STRATOS
A Code for 3D Free Surface Flows with Floating Constraints

B. Bianchi, A. DeSimone*, L. Heltai

Abstract

This report presents a brief discussion of the theoretical aspects and practical implementation of STRATOS . STRATOS is a 3D code for the simulation of hydrodynamic flows for incompressible fluids, in the presence of a *free surface*, capable of simulating the interaction between the free surface and a *floating object* via Lagrange multipliers.

In section 1, we review the governing equations for free surface flows, the associated boundary and initial conditions, and we introduce the floating constraint.

The numerical approximation is described in section 2, where we explore the temporal discretization, the physical domain discretization, and the finite element approximation that are implemented in STRATOS .

In section 3 we analyze in detail the structure and the main subroutines of STRATOS . The explanation of how to modify STRATOS subroutines depending on the problem to solve is postponed to section 4, where some numerical tests of increasing complexity are explained in detail.

*Corresponding Author. Email: Antonio DeSimone <desimone@sissa.it>; Tel.: +39 040 3787445; Fax: +39 040 3787528

1 Mathematical Model

Incompressible flows are typically described by the Navier-Stokes equations. This is generally true also for free surface problems, but the presence of a domain that changes with time needs to be addressed very carefully, especially from the numerical point of view.

The approximation we are going to use was originally designed to simulate tidal behavior in the Venice lagoon. As a consequence, `STRATOS` was written with some assumptions in mind that might be limiting for general free surface problems:

- the free surface is represented as the graph of a *single valued* height function (no breaking waves can be simulated);
- the extension of the domain on the xy plane is predominant with respect to the height of the free surface, and shallow water approximations are used to simplify the Navier-Stokes equations.

We assume that the flow occupies at any instant t the domain

$$\Omega(t) = \{(x, y, z) \text{ s.t. } (x, y) \in \omega \subseteq \Re^2, z \in [-h(x, y), \eta(x, y, t)]\}, \qquad (1)$$

where ω is the projection on the plane xy of $\Omega(t)$, as in figure 1.

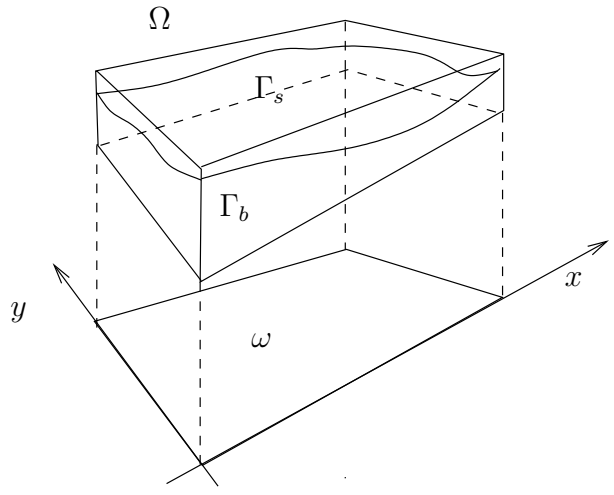

Figure 1: The three dimensional computational domain.

The free surface is identified by

$$\Gamma_s(t) = \{(x, y, \eta(x, y, t))|(x, y) \in \omega\}, \tag{2}$$

while the bottom of the domain is defined as

$$\Gamma_b = \{(x, y, -h(x, y))|(x, y) \in \omega\}. \tag{3}$$

The remaining parts of the boundary are part of the lateral surface, given by the set

$$\Gamma_l(t) = \{(x, y, z)|(x, y) \in \gamma := \partial\omega, z \in [-h(x, y), \eta(x, y, t)]\}, \tag{4}$$

which is the extrusion of the two dimensional boundary $\gamma = \partial\omega$.

Navier-Stokes equations are used to describe the hydrodynamic flow in $\Omega(t)$:

$$\frac{D\mathbf{v}}{Dt} - \text{div}(\nu \text{grad}\mathbf{v}) + \text{grad}p = 0 \qquad \text{in } \Omega(t) \tag{5a}$$

$$\text{div}\mathbf{v} = 0 \qquad \text{in } \Omega(t), \tag{5b}$$

where equations (5a) and (5b) express respectively conservation of momentum and conservation of mass.

1.1 Shallow Water Assumptions

System (5) can be simplified by using shallow water approximations with free surface flows. The idea behind this is to exploit the different physical behavior in the horizontal and vertical directions due to the different scales involved.

We begin by separating the velocity $\mathbf{v}$ in its horizontal component $\bar{u} = (u_x, u_y)$ and its vertical component w. The same is done for the horizontal and vertical viscosity (ν_h and ν_v), while the pressure is split into a hydrostatic term $p_h = p_a + g(\eta - z)$ and a hydrodynamic correction q such that

$$p = g(\eta - z) + q \qquad \text{in } \Omega(t), \tag{6}$$

where we set the atmospheric pressure p_a to zero.

Notice that, strictly speaking, the variable η in (6) is *not the same as* the one that appears in (2), since the domains of the two variables are different.

We will not specify different symbols for the two variables, and it will be assumed that $\eta(x, y, z)$ in (6) is the same as $\eta(x, y)$ in (2).

Indicating with $\nabla = (\partial/\ \partial x, \partial/\ \partial y)$ the gradient operator in the (x, y) plane, and with $D/\ Dt = \partial/\ \partial t + \bar{u} \cdot \nabla + w\partial/\ \partial z$ the total time derivative, or material derivative, we can reformulate system (5) as

$$\frac{D\bar{u}}{Dt} = -g\nabla\eta - \nabla q + \nabla \cdot (\nu_h \nabla \bar{u}) + \frac{\partial}{\partial z}\Big(\nu_v \frac{\partial \bar{u}}{\partial z}\Big) \qquad \text{in } \Omega(t) \qquad (7a)$$

$$\frac{Dw}{Dt} = -\frac{\partial q}{\partial z} + \nabla \cdot (\nu \nabla w) + \frac{\partial}{\partial z}\Big(\nu_v \frac{\partial w}{\partial z}\Big) \qquad \text{in } \Omega(t) \qquad (7b)$$

$$\nabla \cdot \bar{u} + \frac{\partial w}{\partial z} = 0 \qquad \text{in } \Omega(t), \qquad (7c)$$

where equations (7a) and (7b) express respectively the horizontal and vertical conservation of momentum, while (7c) is the conservation of mass, or incompressibility constraint.

The unknowns of system (7) are $\bar{u}$, w, q and η: we need one more equation, plus initial and boundary conditions to make the system solvable.

In shallow water approximations, we can neglect the horizontal displacements of the fluid particles on the free surface $\Gamma_s(t)$. This is a reasonable approximation when the averaged horizontal component of the normal to Γ_s can be neglected.

With this assumption, the material derivative of a fluid particle on the surface $\Gamma_s(t)$ should equate the vertical velocity, i.e.,

$$\left(\frac{D\eta}{Dt} =\right) \frac{\partial \eta}{\partial t} + \bar{u} \cdot \nabla \eta = w. \qquad (8)$$

Condition (8) is also known as the *free surface kinematic condition* for *shallow water approximations*. Similarly, on the bottom surface we have

$$\left(-\frac{Dh}{Dt} =\right) - \bar{u} \cdot \nabla h = w. \qquad (9)$$

If we integrate the conservation of mass (7c) along the vertical axis between $-h$ and η and use the kinematic conditions (8) and (9), we obtain the so called *free surface equation*:

$$\frac{\partial \eta}{\partial t} + \nabla \cdot \int_{-h}^{\eta} \bar{u} dz = 0 \qquad \text{in } \omega, \qquad (10)$$

that shows how the elevation at a given point (x, y) in ω is related to the *average conservation of mass* of the column of fluid contained in $(-h, \eta)$.

Equation (10) should be added to system (7) as an additional equation for the unknown elevation η. The system of *shallow water* equations is then given by:

$$\frac{D\bar{u}}{Dt} = -g\nabla\eta - \nabla q + \nabla\cdot(\nu_h\nabla\bar{u}) + \frac{\partial}{\partial z}\big(\nu_v\frac{\partial\bar{u}}{\partial z}\big) \qquad \text{in } \Omega(t) \tag{11a}$$

$$\frac{Dw}{Dt} = -\frac{\partial q}{\partial z} + \nabla\cdot(\nu\nabla w) + \frac{\partial}{\partial z}\big(\nu_v\frac{\partial w}{\partial z}\big) \qquad \text{in } \Omega(t) \tag{11b}$$

$$\frac{\partial\eta}{\partial t} + \nabla\cdot\int_{-h}^{\eta}\bar{u}dz = 0 \qquad \text{in } \omega \tag{11c}$$

$$\nabla\cdot\bar{u} + \frac{\partial w}{\partial z} = 0. \qquad \text{in } \Omega(t) \tag{11d}$$

The discussion of initial and boundary conditions is postponed to next section.

1.2 Floating Constraints

The simplest way to consider the interaction between the free surface $\Gamma_s(t)$ and a floating object, is to impose a *non compenetration* condition on the variable η, via Lagrange multipliers.

We start by assuming that the position of the floating object is a datum of the problem, specified by the variable $\psi(x, y, t)$, and that for the object to *float* on the free surface, it is necessary that

$$\eta(x, y, t) - \psi(x, y, t) \leq 0 \qquad \forall (x, y) \in \omega, \tag{12}$$

hold at each time t.

Equation (12), also known as *floating constraint*, makes sense under the assumption that the geometry of the hull (the part of the body immersed in water) is simple enough so that the interface surface can be described as the graph of a *single valued* function.

From a physical point of view, the body in the water behaves as an external pressure field on the surface. The constraint can therefore be treated by introducing a dynamic condition on the interface region:

$$p = p_a + \lambda, \qquad \text{on } \Gamma_s(t), \tag{13}$$

where λ is the normal force field applied by the hull to the fluid.

The pressure inside the fluid can then be rewritten as:

$$p(\bar{x}, z, t) = \lambda(\bar{x}, t) + g(\eta(\bar{x}, t) - z) + q(\bar{x}, z, t). \tag{14}$$

In the context of floating boats, or devices that are moving throughout the simulation, it might be convenient to change the frame of reference in order to avoid unnecessary numerical burden. In `STRATOS` this is achieved by introducing a reference frame attached to the immersed device.

