31/03/2014                                                           *
  !                                                                            *
  ! This subroutine tends to modelise a seabed evolution in time, by solving   *
  !  Exner equation.                                                           *
  !                                                                            *
  !   All units sgi                                                            *
  !*****************************************************************************

  use netcdf
  implicit none
  include 'common.inc'

  real(8),dimension(nx,ny),intent(in) :: u,v,h,eta,hmean,qbx1,qby1
  real(8),dimension(nx,ny),intent(out) :: h_n
  real(8),dimension(nx,ny) :: cfl,slope,qb,h1,qbx,qby,h_st,u_c,eta_c,v_c
  real(8) :: maxu,maxqb,maxcflmorpho,maxeta,maxhst
  integer :: i

if (icase==1.or.icase==2.or.icase==3.or.icase==4) then

  call sedimentary_fluxes(u,v,qbx,qby)

  if (ischeme==0) then
     call upwind(h,qb,h_n)
  endif

  if (ischeme==1) then
     call NOCS_coloc_jiang(u,v,qbx1,qby1,eta,hmean,h,qbx,qby,h_n)
  endif
endif
end subroutine morpho
```

```
 !*****************************************************************************
! Program by Tim NAGEL - LEGI                                                 *
!                                                                             *
! date: 18/02/2014                                                            *
!                                                                             *
! This subroutine calculates the sedimentary fluxes for the morphodynamics    *
! model, resolution method depends of initals conditions (see input_morpho)   *
!                                                                             *
!   All units sgi                                                             *
!*****************************************************************************
subroutine sedimentary_fluxes(u,v,qbx,qby)

implicit none
include 'common.inc'

real(8),dimension(nx,ny),intent(in) :: u,v
real(8),dimension(nx,ny),intent(out) :: qbx,qby
real(8),dimension(nx,ny) :: theta,qb_b,qb
real(8) :: m,theta_c,maxtheta,p,qsc,p1
integer :: i,j

```

```
!parameters
theta_c=0.052e0     !Critical Shields Number normal:0.052e0
p=0.5                                          !bed porosity
p1=1/(1-p)
qsc=d*sqrt(((rho_s/rho)-1)*g*d)                !sediment flux scale

!*********************************************************************
!                    Current Shields number calculation
!*********************************************************************
do j=1,ny
   do i=1,nx
      theta(i,j)=0.5*rho*cd*(u(i,j)**2+v(i,j)**2)/((rho_s-rho)*g*d)
   enddo
enddo
maxtheta=maxval(abs(theta))
print *,'maxtheta_inside',maxtheta

if (ibedflux==0) then
  do j=1,ny
     do i=1,nx
        qb(i,j)=alpha*(u(i,j))**beta
     enddo
  enddo
elseif (ibedflux==1) then
  do j=2,ny-1
     do i=2,nx-1
      !qb(i,j)=MAX(p1*qsc*alpha*(theta(i,j)-theta_c)**(beta),0.e0)
        qb_b(i,j)=MAX(p1*qsc*alpha*(theta(i,j)-theta_c)**(beta),0.e0)
        qbx(i,j)=qb_b(i,j)*(u(i,j)/sqrt(u(i,j)**2+v(i,j)**2))
        qby(i,j)=qb_b(i,j)*(v(i,j)/sqrt(u(i,j)**2+v(i,j)**2))
     enddo
  enddo
endif
!*********************************************************************
!                    Flux Boundary Conditions
!*********************************************************************
   do j=1,ny
      qbx(1,j)=0.
      qbx(nx,j)=qbx(nx-1,j)
   enddo
elseif (ibcew==0) then
   do j=1,ny
      qbx(1,j)=qbx(nx-1,j)
      qbx(nx,j)=qbx(2,j)
   enddo
endif

if (ibcns==1) then
   do i=1,nx
      qby(i,1)=0.
      qby(i,ny)=qby(i,ny-1)
   enddo
elseif (ibcns==0) then
   do i=1,nx
      qby(i,1)=qby(i,ny-1)
      qby(i,ny)=qby(i,2)
   enddo
endif

end subroutine sedimentary_fluxes
```

```
 subroutine bc_morpho(h,h_pred,h_n)