If we rename the previous coordinate system as $(\bar{x}, \bar{y}, \bar{z})$, we have

$$\begin{aligned} x &= -\bar{x} + \bar{X}(t) \\ y &= -\bar{y} \\ z &= \bar{z}, \end{aligned} \tag{15}$$

where $\bar{X}(t)$ is the trajectory of the center of mass of the boat in the coordinate system $(\bar{x}, \bar{y}, \bar{z})$. Similarly the velocity in the new coordinates will be expressed as:

$$\begin{aligned} \bar{u}(x, y, z, t) &= -\bar{u}(\bar{x}, \bar{y}, \bar{z}, t) + \dot{\bar{X}}(t) e_x \\ w(x, y, z, t) &= \bar{w}(\bar{x}, \bar{y}, \bar{z}, t). \end{aligned} \tag{16}$$

The following functions are invariant under the change of variables (15):

$$\begin{aligned} \eta(x, y, z, t) &= \bar{\eta}(\bar{x}, \bar{y}, \bar{z}, t) \\ \lambda(x, y, z, t) &= \bar{\lambda}(\bar{x}, \bar{y}, \bar{z}, t) \\ q(x, y, z, t) &= \bar{q}(\bar{x}, \bar{y}, \bar{z}, t). \end{aligned} \tag{17}$$

The final *shallow water* equations with *floating constraints* and frame of reference fixed on the floating object which are solved in `STRATOS` are given by:

$$\frac{D\bar{u}}{Dt} - \nu_v \frac{\partial}{\partial z}\Big(\frac{\partial \bar{u}}{\partial z}\Big) + g\nabla\eta + \nabla\lambda + \nabla q - \ddot{\bar{X}} e_x = 0, \quad \text{in } \Omega(t) \tag{18a}$$

$$\frac{Dw}{Dt} - \nu_v \frac{\partial}{\partial z}\Big(\frac{\partial w}{\partial z}\Big) + \frac{\partial q}{\partial z} = 0 \quad \text{in } \Omega(t) \tag{18b}$$

$$\nabla \cdot \bar{u} + \frac{\partial w}{\partial z} = 0, \quad \text{in } \Omega(t) \tag{18c}$$

$$\frac{\partial \eta}{\partial t} + \nabla \cdot \int_{-h}^{\eta} \bar{u} dz = 0 \quad \text{in } \omega \tag{18d}$$

$$\lambda(\eta - \psi) = 0, \qquad \lambda \geq 0 \qquad \eta \leq \psi \qquad \text{in } \omega, \tag{18e}$$

with initial conditions

$$\bar{u}(x,y,z,0) = \bar{u}_0(x,y,z) \qquad \forall (x,y,z) \in \Omega(0) \tag{19a}$$
$$w(x,y,z,0) = w_0(x,y,z) \qquad \forall (x,y,z) \in \Omega(0) \tag{19b}$$
$$q(x,y,z,0) = q_0(x,y,z) \qquad \forall (x,y,z) \in \Omega(0) \tag{19c}$$
$$\eta(x,y,0) = \eta_0(x,y) \qquad \forall (x,y) \in \omega. \tag{19d}$$

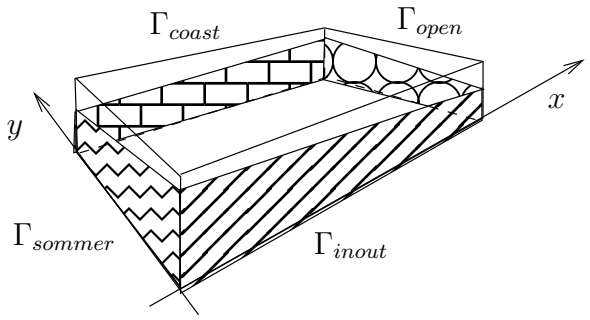

Figure 2: The boundaries of the three dimensional computational domain.

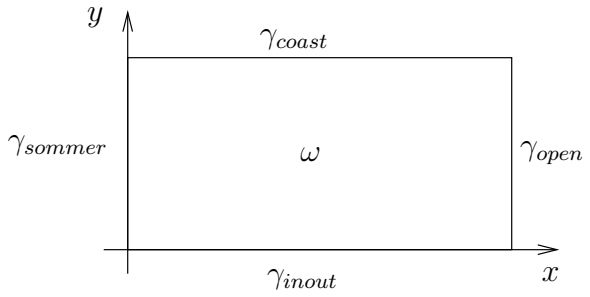

Figure 3: The boundaries of the two dimensional computational domain.

The system of equations in the new frame of reference does not vary except for the horizontal momentum balance. In that equation we have the contribution of the apparent force in a non inertial system which is due to the longitudinal acceleration of the boat. In system (18) we neglected the horizontal viscosity and assumed the vertical viscosity to be constant.

1.2.1 Boundary Conditions

On the free surface $\Gamma_s(t)$ and on the bottom surface Γ_b, standard Neumann boundary conditions are applied, which imply the continuity of the stress across the two surfaces, through dynamic equilibrium conditions. In particular we have

$$\begin{aligned} \nu_v \frac{\partial \bar{u}}{\partial z} &= f_a(W) \qquad && \text{on } \Gamma_s(t) \\ q &= p_a = 0 && \text{on } \Gamma_s(t), \end{aligned} \tag{20}$$

where $f_a(W)$ is the force exerted on the surface by the wind, and p_a is the reference atmospheric pressure (which is set to zero).

On the bottom surface a similar term appears, which can be used to take into account the effect of friction between the fluid and the material at the bottom:

$$\nu_v \frac{\partial \bar{u}}{\partial z} = \frac{|\bar{u}|\bar{u}}{C_D^2} \qquad \text{on } \Gamma_b(t). \tag{21}$$

A more detailed discussion on these issues can be found in section 1.1.2 of [14]. Here we only remark how the above boundary conditions are valid boundary conditions *only* under the *shallow water* assumption, i.e., if we can safely assume that the normals to Γ_s and Γ_b are sufficiently close to $(0, 0, 1)$ and $(0, 0, -1)$.

On the lateral sides of the domain, there are different possible choices of boundary conditions. In `STRATOS` it is possible to select among the following possibilities:

- water-coast boundary, denoted by Γ_{coast}. This is a *no-flux* boundary condition on the velocity $(\bar{u}, w)$, where impenetrability of the coast is assumed;

- in-flow or out-flow boundary, denoted by Γ_{inout}. This is a *Dirichlet* boundary condition on the velocity field $(\bar{u}, w)$, to simulate for example the inflow or outflow parts of a channel;

- fictitious *active* water-water boundary (e.g., the open sea), denoted by $\Gamma_{open}(t)$. This is a *Dirichlet* boundary condition on the variable η, where the elevation is a known function of space and time, to simulate for example incoming waves;

- fictitious *passive* water-water boundary, denoted by Γ_{sommer}. This is a *non-reflective* or *radiative* boundary condition on the variable η, to simulate the truncation of an infinite domain. It is designed to minimize the reflection of the incident waves and it is based on the *Sommerfeld* boundary condition for electromagnetic problems.

These boundary conditions are expressed as:

$$\bar{u} \cdot n = 0 \qquad \text{on } \Gamma_{coast} \tag{22}$$

$$(\bar{u}, w) = (\bar{u}_d, w_d) \qquad \text{on } \Gamma_{inout} \tag{23}$$

$$\eta = \eta_d \qquad \text{on } \gamma_{open} \tag{24}$$

$$\frac{\partial \eta}{\partial t} + \left(\dot{\bar{X}} e_x + \sqrt{g(\eta + h)n}\right) \cdot \nabla \eta = 0 \qquad \text{on } \gamma_{sommer}, \tag{25}$$

where boundary conditions (22) and (23) are on the lateral surface of the *three dimensional* domain Ω, while boundary conditions (24) and (25) are *only* applied on the *two dimensional* domain ω.

One important thing to notice is that the current implementation of the program does *not* allow one to mix the two Dirichlet boundary conditions (23) and (24) on the *same* portion of the boundary, even though this should *in theory* be possible.

Naturally conditions (22) and (25) are incompatible with each other, since one expresses the presence of a *reflecting* wall, while the other expresses the dual situation of a *non-reflective* or *radiative* wall.

2 Numerical Approximation

We discuss now a strategy to solve the problem numerically.

2.1 Time discretization

Let us consider a discretization of the time interval $[0, T]$ into N subintervals $[t^n, t^{n+1}]$ with $\Delta t = t^{n+1} - t^n$ for $n = 0, 1, ..., N-1$. We consider the following semi-implicit time advancing scheme. For each for each $n \geq 0$ we want to try $(u^{n+1}, w^{n+1}, q^{n+1}, \eta^{n+1}, \lambda^{n+1})$ such that

$$\frac{\bar{u}^{n+1}-\bar{u}^n(X)}{\Delta t}-\nu_v\frac{\partial}{\partial z}\Big(\frac{\partial \bar{u}^{n+1}}{\partial z}\Big)+g\nabla\eta^{n+1}+\nabla q^{n+1}+\nabla\lambda^{n+1}=0,\ \text{in}\ \Omega^{n+1} \tag{26}$$

$$\frac{w^{n+1}-w^n(X)}{\Delta t}-\nu_v\frac{\partial}{\partial z}\Big(\frac{\partial w^{n+1}}{\partial z}\Big)+\frac{\partial q^{n+1}}{\partial z}=0, \tag{27}$$

$$\nabla\cdot\bar{u}^{n+1}+\frac{\partial w^{n+1}}{\partial z}=0, \tag{28}$$

$$\frac{\eta^{n+1}-\eta^n(X)}{\Delta t}+\int_{-h}^{\eta^n}\nabla\cdot\bar{u}^{n+1}dz=0, \qquad \text{in}\ \omega \tag{29}$$

$$\lambda^{n+1}(\eta^{n+1}-\psi^{n+1})=0,\quad \lambda\geq 0,\quad \eta^{n+1}-\psi^{n+1}\leq 0, \tag{30}$$

where the quantities at time t^n are assumed to be known, and $X(\tilde{x},t;\tau)$ is the solution of the following problem:

$$\begin{cases}\dfrac{dX(\tilde{x},t;\tau)}{d\tau}=v(X(\tilde{x},t;\tau)) \quad \text{per}\tau\in(0,\text{t}),\\ X(\tilde{x},t;\tau)=\tilde{x},\end{cases} \tag{31}$$

The function $X(\tilde{x},t;\tau)$ is the parametric representation of the streamlines; It is the position at time τ of a particle which has been driven by the velocity field $\bar{u}$ and that occupied the position $\tilde{x}$ at time t.

We have supposed that the total derivative can be discretized as follows:

$$\frac{D\bar{u}}{Dt}\approx\frac{\bar{u}(\tilde{x},t^{n+1})-\bar{u}(X(\tilde{x},t^{n+1};t^n),t^n)}{\Delta t}=\frac{\bar{u}^{n+1}-\bar{u}^n(X)}{\Delta t}.$$

More details on this technique may be found in [8].

2.2 Space discretization

The physical 3D domain $\Omega(t)$ is embedded in a parallelepiped composed of N layers and base in ω. The thickness of layer k will be denoted by δz_k.

To discretize the 2D domain ω we will consider an unstructured triangular mesh; the mesh is placed at the mid-point of each layer. The same triangular grid is replicated in all the layers. In the following, we will denote with Ne the number of triangles and with $Nedge$ the number of edges in the xy plane mesh.

The horizontal velocity is approximated by combining the lowest order Raviart-Thomas finite elements RT_0 in the plane xy with the P_1 finite elements along the vertical direction

$$\bar{u}_h(\bar{x}, z, t) = \sum_{k=1}^{Nl} \sum_{l=1}^{Nedge} J_{k,l} \tau_l(\bar{x}) \varphi_k(z)$$

where $J_{k,l} = \int_l u_k \cdot ndl$, with u_k being the horizontal velocity on the k-th layer, $\tau_l(x, y) \in RT_0(\omega_{xy})$, and $\psi_k(z) \in P_1(-h, \eta)$. As for the vertical velocity w, we use piecewise constant finite elements in the horizontal and linear finite elements in the z direction

$$w_h(\bar{x}, z, t) = \sum_{k=1}^{Nl} \sum_{i=1}^{Noe} w_{k,i} \chi_i(\bar{x}) \varphi_k(z)$$

where $\chi_i(\bar{x}) \in P_0(\omega)$ and $\psi_k(z) \in P_1(-h, \eta)$. Finally the elevation and the hydrodynamic correction are approximated using piecewise constant functions on each triangle and on each prism. We suppose that for each triangle the elevation η is given by

$$\eta_h(\bar{x}, t) = \sum_{i=1}^{Noe} \xi_i(t) \chi_i(x)$$

where $\chi_i(x) \in P_0(\omega)$, while the hydrodynamic pressure is given by

$$q_h(\bar{x}, z, t) = \sum_{k=1}^{Nl} \sum_{i=1}^{Noe} \rho_{k,i}(t) \chi_i(\bar{x}) \phi_k(z)$$

where $\chi_i(\bar{x}) \in P_0(\omega)$ and $\phi_k(z) \in P_0(-h, \eta)$.

We need to define a finite elements basis for the Lagrange multiplier in order to introduce constraint reaction,

$$\lambda_h(\bar{x}) = \sum_{i=1}^{Noe} \mu_i(t) \chi_i(\bar{x}) \,.$$

This is a contribution to the pressure, and we use the same representation used in the code for the hydrostatic term.

2.3 Algebraic System

Combining the time discretization and the space discretization presented above, and neglecting the equation of the floating constraint, we can rewrite system (18) at each temporal step t^{n+1} as the following algebraic system:

$$\begin{vmatrix} \frac{1}{\Delta t}M_u + k_u & gD_{xy}^T & 0 & D_{xy} \\ E & \frac{1}{\Delta t}M_\eta & 0 & 0 \\ 0 & 0 & \frac{1}{\Delta t}M_w + k_w & D_z^T \\ D_{xy} & 0 & D_z & 0 \end{vmatrix} \begin{vmatrix} U^{n+1} \\ \eta^{n+1} \\ W^{n+1} \\ q^{n+1} \end{vmatrix} = \begin{vmatrix} \frac{1}{\Delta t}M_u U^n \\ \frac{1}{\Delta t}M_\eta \eta^n \\ \frac{1}{\Delta t}M_w W^n \\ 0 \end{vmatrix}. \tag{32}$$

We set

$$A = \begin{vmatrix} \frac{1}{\Delta t}M_u + k_u & gD_{xy}^T & 0 \\ E & \frac{1}{\Delta t}M_\eta & 0 \\ 0 & 0 & \frac{1}{\Delta t}M_w + k_w \end{vmatrix} \tag{33}$$

and

$$D = \begin{vmatrix} D_{xy} & 0 & D_z \end{vmatrix} \tag{34}$$

where M_u, M_η M_w are the mass matrices, k_u and k_η k_w the vertical stiffness matrices, D is the algebraic counterpart of the complete 3D divergence operator, D_{xy} acts as a divergence operator limited to the xy plane, D_z is the derivative operator in the vertical direction; finally E accounts for the divergence of the integral term in the equation.

The term $\hat{F} = (\hat{F}_1, \hat{F}_2, \hat{F}_3)^T$ accounts for boundary conditions and for the explicit terms of the time derivatives

$$F_1 = \frac{1}{\Delta t}U^n(X), \quad F_2 = \frac{1}{\Delta t}\varsigma^n \quad F_3 = \frac{1}{\Delta}W^n(X).$$

We can rewrite the system in the following way

$$\begin{vmatrix} A & D^T \\ D & 0 \end{vmatrix} \begin{vmatrix} V^{n+1} \\ Q^{n+1} \end{vmatrix} = \begin{vmatrix} \hat{F} \\ 0 \end{vmatrix} \tag{35}$$

where $V^{n+1} = (U^{n+1}, \zeta^{n+1}, W^{n+1})$ is the array of unknowns: the horizontal velocity, the elevation and the vertical velocity.

If A is invertible, we can factorize system (35) in the following way:

$$\underbrace{\begin{vmatrix} A & D^T \\ D & 0 \end{vmatrix}}_{\mathcal{A}} = \underbrace{\begin{vmatrix} A & 0 \\ D & -DA^{-1}D^T \end{vmatrix}}_{\mathcal{L}} \underbrace{\begin{vmatrix} I & A^{-1}D^T \\ 0 & -DA^{-1}D^T \end{vmatrix}}_{\mathcal{U}} \tag{36}$$

Using this factorization, solving system (35) is equivalent to solving the two block-triangular systems

$$\begin{cases} \mathcal{L}\mathcal{Y} = \mathcal{F} & \text{L} - \text{step} \\ \mathcal{U}\mathcal{X} = \mathcal{Y} & \text{U} - \text{step.} \end{cases} \tag{37}$$

We suppose that $A^{-1} \approx H_1$ in the L factor and $A^{-1} \approx H_2$ in the U factor. Then we obtain the following matrix that approximates matrix $\mathcal{A}$

$$\hat{\mathcal{L}}\hat{\mathcal{U}} = \begin{vmatrix} A & 0 \\ D & -DH_1D^T \end{vmatrix} \begin{vmatrix} I & H_2D^T \\ 0 & -DA^{-1}D^T \end{vmatrix} = \begin{vmatrix} A & AH_2D^T \\ D & D(H_2 - H_1)D^T \end{vmatrix} = \hat{\mathcal{A}} \tag{38}$$

If we define A as follows

$$A = \frac{1}{\Delta t} \underbrace{\begin{vmatrix} M_u & 0 & 0 \\ 0 & M_\eta & 0 \\ 0 & 0 & M_w \end{vmatrix}}_{\mathcal{M}} + \mathcal{K} \tag{39}$$

we can define H as

$$H = \Delta t \mathcal{M}^{-1}.$$

Now we describe two different algebraic methods to solve the factorization introduced above.

2.3.1 The generalized Chorin-Temam scheme

We choose $H = H_1 = H_2$ in the LU factorization; The approximate values of V^{n+1} and Q^{n+1} can be computed by means of the following sequence of steps:

$$L - Step = \begin{cases} A\tilde{V}^{n+1} = F, \\ D\tilde{V}^{n+1} - DHD^T\tilde{Q}^{n+1} = 0, \end{cases} \tag{40}$$

$$U - Step = \begin{cases} V^{n+1} + HD^T Q^{n+1} = \tilde{V}^{n+1}, \\ Q^{n+1} = \tilde{Q}^{n+1}. \end{cases} \tag{41}$$

So we will have a hydrostatic step where we will solve the following equations:

1. We solve the approximate elevation and the approximate horizontal velocity

$$\begin{cases} \big(\frac{1}{\Delta t} M_u + k_u\big)\tilde{U}^{n+1} + gD_{xy}^T \tilde{\eta}^{n+1} = \frac{1}{\Delta t} M_u U^n(X), \\ -E\tilde{U}^{n+1} + \frac{1}{\Delta t} M_\eta \tilde{\eta}^{n+1} = \frac{1}{\Delta t} M_\eta \eta^n \end{cases} \tag{42}$$

2. The intermediate vertical velocity

$$\big(\frac{1}{\Delta t} M_w + k_w\big)\tilde{W}^{n+1} = \frac{1}{\Delta t} M_w W^n(X), \tag{43}$$

3. The intermediate pressure

$$-\Delta t(D_{xy} M_u^{-1} D_z^T + D_z M_w^{-1} D_z^T)\tilde{Q}^{n+1} = -\big(D_{xy}\tilde{U}^{n+1} + D_z \tilde{W}^{n+1}\big). \tag{44}$$

Then a hydrodynamic correction, that will be given by:

$$\begin{cases} U^{n+1} + \Delta t M_u^{-1} D_{xy}^T Q^{n+1} = \tilde{U}^{n+1}, \\ \eta^{n+1} = \tilde{\eta}^{n+1}, \\ W^{n+1} + \Delta t M_w^{-1} D_z^T Q^{n+1} = \tilde{W}^{n+1}. \end{cases} \tag{45}$$

2.3.2 The generalized Yosida scheme

We choose $H_1 = H = \Delta t M^{-1}$ and $H_2 = A^{-1}$. Using the same approach as in the previous subsection, we obtain the same hydrostatic step, described by the equations (42), (43), (44), and a new hydrodynamic step that will be given by:

$$Q^{n+1} = \tilde{Q}^{n+1}, \tag{46}$$

$$\begin{cases} \Big(\frac{1}{\Delta t} M_w + K_w\Big) U^{n+1} + g D_{xy}^T \eta^{n+1} = \Big(\frac{1}{\Delta t} M_w + K_w\Big) \tilde{U}^{n+1} \\ \\ + g D_{xy}^T \tilde{\eta}^{n+1} - D_{xy}^T Q^{n+1}, \\ \\ -E U^{n+1} + \frac{1}{\Delta t} M_\eta \eta^{n+1} = -E \tilde{U}^{n+1} + \frac{1}{\Delta t} M_\eta \tilde{\eta}^{n+1}, \end{cases} \tag{47}$$

$$\Big(\frac{1}{\Delta t} M_w + K_w\Big) W^{n+1} = \Big(\frac{1}{\Delta t} M_w + K_w\Big) \tilde{W}^{n+1} - D_z^T Q^{n+1}. \tag{48}$$

2.4 Uzawa's method

In the following section, we will briefly describe how the values of λ^k can be determined using Uzawa's algorithm. A more detailed discussion of this implementation can be found in the appendix of [14].

As we have already observed, the pressure introduced by the constraint has only a role in the horizontal component of the momentum balance equation. In the hydrostatic step we found a solution to such equation together with the elevation equation and we could rewrite it using the single variable η.

The trajectory of the system is the one that minimize the action functional. In fact, we can show that resolving equation (42) means to resolve a problem of constrained minimization, (see chapter 3 in [14]). In Uzawa's method one replaces the constrained minimization problem with a sequence of unconstrained minimization problems.