  !*************************************************************************
  ! Program by Tim NAGEL - LEGI                                            *
  !                                                                        *
  ! date: 28/04/2014                                                       *
  !                                                                        *
  ! This subroutine establish the BC for the morphodynamic model.          *
  ! BC are normally periodic but can also respect Bernoulli conditions     *
                          !                                                    *
  !   All units sgi                                                        *
  !*************************************************************************
  use netcdf
  implicit none
  include 'common.inc'

  real(8),dimension(nx,ny),intent(in) :: h
  real(8),dimension(nx,ny),intent(inout) :: h_pred,h_n
  integer :: i,j

!*********************************************************************
!                       East_West BC
!  Bernoulli: imposed input velocity and imposed output height
!*********************************************************************
  if (ibcew==1) then
     do j=1,ny
        h_n(1,j)=h(1,j)                !West
        h_pred(1,j)=h(1,j)
        h_n(nx,j)=h_n(nx-1,j)          !East
```

```
        h_pred(nx,j)=h_pred(nx-1,j)
     enddo
!**********************************************************************
!                        East_West BC
!                          Periodic
!**********************************************************************
  elseif (ibcew==0) then
     do j=1,ny
        h_n(1,j)=h_n(nx-1,j)           !West
        h_pred(1,j)=h_pred(nx-1,j)
        h_n(nx,j)=h_n(2,j)             !East
        h_pred(nx,j)=h_pred(2,j)
     enddo
  endif
!**********************************************************************
!                       South_North BC
!  Bernoulli: imposed input velocity and imposed output height
!**********************************************************************
  if (ibcns==1) then
     do i=1,nx
        h_n(i,1)=h(i,1)                !South
        h_pred(i,1)=h(i,1)
        h_n(i,ny)=h_n(i,ny-1)          !North
        h_pred(i,ny)=h_pred(i,ny-1)
     enddo
!**********************************************************************
!                       South_North BC
!                          Periodic
!**********************************************************************
  elseif (ibcns==0) then
     do i=1,nx
        h_n(i,1)=h_n(i,ny-1)           !South
        h_pred(i,1)=h_pred(i,ny-1)
        h_n(i,ny)=h_n(i,2)             !North
        h_pred(i,ny)=h_pred(i,2)
     enddo
  endif

end subroutine bc_morpho
```

```
 !*****************************************************************************
! Function by Tim NAGEL - LEGI                                                *
!                                                                             *
! date: 24/02/2014                                                            *
!                                                                             *
! This function calculates the derivative approximation of a so called phi    *
! function. This function assures a TVD (Total Variation Diminishing) in the  *
! scheme in the main program. Here the Beta-limiters are used.        *
!                                                                             *
!   All units sgi                                                             *
!*****************************************************************************

function phi(a,x1,x2,x3)
implicit none
real,intent(in):: a,x1,x2,x3  ! a=function which will be derivated
real:: phi       ! in a function its name must be declared

if ((x1.gt.0).and.(x2.gt.0).and.(x3.gt.0)) then
  phi=min(x1,x2,x3)
elseif ((x1.lt.0).and.(x2.lt.0).and.(x3.lt.0)) then
  phi=max(x1,x2,x3)
else
  phi=0
endif

end function phi


function phi_bis(x1,x2)
implicit none
real,intent(in):: x1,x2  ! a=function which will be derivated
real:: phi_bis       ! in a function its name must be declared

if ((x1.gt.0).and.(x2.gt.0)) then
  phi_bis=min(x1,x2)
elseif ((x1.lt.0).and.(x2.lt.0)) then
  phi_bis=max(x1,x2)
else
  phi_bis=0
endif

end function phi_bis
```

```
 !*****************************************************************************
! Program by Tim NAGEL - LEGI                                                 *
!                                                                             *
! date: 15/05/2014                                                            *
!                                                                             *
! This subroutine computes the bedform elevation in time for the morphodynamic*
! model using NOCS staggered scheme. NOCS scheme is described in Jiang and al.*
```