We proceed in the following way. Let us define a sequence $(\lambda^k)_{k\geq 0}$ using a recurrence relation

$$\lambda^{k+1} = P_+(\lambda^k + \rho \nabla G(\lambda^k)), \quad k \geq 0,$$

where the parameter ρ is a positive term, as we are looking for a maximum, which should be adequately chosen in order to speed up the algorithm's convergence. In some cases we are able to calculate the gradient of function G:

$$(\nabla G(\mu))_i = \varphi(u_\mu), \quad 1 \leq i \leq m,$$

where the vectors u_μ give the solution to the unconstrained minimization problem. This is enough to formulate the iterative algorithm. Starting from an arbitrary element $\lambda_0 \in R_+^m$, we can define a sequence of pairs $(\lambda^k, u^k) \in R_+^m \times V$, $k \geq 0$ formulas of recurrence as follows:

- compute u^k:

$$J(u^k) + \sum_{i=1}^{m} \lambda_i^k \varphi_i(u^k) = \inf_{v \in V} \Big(J(v) + \sum_{i=1}^{m} \lambda_i^k \varphi_i(v) \Big),$$

- compute λ_i^{k+1}:

$$\lambda_i^{k+1} = \max\{\lambda + \rho\varphi_i(u^k), 0\} \quad 1 \leq i \leq m,$$

where $u^k = u_{\lambda^k}$. At each iteration k, the problem coincides with the unconstrained one, after the estimate of the generalized Lagrange multiplier. The convergence of this method depends on a correct estimate of the acceleration parameter ρ.

3 Structure of `STRATOS`

The code `STRATOS` is composed of several subroutines and in this section we briefly sketch the role of the most important ones. Subroutines governing the setup of the model and computations:

- `main.f90` is the main program. In this program the main file is read and the model is initialized: the input file and mesh file are read (we will look at them in more detail in section (3.4.2)); in the second phase the structures needed during the simulation are built. Finally the `sub_princip.f90` subroutine is called;

- `sub_princip.f90` is the core of the code. It performs all the computations. In section (3.1) we will analyze its structure.

The initial and boundary conditions are set in a group of subroutines whose names start with S. The initial conditions (19) are assigned in the following subroutines:

- `Sfunc_init_velxy.f90` in this subroutine the initial condition for the horizontal velocity $\bar{u}_0(x, y, z)$ is specified;

- `Sinitial_vel_w.f90` in this subroutine the initial condition for the vertical velocity $w_0(x, y, z)$ is specified;

- `Sinitial_q.f90` in this subroutine the initial condition for the hydrodynamic pressure $q_0(x, y, z)$ is specified;

- `Sinitial_eta.f90` in this subroutine the initial condition for the elevation $\eta_0(x, y)$ is specified;

The boundary conditions are set in the following subroutines:

- `Sfunc_eta.f90` in this subroutine the boundary conditions for the elevation η is specified; OUTPUT: eta_bordo. That is if we decide to assign (23) as boundary condition.

- `Scalcola_vel_bordo.f90` and `Sfunc_vel.f90`: in these subroutine the boundary conditions for the horizontal velocity $\bar{u} = (u_x, u_y)$ are specified; OUTPUT: (v_x, v_y) and OUTPUT: vel_ass. Note that if the convective term is equal to zero we must assign only vel_ass. That is if we decide to assign (24) as boundary condition.

- If we assign boundary conditions 22 or (25) , then we have nothing else to do.

3.1 Structure of `sub_princip.f90`

At the beginning all input variables are defined. Note that this subroutine doesn't give back any values. The whole problem is then solved inside the subroutine including the variable values to be plotted. So the variables for the geometry of the channel and the ship are defined. In all our tests we assigned the following values to the object:

```
geom%Lungh_CANOA = 0.3d0
geom%Largh_CANOA = 0.3d0
geom%Altz_CANOA  = 0.1d0
```

Then we call the subroutine where we initialize the unknown function, the initial solution for η, for horizontal and vertical velocity, and for the hydrodynamic pressure.

```
call init_eta
call initial_velxy OUTPUT: velocità_iniziale
call initial_vel_w
call initial_q     OUTPUT: q
```

In this case we also need the dynamics of the ship, which can be called through

```
call DinamicaImposta2 OUTPUT: CG, CGp, CGpp
```

which gives back the position, velocity, and acceleration of the ship.

Now the temporal step can begin. We fix Uzawa's parameter `tol itmax` and we initialize the parameters `lambda, rho, err_elev`. At every step, for `i = 1..passi` after initializing Uzawa `it, error`, we call the function that handles the presence of the boat

```
call constraint OUTPUT: psi, psiInt
```

Now let us resolve the hydrostatic step, which means to find a solution to system (40). The first equation we have to solve is (42), which can be rewritten in terms of the variable η alone. Then, as described in the section above, at each iteration we compute the values of λ. These values are computed each time with Uzawa's method while η is approximated as well

```
call build_coeff_ternot % build element of system
call assembla           % assemblo la matrice per eta
call reshape_solve      % resolve the system in eta
  phi= nuova_sol-psi
  lambda=lambda+rho*phi   % compute lambda
```

With the approximate values of η and λ at our disposal, we proceed by calculating velocity and pressure. First of all we compute the horizontal velocity (42)

```
call solve_vel OUTPUT: velocita
```

So if we have `hydro = 1` in the input file, we compute the vertical velocity eq. (43) and the pressure eq. (44) and we call:

```
call risolvi_w_step_1 OUTPUT: w, wbottom
call calcolo_press_nonidr OUTPUT: q
```

Now in order to compute the hydrodynamic step, we must resolve system (41), and, as we have seen in the previous sections, the values of η will be corrected depending on the chosen method (either Chorim-Teman or Yosida).

If we choose `method = 'chor'` in the input file we will resolve system (45) trough the following subroutine:

```
call calcola_velocita_step_2 OUTPUT: q, vel_step_2, w_step_2
```

If we choose `method='Yosida'` we will resolve equations (46), (47) (46)

```
call agg_cont_idros
call build_ternot_yos
call ass_yos
call reshape_solve   % OUTPUT eta
call solve_vel       % OUTPUT horizontal velocity
call risolvi_w_step_2 % OUTPUT pressure and vertical velocity
we put nuova_sol_eta = sol_eta_yos
```

Alternatively, if in the input file `hydro = 0` we call these subroutine:

```
call risolvi_wn OUTPUT: w, w_bottom
call plot_vel_surface % at every second we plot velocity
Compute the stress
Compute the dynamics
call aggiorna_geometry % update layer
% Memorizzo tutte le vecchie variabili
% Costruisco le nuove
end temporal loop
```

3.2 Structure of `constraint.f90` and `DinamicaImposta.f90`

The subroutine `constraint` of `constraint.f90` is the function that passes the analytic expression describing the shape of the boat. Actually there are three possibilities: an ellipsoid, a canal boat and a Wigley hull. Naturally, if we want no object in the fluid we put `choise` $= 0$, so that we can avoid the cycle, and the function ψ that describes the boat doesn't exit. The subroutine `DinamicaImposta2` of `DinamicaImposta.f90` is the function that describes the dynamics of the boat.

3.3 Structure of `geometry.f90`

The subroutine `aggiorna_geometry` of `geometry.f90` is a structure of dimension (`n_lati_tot x nlayer`) in which the layer thicknesses are stored. This structure is updated at each time step considering the new position of the free surface. The vector `nzdz` is an array containing the thickness of the layers for each edge where nzdz(1) is the free surface and nzdz(nzs) is the bottom; The vectors nzdz are obtained from the geometry structure. Note that `n_lati_tot` is `noe+non-1`, while `nlayer` represents the number of layers.

3.4 Description of the files required by the code

3.4.1 INPUT_FILE

The main input file which contains the principal data for the simulation must be put in the INPUT_FILE subdirectory. This file is organized as follows:

- `mesh_file`: the name of the the file containing the mesh; this file has to be placed into the subdirectory MESH
- `output path`: the directory where the output files are stored
- `convective term`: if it's equal to 0, then the convective term are not considered
- `theta`: a real number in the interval $[1/2, 1]$
- `latitude`: if it is equal to zero the Coriolis force are not taken into account

- `hydro`: a flag to toggle between the hydrostatic model (hydro=0) and non-hydrostatic (hydro=1)

- `href`: reference level

- `deltat`: the time step used in the simulation

- `stepplot`: the solution is saved every stepplot time steps

- `plot type`: it was used to choose the format of the output file; no longer used

- `solver`: it is the solver for the linear system; either UMFPACK or SPARKIT

- `method`: the method for the inexact factorization; there are two possibilities: the Chorin-Temam method 'chor' and the Yosida method 'yosi'

- `zip`: the output file can be gzipped

- `mass balance`: if this parameter is set to 1 a mass balance is performed, 0 means no mass balance required

- `vector scale`: it was used only to output a file in plotmtv; not used here

- `viscosity`: the vertical viscosity of the fluid

- `bottom friction`: 0 if we don't have bottom friction, 1 if we have bottom friction

- `Chezy`: Chezy coefficient, used if bottom friction is equal to 1

- `wind_vel_x`: the x-component of the wind velocity field

- `wind_vel_y`: the y-component of the wind velocity field

- `x_factor`: scale factor for the x coordinates of the mesh

- `y_factor`: scale factor the y coordinates of the mesh

- `nlayer`: number of layers

- `layers thicknes`

3.4.2 Mesh

STRATOS looks for mesh files in the subdirectory MESH. The original format that STRATOS supported is a modification of the TMG format (originally a binary format), to a textual input. A reference for the TMG mesh scheme can be found in http://www.dmf.unicatt.it/~paolini/tmg/.

While the TMG format would be enough for our purposes, we did not receive together with STRATOS the routines to modify TMG formats in a STRATOS compatible way. For this reason we chose to use another mesh generator altogether. Our choice was the GMSH mesh generator, mainly for the ease of use of its graphical interface, and for its elementary mesh format (see http://www.geuz.org/gmsh/ for a complete reference).

We generated the fortran subroutine gsm_read_mesh.f90 to transform the Gmsh generated mesh in a text file readable by the STRATOS subroutine tmg_read_mesh.f.

Gmsh uses a boundary representation to describe geometries. Models are created in a bottom-up flow by successively defining points, oriented lines (line segments, circles, ellipses, splines, ...), oriented surfaces (plane surfaces, ruled surfaces, triangulated surfaces, ...) and volumes.

Compound groups of geometrical entities (called either *physical entities* or *groups*) can also be defined, based on these elementary geometric entities. Gmsh's scripting language allows all geometrical entities to be fully parametrized. For our purposes, only the two dimensional capabilities of the software are needed.

Gmsh processes a scripting file (extension .geo) which describes the geometrical entities that form the domain, and produces as output a mesh file (extension .msh) that contains the definition of each vertex, triangle and boundary edge of the triangulation.

A very rough example is given below. We create a file called square.geo that defines the four vertices of a square, each of which has a *characteristic length* (called Ref in the file) attached to it:

```
Ref = 0.4;
Point (1) = {0, 0, 0, Ref};
Point (2) = {1, 0, 0, Ref};
Point (3) = {1, 1, 0, Ref};
Point (4) = {0, 1, 0, Ref};
...
```

The *characteristic length* is a quantity that describes the average size of the segments that will touch that vertex. This allows for pretty complex mesh refinements, by simply specifying different characteristic lengths for different vertices (an example usage of this capability is presented in example 4.2).