```
!*****************************************************************************
! HIGH-RESOLUTION NONOSCILLATORY CENTRAL SCHEMES WITH NONSTAGGERED GRIDS FOR  *
! HYPERBOLIC CONSERVATION LAWS (1998)           *
!                                                                             *
!   All units sgi                                                             *
!*****************************************************************************
subroutine NOCS_coloc_jiang(u,v,qbx1,qby1,eta,hmean,h,qbx,qby,h_n)

implicit none
include 'common.inc'

real(8),dimension(nx,ny),intent(in) :: qbx,qby,h,u,v,eta,hmean,qbx1,qby1
real(8),dimension(nx,ny),intent(out) :: h_n
real(8),dimension(nx,ny) :: s
real(8),dimension(0:nx,0:ny) :: h_stagg_prime_y,h_stagg,delta_hx,delta_hy,
    h_stagg_prime_x
real(8) :: maxupred,maxu,maxeta,maxhn,maxqb,maxqbpred,maxhpred

integer :: i,j
real(8) :: phi,phi_bis,x1,x2,x3
real(8), dimension(nx,ny) :: qbx_prime,qby_prime,qbx_pred,qby_pred,hx_prime,hy_prime,
    h_pred

!*****************************************************************************
!                        predictor step qbx
!*****************************************************************************

do j=2,ny-1
   do i=2,nx-1
      x1=lim*(qbx(i,j)-qbx(i-1,j))
      x2=0.5*(qbx(i+1,j)-qbx(i-1,j))
      x3=lim*(qbx(i+1,j)-qbx(i,j))
      qbx_prime(i,j)=phi(qbx(i,j),x1,x2,x3)   !derivative approximation of qbx(i)
   enddo
enddo
print *, 'morpho ok'
!*****************************************************************************
!                        predictor step qby
!*****************************************************************************
do j=2,ny-1
   do i=2,nx-1
      x1=lim*(qby(i,j)-qby(i,j-1))
      x2=0.5*(qby(i,j+1)-qby(i,j-1))
      x3=lim*(qby(i,j+1)-qby(i,j))
      qby_prime(i,j)=phi(qby(i,j),x1,x2,x3)   !derivative approximation of qby(i)
   enddo
enddo

do j=2,ny-1
   do i=2,nx-1
      h_pred(i,j)=h(i,j)-(dtmorpho*0.5/dx)*qbx_prime(i,j)-(dtmorpho*0.5/dy)*qby_prime(i,
          j)
   enddo
enddo

call bc_morpho(h,h_pred,h_n)
!*****************************************************************************
!                       compute predicted sediment fluxes
!*****************************************************************************
qbx_pred=qbx+0.5*(qbx-qbx1)
qby_pred=qby+0.5*(qby-qby1)
!*****************************************************************************
!             derivative approximation of h(i,j) in the x direction
!*****************************************************************************
do j=2,ny-1
   do i=2,nx-1
      x1=lim*(h(i,j)-h(i-1,j))
      x2=(0.5)*(h(i+1,j)-h(i-1,j))
      x3=lim*(h(i+1,j)-h(i,j))
      hx_prime(i,j)=phi(h(i,j),x1,x2,x3)
   enddo
enddo
!*****************************************************************************
!             derivative approximation of h(i,j) in the y direction
!*****************************************************************************
do j=2,ny-1
   do i=2,nx-1
      x1=lim*(h(i,j)-h(i,j-1))
      x2=(0.5)*(h(i,j+1)-h(i,j-1))
      x3=lim*(h(i,j+1)-h(i,j))
      hy_prime(i,j)=phi(h(i,j),x1,x2,x3)
   enddo
enddo
!*****************************************************************************
!                     stagerred corrector step
!*****************************************************************************
do j=1,ny-2
   do i=1,nx-2
      h_stagg(i,j)=(0.25)*(h(i,j)+h(i+1,j)+h(i,j+1)+h(i+1,j+1))&
                     +(0.0625)*(hx_prime(i,j)-hx_prime(i+1,j)+hx_prime(i,j+1)-hx_prime(i
                         +1,j+1))&
```