Once the vertices have been defined, we join them through straight lines:

```
Line (1) = {1, 2};
Line (2) = {2, 3};
Line (3) = {3, 4};
Line (4) = {4, 1};
...
```

and we generate a surface using a loop of these lines:

```
Line Loop (8) = {1, 2, 3, 4};
Plane Surface (8) = {8};
...
```

In order to specify different boundary conditions on different sections of the boundary, we exploit the capability of Gmsh to compound into *physical entities* any geometrical entity previously defined. This will attach a *tag* to the given element, that will be interpreted by `STRATOS` as boundary condition tags:

```
Physical Line (10001) = {1, 3};  //TAG BOUNDARY CONDITIONS
Physical Line (20001) = {2, 4};
Physical Surface (0) = {8};
```

In the example above we set the top and bottom edges of the square to have *no-flux* boundary condition tag `10001` (as in equation (22)) and the left and right edge of the square to have *inflow-outflow* boundary condition tag `20001` (as in equation (23)). Other possible tags are `30001` and `40001` respectively for *open-sea* boundary conditions (as in equation(24)) and *sommerfeld* boundary conditions (as in equation (25)).

Feeding the `square.geo` file given above to Gmsh will produce a `square.msh` file that looks like:

```
$MeshFormat
2 0 8
$EndMeshFormat
```

```
$Nodes
11
1 0 0 0
2 1 0 0
3 1 1 0
4 0 1 0
5 0.4999999999990217 0 0
6 1 0.4999999999990217 0
7 0.5000000000013878 1 0
8 0 0.5000000000013878 0
9 0.5000000000000585 0.3571428571429449 0
10 0.6500000000004281 0.7214285714285321 0
11 0.3300000000003749 0.7157142857145729 0
$EndNodes
$Elements
24
1 15 2 0 1 1
2 15 2 0 2 2
3 15 2 0 3 3
4 15 2 0 4 4
5 1 3 10001 2 0 1 5
6 1 3 10001 2 0 5 2
7 1 3 20001 3 0 2 6
8 1 3 20001 3 0 6 3
9 1 3 10001 4 0 3 7
10 1 3 10001 4 0 7 4
11 1 3 20001 5 0 4 8
12 1 3 20001 5 0 8 1
13 2 3 0 8 0 8 1 9
14 2 3 0 8 0 9 1 5
15 2 3 0 8 0 5 2 9
16 2 3 0 8 0 9 2 6
17 2 3 0 8 0 6 3 10
18 2 3 0 8 0 10 3 7
19 2 3 0 8 0 10 9 6
20 2 3 0 8 0 11 4 8
21 2 3 0 8 0 7 4 11
22 2 3 0 8 0 8 9 11
```

```
23 2 3 0 8 0 10 7 11
24 2 3 0 8 0 10 11 9
$EndElements
```

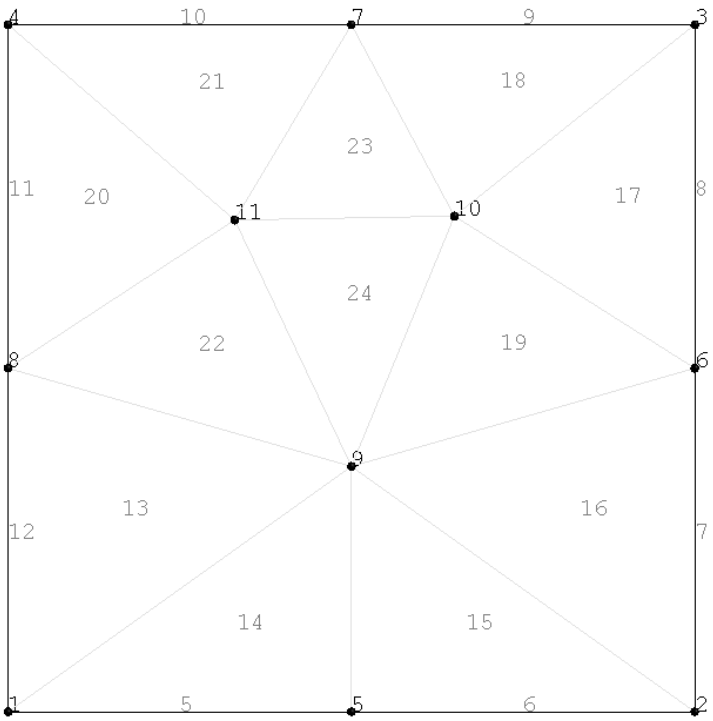

Figure 4: Mesh generated by Gmsh.

The generated mesh can be seen in figure 4. This intermediate mesh format (extension of the file: `.msh`) is not understood by `STRATOS` , and we have to filter it using the `gmsh_to_mesh.f90` subroutine.

To simplify this task, we created a script called `./make_mesh.sh` that generates both the intermediate `.msh` file in the subdirectory `GEO` and the final `.mesh` file in the subdirectory `MESH`.

This file can be executed by typing

```
./make_mesh.sh  FILENAME[.geo] [SCALE]
```

where elements between square brackets are optional, and `SCALE` is a floating number that allows to scale the average size of the triangles without modifying the `.geo` file (it defaults to 1, i.e., no scaling is performed).

The above command will look for the file `GEO/FILENAME.geo`, generate the intermediate file `GEO/FILENAME.msh`, eventually scaling down (`SCALE` $<$ 1) or up (`SCALE` $>$ 1) the size of the triangles, and process it to the final file `MESH/FILENAME.mesh`.

The `.mesh` format is the one that `STRATOS` will read. Its structure is relatively simple, and starts with a line of the kind

```
 12          11          11           3           8           2
```

where these numbers will be stored in `STRATOS` as respectively `noe` or `N_OF_ELEM`, `non` or `N_OF_NODES`, `nln` or `N_LOC_FACE`, `nbf` or `N_BOUND_FACE` and `nlf`.

The list of nodes in fortran compatible mode follows:

```
0.0000000       0.0000000   % coord - array (2 x non)
1.0000000       0.0000000   % nodes coordinates
1.0000000       1.0000000
0.0000000       1.0000000
0.5000000       0.0000000
1.0000000       0.5000000
0.5000000       1.0000000
0.0000000       0.5000000
0.5000000       0.3571429
0.6500000       0.7214286
0.3300000       0.7157143
```

and immediately after that, the list of connections that make up the triangles of the mesh:

```
 8           1           9    % lel - array (3 x noe)
 9           1           5    % node indices of each triangle
 5           2           9
 9           2           6
 6           3          10
10           3           7
10           9           6
11           4           8
 7           4          11
 8           9          11
10           7          11
10          11           9
```

Boundary nodes follow next

```
 1           5               % lgf - array (2 x nbf)
 5           2               % node indices for each
 2           6               % boundary node
 6           3
 3           7
 7           4
 4           8
 8           1
```

together with the tags specifying the boundary conditions

```
10001       10001           % lbc - array (2 x nbf)
10001       10001           % boundary conditions
20001       20001           % for each boundary node
20001       20001
10001       10001
10001       10001
20001       20001
20001       20001
```

One of the main difference with the `.msh` format is the fact that the `.mesh` format contains also *neighboring* information for each of the edges of the triangles in the mesh. When the edge is a boundary edge, then its neighbor is a negative number whose absolute value identifies the index of the boundary tags specified above:

```
2          10          -8    % adiac - array (3 x noe)
1           3          -1    % for each face, the index of the
2           4          -2    % neighbor triangle. If it is a
3           7          -3    % boundary node, it contains
6           7          -4    % -index_b, where index_b is the
5          11          -5    % index of the boundary tag above
4           5          12
9          10          -7
8          11          -6
1           8          12
6           9          12
7          10          11
```

Once a mesh has been created and stored in the subdirectory `MESH`, its name can be inserted in the dedicated option file in the subdirectory `INPUT_DATA`, and `STRATOS` will be able to use the given domain.

4 Numerical Examples

4.1 Hydrostatic, without the ship, only inflow and outflow

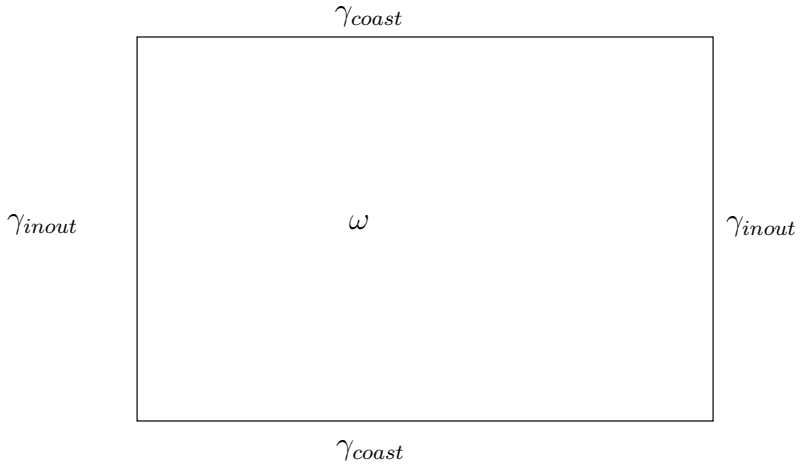

Figure 5: Domain for example 4.1.

Let us consider system (18). In the hydrostatic case, we will have $q = 0$. Without the ship there will be no floating constraint, hence $\lambda = 0$. We thus

have

$$\begin{cases} \dfrac{D\bar{u}}{Dt} - \nu_v \dfrac{\partial}{\partial z}\Big(\dfrac{\partial \bar{u}}{\partial z}\Big) + g\nabla\eta = 0, \\ \dfrac{Dw}{Dt} - \nu_v \dfrac{\partial}{\partial z}\Big(\dfrac{\partial w}{\partial z}\Big) = 0, \\ \nabla \cdot \bar{u} + \dfrac{\partial w}{\partial z} = 0, \\ \dfrac{\partial \eta}{\partial t} + \nabla \cdot \displaystyle\int_{-h}^{\eta} \bar{u} dz = 0. \end{cases} \tag{49}$$

Just to clarify, we have already neglected the horizontal viscosity and assumed the vertical one as constant. We assign the following initial conditions:

$$\begin{aligned} &\eta_0(\bar{x}) = 0 \\ &\bar{u}_0(\bar{x}, z) = w_0(\bar{x}, z) = q_0(\bar{x}, z) = 0 \end{aligned} \tag{50}$$

and the following boundary conditions:

$$\begin{aligned} &\bar{u}(x, y, z, t) \cdot n = 0, && \text{on } \Gamma_{coast}(t), \\ &\bar{u}(x, y, z, t) = \bar{u}_0(x, y, z, t), && \text{on } \gamma_{inout}. \end{aligned} \tag{51}$$

We consider a 3×2 square centered in $(-1, 0)$. The boundaries are selected as in figure 5, and the mesh built using the *descriptive geometry* file `base.geo`, saved in the `GEO` folder.