```
                    +(0.0625)*(hy_prime(i,j)-hy_prime(i,j+1)+hy_prime(i+1,j)-hy_prime(i
                        +1,j+1))&
            -(dtmorpho/(2*dx))*(qbx_pred(i+1,j)-qbx_pred(i,j)+qbx_pred(i+1,j+1)-
                qbx_pred(i,j+1))&
            -(dtmorpho/(2*dy))*(qby_pred(i,j+1)-qby_pred(i,j)+qby_pred(i+1,j+1)-
                qby_pred(i+1,j))
   enddo
enddo

  do j=0,ny
     h_stagg(0,j)=h_stagg(nx-2,j)          !West
     h_stagg(nx,j)=h_stagg(2,j)            !East
     h_stagg(nx-1,j)=h_stagg(1,j)
  enddo
  do i=0,nx
     h_stagg(i,0)=h_stagg(i,ny-2)          !South
     h_stagg(i,ny)=h_stagg(i,2)            !North
     h_stagg(i,ny-1)=h_stagg(i,1)
  enddo
!**********************************************************************
!                    stagerred discrete derivative
!**********************************************************************

do j=1,ny-2
   do i=1,nx-2
      delta_hx(i,j)=h_stagg(i,j)-h_stagg(i-1,j  )
      delta_hy(i,j)=h_stagg(i,j)-h_stagg(i  ,j-1)
   enddo
enddo
!BC
  do j=0,ny
    delta_hx(0,j)=delta_hx(nx-2,j)         !West
    delta_hy(0,j)=delta_hy(nx-2,j)
    delta_hx(nx,j)=delta_hx(2,j)           !East
    delta_hy(nx,j)=delta_hy(2,j)
    delta_hx(nx-1,j)=delta_hx(1,j)
    delta_hx(nx-1,j)=delta_hx(1,j)
  enddo
  do i=0,nx
    delta_hx(i,0)=delta_hx(i,ny-2)         !South
    delta_hy(i,0)=delta_hy(i,ny-2)
    delta_hx(i,ny)=delta_hx(i,2)           !North
    delta_hy(i,ny)=delta_hy(i,2)
    delta_hx(i,ny-1)=delta_hx(i,1)
    delta_hx(i,ny-1)=delta_hx(i,1)
  enddo

do j=1,ny-2
   do i=1,nx-2
      h_stagg_prime_x(i,j)=phi(h_stagg(i,j),delta_hx(i,j),0.5*(delta_hx(i,j)+delta_hx(i
          +1,j)),delta_hx(i+1,j))
      h_stagg_prime_y(i,j)=phi(h_stagg(i,j),delta_hy(i,j),0.5*(delta_hy(i,j)+delta_hy(i,
          j+1)),delta_hy(i,j+1))
   enddo
enddo
!BC
 do j=0,ny
     h_stagg_prime_x(0,j)=h_stagg_prime_x(nx-2,j)      !West
     h_stagg_prime_y(0,j)=h_stagg_prime_y(nx-2,j)
     h_stagg_prime_x(nx,j)=h_stagg_prime_x(2,j)        !East
     h_stagg_prime_y(nx,j)=h_stagg_prime_y(2,j)
     h_stagg_prime_x(nx-1,j)=h_stagg_prime_x(1,j)
     h_stagg_prime_y(nx-1,j)=h_stagg_prime_y(1,j)
  enddo
  do i=0,nx
     h_stagg_prime_x(i,0)=h_stagg_prime_x(i,ny-2)      !South
     h_stagg_prime_y(i,0)=h_stagg_prime_y(i,ny-2)
     h_stagg_prime_x(i,ny)=h_stagg_prime_x(i,2)        !North
     h_stagg_prime_y(i,ny)=h_stagg_prime_y(i,2)
     h_stagg_prime_x(i,ny-1)=h_stagg_prime_x(i,1)
     h_stagg_prime_y(i,ny-1)=h_stagg_prime_y(i,1)

  enddo
!**********************************************************************
!                    non stagerred corrector step
!**********************************************************************
do j=2,ny-1
   do i=2,nx-1
      h_n(i,j)=(0.25)*(h_stagg(i,j)+h_stagg(i-1,j)+h_stagg(i-1,j-1)+h_stagg(i,j-1))&
           +(0.0625)*(h_stagg_prime_x(i-1,j-1)-h_stagg_prime_x(i,j-1)+h_stagg_prime_x(i
               -1,j)-h_stagg_prime_x(i,j))&
    +(0.0625)*(h_stagg_prime_y(i-1,j-1)-h_stagg_prime_y(i-1,j)+h_stagg_prime_y(i,j-1)-
        h_stagg_prime_y(i,j))
   enddo
enddo