In the following examples we are going to use the same mesh, simply changing the `TAGs` for the boundary conditions, and possibly changing mesh refinement. Only this time we write in full the file `base.geo`:

```
Ref=.2;   // mesh piuttosto rada
Ax=-2;
Ay=-1;
H=2;
L=3;
Point (1) = {Ax  , Ay  , 0, Ref};
Point (2) = {Ax+L, Ay  , 0, Ref};
Point (3) = {Ax+L, Ay+H, 0, Ref};
Point (4) = {Ax  , Ay+H, 0, Ref};
```

```
Line (2) = {1, 2};
Line (3) = {2, 3};
Line (4) = {3, 4};
Line (5) = {4, 1};
Line Loop (8) = {4, 5, 2, 3};
Plane Surface (8) = {8};
Physical Point (0) = {1, 2, 3, 4};
Physical Line (10001) = {2, 4}; // TAG NO - FLUX
Physical Line (20001) = {3, 5}; // TAG INFLOW - OUTFLOW
Physical Surface (0) = {8};
```

We save `base.geo` as `inout.geo`, and we launch `./make_mesh.sh inout.geo`, which generate and save `inout.mesh` in a format which is compatible with `STRATOS` in the `MESH` folder. Now we are ready to create the input file `inout` in the folder `INPUT_FILE`:

```
'inout.mesh'   ! name of mesh file
'OUTPUT/inout' ! path - folder of output
0              ! 0 if the convective terms are not considered
1              ! theta
0              ! latitude. 0 if there isn't Coriolis's force
0              ! hydro = 0 if hydrostatic,  1 if hydrodynamics
0.3            ! href reference level
0.1            ! deltat
100            ! total number of time steps in the simulation
1              ! plotting step
plotmtv        ! plotmtv o avs
spar           ! umf o spar
chor           ! method - in alternativa "yos"
no             ! gzip
0              ! mass balance
1.0            ! vector scale
0.000001054    ! vertical viscosity of the fluid
0.0            ! 1.0: no bottom friction, 0.0: bottom friction
20             ! Chezy coefficient - but bottom friction = 0
0.0            ! wind velocity x
0.0            ! wind velocity y
1.0            ! scale factor for the x coordinates of the mesh
```

```
1.0          ! scale factor for the y coordinates of the mesh
15           ! number of layers
0.0          ! layers thickness
0.05
0.1
...
```

The numerical solution to this problem was calculated using the Chorin-Temam method, with a temporal scale 0.1 and 100 steps. Notice that the layer thickness must be smaller close to href.

The boundary and initial conditions need to be taken into account by modifying some of the subroutines of `STRATOS` . The initial conditions expressed in (50) should be set in the following files:

- The initial horizontal velocities $\bar{u}_0(\bar{x}, z)$ are assigned in the `velocita_iniziale` subroutine, in `Sfunc_init_velxy.f90`:

  ```
  u = 0.d0
  v = 0.d0
  ```

- The initial vertical velocity $w_0(\bar{x}, z)$ is assigned in the `initial_vel_w` subroutine, in `Sinitial_vel_w.f90`:

  ```
  w = 0.d0
  w_bottom = 0.d0
  ```

  Notice that in `STRATOS` the initial vertical velocity was originally set to 0 directly in the `sub_princip` routine. In the following tests we always assign initial vertical velocity equal to zero.

- The initial elevation $\eta_0(\bar{x})$ is assigned in the `init_eta` function of `Sinitial_eta.f90`

  ```
  sol_eta(elem) = 0.d0
  ```

- The initial pressure $q_0(\bar{x}, z)$ is assigned in `Sinitial_q.f90`,

```
q_0 = 0.d0
```

- The initial guess of the Lagrange multiplier λ_0 is set directly into `sub_princip.f90`.

The boundary conditions, which are selected according to the tags specified in the `.geo` file, should be modified as follows:

- the velocity $(\bar{u}, w)$ on Γ_{inout} (as in equation (23)) should be set only in the `Sfunc_vel.f90` file:

```
vel_ass = CGp(1) ! this set the inflow velocity
                 ! to the velocity of the boat
do i = 1,nzs
   if ((latnum.ge.1).and.(latnum.le.100)) then
      vel_ass(i) = velocita*lungh
   else
      vel_ass(i) = -velocita*lungh
   end if
end do
```

We then modify the file `constraint.f90`, which defines the analytic function describing the floating object's shape, in order to ignore the presence of the ship. As we do not want to consider any floating constraint, we can modify the file by selecting

```
choice = 0
```

where required. We obtain the expected numerical results, with η is constantly equal to its initial configuration, and $\bar{u}$ is constantly equal to $(-2, 0)$.

4.2 Hydrostatic, without the ship, and considering a wave

The following example explores a different boundary condition on the domain ω. In particular, we analyze both the *Dirichlet* boundary conditions

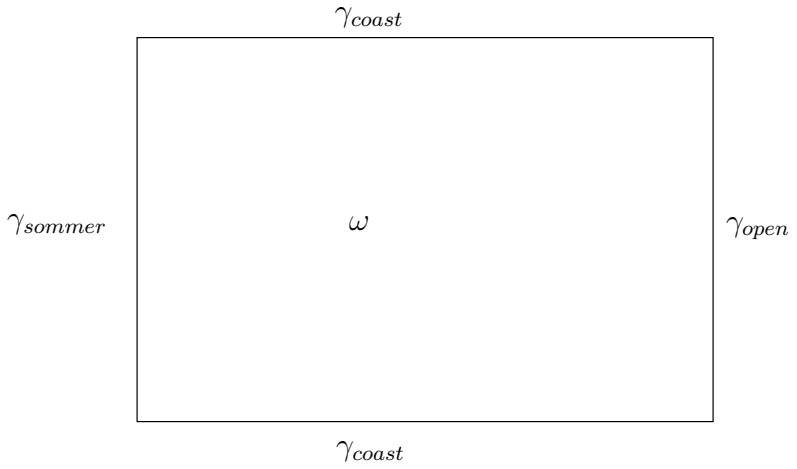

Figure 6: Domain for example 4.2.

and the *Sommerfeld* boundary conditions for η, as shown in figure 6, maintaining the same initial conditions. The goal of this example, is to verify the behavior of `STRATOS` in treating incoming waves inside a channel.

We give the following boundary conditions

$$\begin{aligned} &\bar{u}(x,y,z,t)\cdot n = 0 && \text{on } \Gamma_{coast}(t) \\ &\eta(x,y,t) = \eta_0 \cos(2\pi t) && \text{on } \gamma_{open} \\ &\frac{\partial \eta}{\partial t} + \left(\sqrt{g(\eta+h)}\right)\nabla\eta\cdot n = 0 && \text{on } \gamma_{sommer}, \end{aligned} \tag{52}$$

where $\eta_0 = 0.1$.

The domain for this example is the one represented in figure 6. In order to have a better resolution around the region where the wave is being generated, we refine the zone closer to the edge where we assign the Dirichlet boundary condition on η. This is achieved by modifying the *characteristic length* of the nodes that correspond to the given boundary, as we did in file `sommer_wave.geo`:

```
Ref=.08;
Ax=-2;
Ay=-1;
H=2;
L=3;
```

```
Point (1) = {Ax  , Ay  , 0, Ref};
Point (2) = {Ax+L, Ay  , 0, Ref/4}; // more refined on edge 3
Point (3) = {Ax+L, Ay+H, 0, Ref/4};
Point (4) = {Ax  , Ay+H, 0, Ref};
...
```

The different boundary conditions are achieved by specifying the physical identifications of the boundary segments in `sommer_wave.geo` as follows:

```
...
Physical Line (10001) = {2, 4}; // TAG  NO - FLUX
Physical Line (30001) = {3};    // TAG DIRICHLET CONDITION ON ETA
Physical Line (40001) = {5};    // TAG SOMMERFELD CONDITION
...
```

The input file `sommer_wave` in `INPUT_FILE` has been modified as follows:

```
'sommer_wave.mesh'  ! name mesh file
'OUTPUT/sommer_wave'! path - folder of output
...
0.01        ! deltat
1000        ! total number of time steps in simulation
...
```

to take into account the higher refinement of the mesh.

Once we tagged in the mesh file the boundary conditions for each of the boundary edge, we need to modify `STRATOS` subroutines that actually define those boundary conditions. Here the only file that needs to be changed is the one that determines the Dirichlet data on η, since the other boundary conditions require no input data:

- the elevation η on Γ_{open} (as in equation (24)) should be set in the file `Sfunc_eta.f90`

  ```
  freq = 2*3.14*3
  eta_bordo = 0.01*dsen(tempo*freq)
  ```

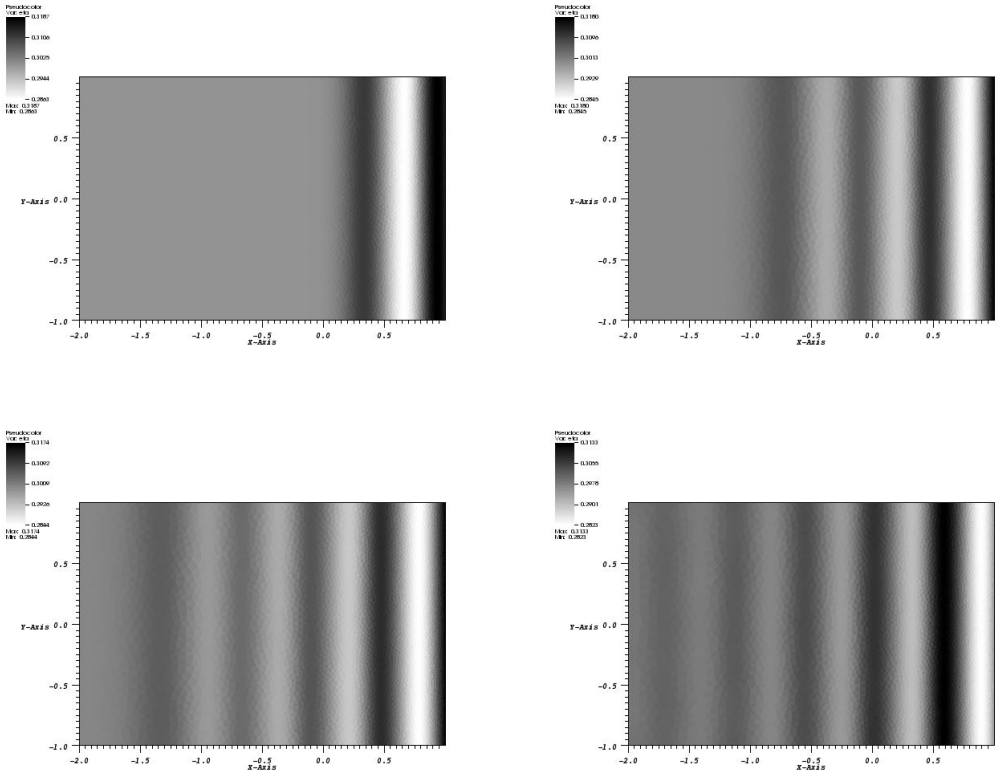

Figure 7: Result for example 4.2.

4.3 Hydrostatic, with the ship, no-flux and Sommerfeld condition

The following example explores a different boundary condition on the domain ω. In particular, we give now *no-flux* on three edges and in the remaining one we impose a *Sommerfeld* condition, as shown in figure **??**.

We also consider a floating constraint given by a ball with radius 0.02 centered in $(0, 0)$, in the domain shown in figure 8.

We assume the following boundary conditions:

$$\begin{aligned} &\bar{u}(x, y, z, t) \cdot n = 0 && \text{on } \Gamma_{coast}(t) \\ &\frac{\partial \eta}{\partial t} + \left(\dot{\bar{X}} e_x + \sqrt{g(\eta + h)n}\right) \cdot \nabla \eta = 0 && \text{on } \gamma_{sommer}. \end{aligned} \tag{53}$$

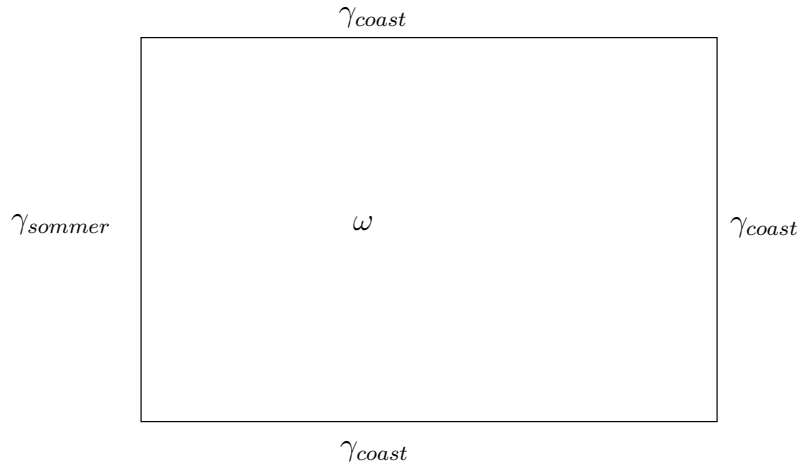

Figure 8: Domain for example 4.3.

These boundary conditions have been saved in the file `contraint.geo` file as:

```
...
Physical Line (10001) = {2, 3}; // TAG NO - FLUX
Physical Line (40001) = {3, 5}; // TAG SOMMERFELD CONDITION
...
```

We save this file as `sommer_wall_constraint.geo`, and then we launch the command `./make_mesh.sh sommer_wall_constraint.geo` 0.02, and `sommer_wall_constrain` will be saved in the `MESH` folder.

The input file `sommer_wall_constraint` of `INPUT_FILE` should be modified as follows:

```
'sommer_wall_constraint.mesh'   ! name mesh file
'OUTPUT/sommer_wall_constraint' ! path - output folder
...
```

The boundary conditions we selected do not require any modification in `STRATOS` . However we now want to introduce a floating constraint. This is done modifying the file `constraint.f90`, which defines the analytic function that describes the shape of the floating object.

There are several possible floating objects that can be selected by changing the variable `choice`. If `choice` is set to zero, then no floating constraint will be introduced. Now we select

```
choice = 2
```

which describes the following ellipsoid:

$$\left(\frac{2x}{\text{Lungh}}\right)^2 + \left(\frac{2y}{\text{Largh}}\right)^2 + \left(\frac{z}{\text{Altz}}\right)^2 = (1)^2.$$

The variables `Lungh Largh Altz` are defined in `sub_princip.f90` as:

```
...
geom%Lungh_CANOA = 0.3d0
geom%Largh_CANOA = 0.3d0
geom%Altz_CANOA  = 0.1d0
...
```

The position and the dynamics of the object are described in the `DinamicaImposta2` subroutine of `DinamicaImposta.f90`. In our example we put:

```
U = 0.0d0
...
CGp(1)  = U   ! velocità longitudinale iniziale
CG(1)   = 0   ! posizione iniziale
...
```

which correspond to an object that is standing still in its original position.

Notice that at time $t = 0$ the object is above the level of the water. It is pushed to its final position in 4 time steps, as described in the subroutine `DinamicaImposta2`.

4.4 Hydrostatic, with the ship, and considering a wave

In the following example, we have the boundary condition on the domain ω given by *No-flux* on the lateral surface, while we impose *Sommerfeld* condition and a *Dirichlet* condition on η for the other two surface, as shown in figure **??**.

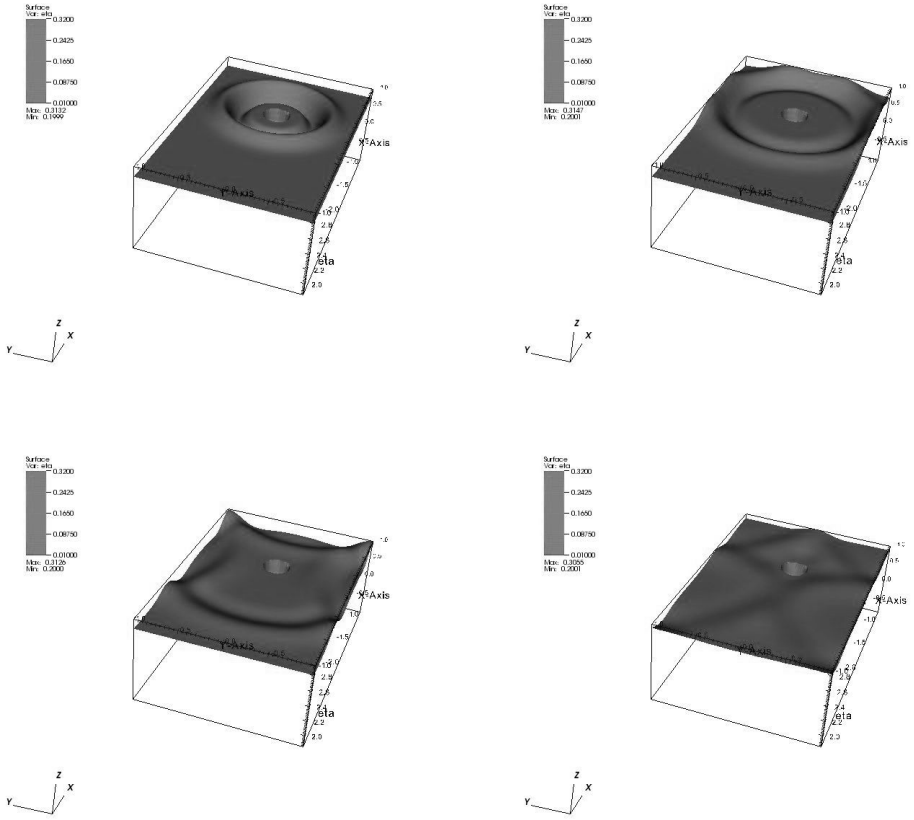

Figure 9: Result for example 4.3.

As in the case above we consider a floating constraint in the domain. The floating constraint is again given by a ball with radius 0.02 centered at $(0, 0)$. The domain ω is the same square considered above.

We assume the following boundary conditions:

$$\begin{aligned} &\bar{u}(x, y, z, t) \cdot n = 0, && \text{on } \Gamma_{coast}(t), \\ &\eta(x, y, t) = \eta_0 \cos(2\pi t), && \text{on } \gamma_{open}, \\ &\frac{\partial \eta}{\partial t} + \left(\sqrt{g(\eta + h)}\right) \nabla \eta \cdot n = 0, && \text{on } \gamma_{sommer}. \end{aligned} \tag{54}$$

These boundary conditions can be obtained by modifying the `base.geo` file as follows

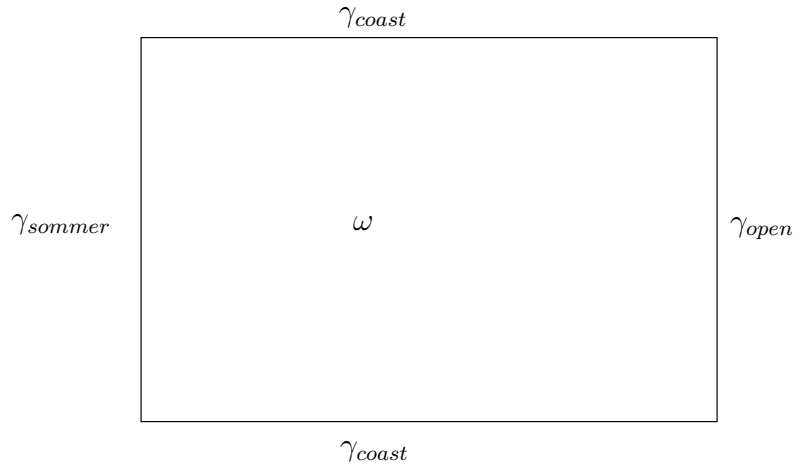

Figure 10: Domain for example 4.4

```
Ref = 1;                          // base refine
...
Physical Line (10001) = {2, 4}; // TAG NO - FLUX
Physical Line (30001) = {3}     // TAG DIRICHLET CONDITION
Physical Line (40001) = {5};    // TAG SOMMERFELD CONDITION
...
```

Once again, we save this file as `sommer_wave_constraint.geo` and we then launch `./make_mesh.sh sommer_wave_constraint.geo` 0.02. Then the file `sommer_wave_constraint.mesh` will be saved in the `MESH` folder.

The input file `sommer_wave_constraint` in `INPUT_FILE` should be modified as follows:

```
'sommer_wave_constraint.mesh'   ! name mesh file
'OUTPUT/sommer_wave_constraint' ! path - output folder
...
```

The boundary conditions are modified in the same way described above. Moreover, to describe the presence of a floating object, we choose the same ellipsoid used above. We can modify the subroutine `constraint.f90` which defines the analytic function describing the shape of the floating object and `DinamicaImposta.f90` which specifies its dynamics in the same way as in to the previous example.