call bc_morpho(h,h_pred,h_n)

maxhn=maxval(h_n)
print *, 'maxhn', maxhn
```

```
177
178 end subroutine NOCS_coloc_jiang
```

```
1   !*****************************************************************************
2  ! Program by Tim NAGEL - LEGI                                                 *
3  !                                                                            *
4  ! date: 28/02/2014                                                           *
5  !                                                                            *
6  ! This subroutine changes the bedform elevation when the slope angle of      *
7  ! betwee, two grid points is higher than the repose angle of the sediment.   *
8  ! It simulates local avalanches by reducing this slope to the repose angle   *
9  ! Equations are inspired from Sanchez et Al 2011 paper:  A non-equilibrium   *
10 ! sediment transport model for coastal inlets and navigation channels, Journal*
11 ! of Coastal Research, pp. 39 to 48, 2011.                                    *
12 ! module devellopped for 1D cases only                                       *
13 !                                                                            *
14 !   All units sgi                                                            *
15 !*****************************************************************************
16 subroutine avalanche_manag(h,h_n,h_st)
17
18 implicit none
19 include 'common.inc'
20
21 real,dimension(nx),intent(in) :: h
22 real,dimension(nx),intent(out) :: h_n
23 real,dimension(nx,2),intent(out) :: h_st ! only in the case of the slump test
24
25 integer :: i,test,m
26 real :: R,maxslope,maxdh
27 real, dimension(:),  ALLOCATABLE :: slope,dh,h_av
28
29 ALLOCATE(slope(nx),dh(nx),h_av(nx))
30      test=1
31      m=0
32      h_av=h
33     ! do while ((test==1).and.(k.lt.100))
34      do while ((test==1))
35         m=m+1
36         test=0
37         !print *, 'avalanche'
38  R=0.5
39         do i=1,nx-1
40 !*********************************************************************
41 !                        predictor step qbx
42 !*********************************************************************
43           !compute the local bed slope
44    slope(i)=(h_av(i+1)-h_av(i))/dx
45  enddo
46         slope(nx)=(h_av(nx)-h_av(nx-1))/dx
47
48         do i=1,nx
49            dh(i)=0.
50  enddo
51
52  do i=2,nx-1
53     dh(i)=0.
54 !*********************************************************************
55 !                        bathymetry correction calculation
56 !*********************************************************************
57      if (abs(slope(i-1)).gt.slope_lim) then
58               test=1
59         dh(i)=dh(i)-R*0.5*(h_av(i)-h_av(i-1)-sign(slope_lim_post*dx,h_av(i)-h_av(i-1)))
60      endif
61
62              if (abs(slope(i)).gt.slope_lim) then
63                test=1
64         dh(i)=dh(i)+R*0.5*(h_av(i+1)-h_av(i)-sign(slope_lim_post*dx,h_av(i+1)-h_av(i)))
65      endif
66   enddo
67 !*********************************************************************
68 !                        bathymetry correction application
69 !*********************************************************************
70         do i=1,nx
71             h_n(i)=h_av(i)+dh(i)
72             h_av(i)=h_n(i)
73         enddo
74 !!$!only in the case of the slump test, in order to visualize process for intermediary
       times steps
75 !!$         if (k == 10) then
76 !!$            do i=1,n
77 !!$               h_st(i,1)=h_n(i)
78 !!$            enddo
79 !!$         endif
80 !!$
81 !!$         if (k == 40) then
82 !!$            do i=1,n
83 !!$               h_st(i,2)=h_n(i)
84 !!$            enddo
85 !!$         endif
86       enddo
87 end subroutine avalanche_manag
```

```
 subroutine netcdf_creation_ocean(outputfile,x,y,h,s,u,v,t,bpot,W,Wu,omega,eta)

  use netcdf
  implicit none
  include 'common.inc'

  real(8),dimension(nx,ny),intent(in) :: h,s,u,v,bpot,W,Wu,omega,eta
  real(8),dimension(nx),intent(in) :: x
  real(8),dimension(ny),intent(in) :: y
  real(8),dimension(nx,ny),intent(in) :: t

!*********************************************************************
!                  name of the data file which will be create.
!*********************************************************************
  character(len=80):: outputfile
!*********************************************************************
!                writing 3D data, with lenght (x,y), and time (t)
!*********************************************************************
  integer, parameter :: NDIMS = 3
  character (len = *), parameter :: XDIM_NAME = "xdim"
  character (len = *), parameter :: YDIM_NAME = "ydim"
  character (len = *), parameter :: TDIM_NAME = "timedim"
  character (len = *), parameter :: X_NAME = "x"
  character (len = *), parameter :: y_NAME = "y"
  character (len = *), parameter :: T_NAME = "time"
  integer :: x_dimid, y_dimid, t_dimid
!*********************************************************************
! The start and count arrays will tell the netCDF library where to
!                    write our data.
!*********************************************************************
  integer :: start(NDIMS), count(NDIMS), iprint
!*********************************************************************
!       These program variables hold the lenght and height variables.
!*********************************************************************
  integer :: x_varid, y_varid
!*********************************************************************
!                        variables name
!*********************************************************************
  character (len = *), parameter :: H_NAME="h"
  character (len = *), parameter :: S_NAME="surface"
  character (len = *), parameter :: U_NAME="velocity x"
  character (len = *), parameter :: V_NAME="velocity y"
  character (len = *), parameter :: BPOT_NAME="Bernoulli Potential"
  character (len = *), parameter :: OMEGA_NAME="Vorticty"
  character (len = *), parameter :: ETA_NAME="Free surface elevation"
  integer :: dimids(NDIMS)
!*********************************************************************
!               each variable carry a "units" attribute.
!*********************************************************************
  character (len = *), parameter :: UNITS = "units"
  character (len = *), parameter :: H_UNITS = "meters"
  character (len = *), parameter :: X_UNITS = "meters"
  character (len = *), parameter :: Y_UNITS = "meters"
  character (len = *), parameter :: T_UNITS = "seconds"
  character (len = *), parameter :: S_UNITS = "meters"
  character (len = *), parameter :: U_UNITS = "meters/seconds"
  character (len = *), parameter :: V_UNITS = "meters/seconds"
  character (len = *), parameter :: BPOT_UNITS = "meters"
  character (len = *), parameter :: OMEGA_UNITS = "(second)^-1"
  character (len = *), parameter :: ETA_UNITS = "meters"
!*********************************************************************
!                          Create the file.
!*********************************************************************
  call check( nf90_create(outputfile, nf90_clobber, ncid) )
!*********************************************************************
!     Define the dimensions. The record dimension is defined to have
! unlimited length - it can grow as needed. Here it is the time dimension
!*********************************************************************.
  call check( nf90_def_dim(ncid, XDIM_NAME, NX, x_dimid) )
  call check( nf90_def_dim(ncid, YDIM_NAME, NY, y_dimid) )
  call check( nf90_def_dim(ncid, TDIM_NAME, NF90_UNLIMITED, t_dimid) )
!*********************************************************************
!                        Define the coordinate variables
!*********************************************************************
  call check( nf90_def_var(ncid, X_NAME, NF90_DOUBLE, x_dimid, x_varid) )
  call check( nf90_def_var(ncid, Y_NAME, NF90_DOUBLE, y_dimid, y_varid) )
  call check( nf90_def_var(ncid, T_NAME, NF90_DOUBLE, t_dimid, t_varid) )
!*********************************************************************
!              Assign units attributes to coordinate variables.
!*********************************************************************
  call check( nf90_put_att(ncid, x_varid, UNITS, X_UNITS) )
  call check( nf90_put_att(ncid, y_varid, UNITS, Y_UNITS) )
  call check( nf90_put_att(ncid, t_varid, UNITS, T_UNITS) )
!*********************************************************************
!   The dimids array is used to pass the dimids of the dimensions of
!  the netCDF variables. Both of the netCDF variables we are creating
!       share the same four dimensions. In Fortran, the unlimited
!           dimension must come last on the list of dimids.
!*********************************************************************
  dimids = (/ x_dimid, y_dimid, t_dimid /)
```