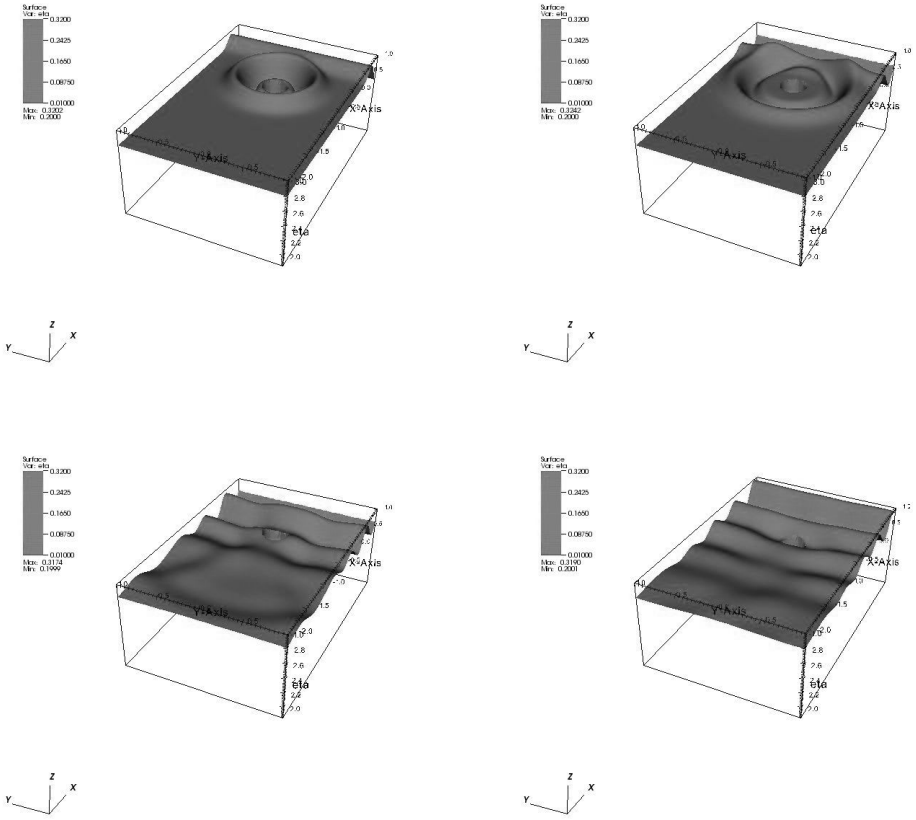

Figure 11: Result for example 4.4.

4.5 Hydrostatic, without constraint, with wave on two edges

In the following example, we have the boundary condition on the domain ω given by *No-flux* on the lateral surface, while we impose *Dirichlet* condition on η for the other two surface. We don't consider a floating constrain in the domain. The dimensions of domain ω are the same of the square considered in the previous subsections.

We assume the following boundary conditions:

$$\bar{u}(x, y, z, t) \cdot n = 0, \quad \text{on } \Gamma_{coast}(t). \tag{55}$$

$$\eta(x,y,t) = \begin{cases} \eta_0, & x < 0, \\ \eta_0/2, & x > 0, \end{cases} \quad \text{on } \gamma_{open}. \tag{56}$$

These boundary conditions can be obtained by modifying the `base.geo` file as follows:

```
...
Physical Line (10001) = {2, 4}; \\ TAG NO - FLUX
Physical Line (30001) = {3, 5}; \\ TAG DIRICHLET CONDITION
...
```

and we rename this file `solitone.geo` in the `GEO` folder. Once again, we launch `./make_mesh.sh solitone.geo` 0.05, and `solitone.mesh` will be saved automatically in the `MESH` folder.

The input file `solitone` should be modified as follows:

```
'solitone.mesh'   ! name mesh file
'OUTPUT/solitone' ! path - output folder
...
```

The initial conditions for this problem can be achieved by modifying:

- the initial horizontal velocity (u_x, u_y) in the subroutine `velocita_iniziale` of `Sinitial_velxy.f90`:

  ```
  v = 0
  u = 0
  ```

- the initial vertical velocity $w_0(x,y,z)$ is always suppose equal 0 in the `sub_princip` subroutine.

- the elevation η should be set in the `Sinitial_eta_bordo.f90` file:

  ```
  eta(elem) = 0
  ```

The boundary conditions for this problem can be achieved by modifying:

- the elevation η on Γ_{open} should be set in the `Sfunc_eta.f90` file:

```
if (lato_bordo.ge.1).and.(lato_bordo.le.123) then
 eta_bordo = 0.2d0
else
 eta_bordo = 0.4d0
end if
```

Notice that the cardinality of `lato_bordo` is 246.

- the velocity v should be set in the `Sfunc_vel.f90` file:

```
vel_ass = 0
```

4.6 Hydrostatic, with Sommerfeld conditions and we move the ship

In the following example, the boundary condition on the domain ω are always given by *No-flux* on the lateral surfaces, while we impose *Sommerfeld* condition on other two surfaces. We consider a floating object in the domain moving at a constant velocity. The object we consider is a ball with the same dimensions as in the previous cases.

We assume the following boundary conditions:

$$\bar{u}(x, y, z, t) \cdot n = 0, \quad \text{on } \Gamma_{coast}(t). \tag{57}$$

$$\frac{\partial \eta}{\partial t} + \left(\sqrt{g(\eta + h)}\right) \nabla \eta \cdot n = 0, \text{ on } \gamma_{sommer}. \tag{58}$$

These boundary conditions can be obtained by modifying the `base.geo` file as follows:

```
...
Physical Line (10001) = {2, 4}; \\ TAG NO - FLUX
Physical Line (40001) = {3, 5}; \\ TAG SOMMERFELD CONDITION
...
```

and we rename this file `ship_move.geo` in the `GEO` folder. Once again, we launch `./make_mesh.sh ship_move.geo 0.4`, and `ship_move.mesh` will be saved automatically in the `MESH` folder.

The input file `ship_move` should be modified as follows:

```
'ship_move.mesh'   ! name mesh file
'OUTPUT/ship_move' ! path - output folder
...
          ! deltat
          !numero di passi
...
```

The initial conditions for this problem can be achieved by modifying:

- the initial horizontal velocity (u_x, u_y) in the subroutine `velocita_iniziale` of `Sinitial_velxy.f90`:

  ```
  v = 0
  u = 0
  ```

- the initial vertical velocity $w_0(x, y, z)$ is always suppose equal 0 in the `sub_princip` subroutine.

- the elevation η should be set in the `Sinitial_eta_bordo.f90` file:

  ```
  eta(elem) = 0
  ```

The results are shown in figure 12.

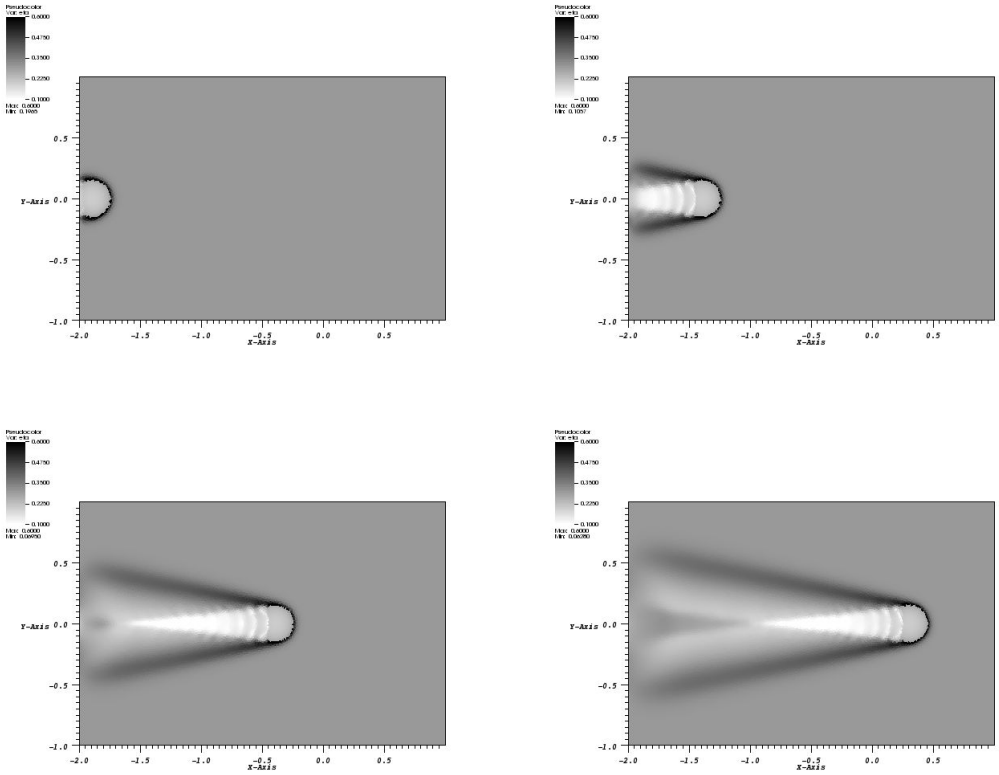

Figure 12: Result for example 4.6.

References

[1] K. Athinson, W. Han. *Theoretical Numerical Analysis - A functional analysis framework*. Springer, 2001.

[2] S. Brenner, R. Scott. *The mathematical theory of finite element method*. Springer, 2002.

[3] P. Causin, E. Miglio, F. Saleri. *Algebraic factorizations for 3D non-hydrostatic free surface flows*. Computing and Visualization in Science, Vol 5, pp. 85-94, 2002.

[4] L. Fontana, E. Miglio, A. Quarteroni, F. Saleri. *A finite element method for 3D hydrostatic water flows*. Computing and Visualization in Science, Vol 2, pp. 85-93, 1999.

[5] H. Fujita Yashima. *Note per il corso di analisi non-lineare*. Universita degli Studi di Torino, 2006-2007.

[6] M. Metcalf, J. Reid. *Fortran 90/95 explained*. Oxford University Press, 2000.

[7] E. Miglio, F. Saleri. *Lecture notes: Mathematical and numerical modelling for environmental applications*. Dipartimento di Matematica, Politecnico di Milano.

[8] E. Miglio, A. Quarteroni, F. Saleri. *Finite element approximation of Quasi-3D shallow water equations*. Computer methods in applied mechanics and engineering, Vol 174, pp. 355-369, 1999.

[9] E. Miglio. *Stratos: A finite element code for 3D free surface flows*.

[10] E. Miglio. *Mathematical and numerical modelling for environmental application*. Tesi PhD in Matematica Computazionale, 2000.

[11] A. Scotti. *Un approcccio variazionale per il vincolo di galleggiamento*. Tesi di Laurea in Ingegneria Aerospaziale, Politecnico di Milano, 2004.

[12] A. Scotti. *Approssimazione numerica di un problema di galleggiamento*. Tesi di Laurea in Ingegneria Aerospaziale, Politecnico di Milano, 2006.

[13] A. Paradiso. *Modello numerico per la dinamica di corpi galleggianti applicato alle canoe da regata.* Tesi di Laurea in Ingegneria Aerospaziale, Politecnico di Milano, 2004.

[14] A. Paradiso. *Un approccio variazionale per il vincolo di galleggiamento applicato alla dinamica di imbarcazioni.* Tesi di Laurea in Ingegneria Aerospaziale, Politecnico di Milano, 2006.