```
!*********************************************************************
!                 Define the netCDF variables data.
!*********************************************************************
  ! Define the netCDF variables for the pressure and temperature data.
  call check( nf90_def_var(ncid, H_NAME, NF90_DOUBLE, dimids, h_varid) )
  call check( nf90_def_var(ncid, S_NAME, NF90_DOUBLE, dimids, s_varid) )
  call check( nf90_def_var(ncid, U_NAME, NF90_DOUBLE, dimids, u_varid) )
  call check( nf90_def_var(ncid, V_NAME, NF90_DOUBLE, dimids, v_varid) )
  call check( nf90_def_var(ncid, BPOT_NAME, NF90_DOUBLE, dimids, bpot_varid) )
  call check( nf90_def_var(ncid, OMEGA_NAME, NF90_DOUBLE, dimids, omega_varid) )
  call check( nf90_def_var(ncid, ETA_NAME, NF90_DOUBLE, dimids, eta_varid) )
!*********************************************************************
!            Assign units attributes to the netCDF variables
!*********************************************************************
  call check( nf90_put_att(ncid, h_varid, UNITS, H_UNITS) )
  call check( nf90_put_att(ncid, s_varid, UNITS, S_UNITS) )
  call check( nf90_put_att(ncid, u_varid, UNITS, U_UNITS) )
  call check( nf90_put_att(ncid, v_varid, UNITS, V_UNITS) )
  call check( nf90_put_att(ncid, bpot_varid, UNITS, BPOT_UNITS) )
  call check( nf90_put_att(ncid, omega_varid, UNITS, OMEGA_UNITS) )
  call check( nf90_put_att(ncid, eta_varid, UNITS, ETA_UNITS) )
!*********************************************************************
!                          End define mode
!*********************************************************************
  call check( nf90_enddef(ncid) )
!*********************************************************************
!       Write the coordinate variable data. This will put the x
!            and y of our data grid into the netCDF file.
!*********************************************************************
  call check( nf90_put_var(ncid, x_varid, x) )
  call check( nf90_put_var(ncid, y_varid, y) )
end subroutine netcdf_creation_ocean



subroutine netcdf_save_ocean(t,h,s,u,v,iprint,bpot,W,Wu,omega,eta)

  use netcdf
  implicit none
  include 'common.inc'

  integer,intent(in) :: iprint
  real(8),dimension(nx,ny),intent(in) :: h,s,u,v,bpot,W,Wu,omega,eta
  real(8),dimension(nx,ny),intent(in) :: t
  integer, parameter :: NDIMS = 3
  integer :: start(NDIMS), count(NDIMS)

  call check( nf90_put_var(ncid, t_varid, t, start = (/ iprint /), count = (/ 1 /)))

  count = (/  NX, NY, 1 /)
  start = (/ 1, 1, iprint /)

  call check( nf90_put_var(ncid, h_varid, h, start = start, count = count) )
  call check( nf90_put_var(ncid, s_varid, s, start = start, count = count) )
  call check( nf90_put_var(ncid, u_varid, u, start = start, count = count) )
  call check( nf90_put_var(ncid, v_varid, v, start = start, count = count) )
  call check( nf90_put_var(ncid, bpot_varid, bpot, start = start, count = count) )
  call check( nf90_put_var(ncid, omega_varid, omega, start = start, count = count) )
  call check( nf90_put_var(ncid, eta_varid, eta, start = start, count = count) )

end subroutine netcdf_save_ocean


subroutine netcdf_close_ocean
  use netcdf
  implicit none
  include 'common.inc'
!*********************************************************************
!      Close the file. This causes netCDF to flush all buffers and make
!              sure your data are really written to disk.
!*********************************************************************
  call check( nf90_close(ncid) )
end subroutine netcdf_close_ocean

subroutine check(status)
  use netcdf
  implicit none
  integer, intent ( in) :: status

  if(status /= nf90_noerr) then
     print *, trim(nf90_strerror(status))
     stop "Stopped_ocean"
  end if
end subroutine check
```