IHM—JAVA

Primitive Equations of the Ocean

Adrien Moreno - François Stoltz - MAM4

project directed by Pierre Dreyfuss

Wednesday, January 28th 2009

Acknowledgements

We would like to deeply thank the people who, during these three weeks, provided us with useful information and help assistance.

Firstly, we would like to thank the teachers Pierre DREYFUSS and Jean-François COLLET for their interest and their patience.

Secondly, we would like to thank all the people who discussed this project with us, and provided very good advice.

Abstract

This work is dedicated to the Primitive Equations *(PE)* of the Ocean, both from the theoretical and numerical viewpoints. The PE are fundamental equations of geophysical fluid dynamics, based on the hydrostatic and Boussinesq approximations.

The interest of this subject consists in the derivation of efficient domain decomposition methods for the viscous PE of the ocean. We consider the rotating 3d incompressible hydrostatic Navier-Stokes equations with free surface. Performing an asymptotic analysis of the system with respect to the Rossby number, we compute an approximated Dirichlet to Neumann operator and build a monodomain solution thanks to numerical schemes.

Key words : Primitive Equations, boundary conditions, numerical schemes.

Contents

1 Introduction

Initially, our project consists in implemanting a graphic interface to show the evolution of several parameters which represents the Ocean -particularly velocity and height of the water. An other aim more valuable, is to create an applet -a web application- to make accessible the application, and so notice the phenomenon.

However, so as to obtain graphics which represent the ocean's evolution, we must implement algorithms to find solutions of the equations. Afterward, we might see that the velocity matches a Crank-Nicholson Scheme, whereas the height matches an upwind scheme.

In a first part, from the Physical Model and hypotesis, we will intent to establish these schemes, which will correspond to the Mathematical Model. Thus, we will be able to implement it in a second part, and after this we might do a comparison between our results and the theoretical solution.

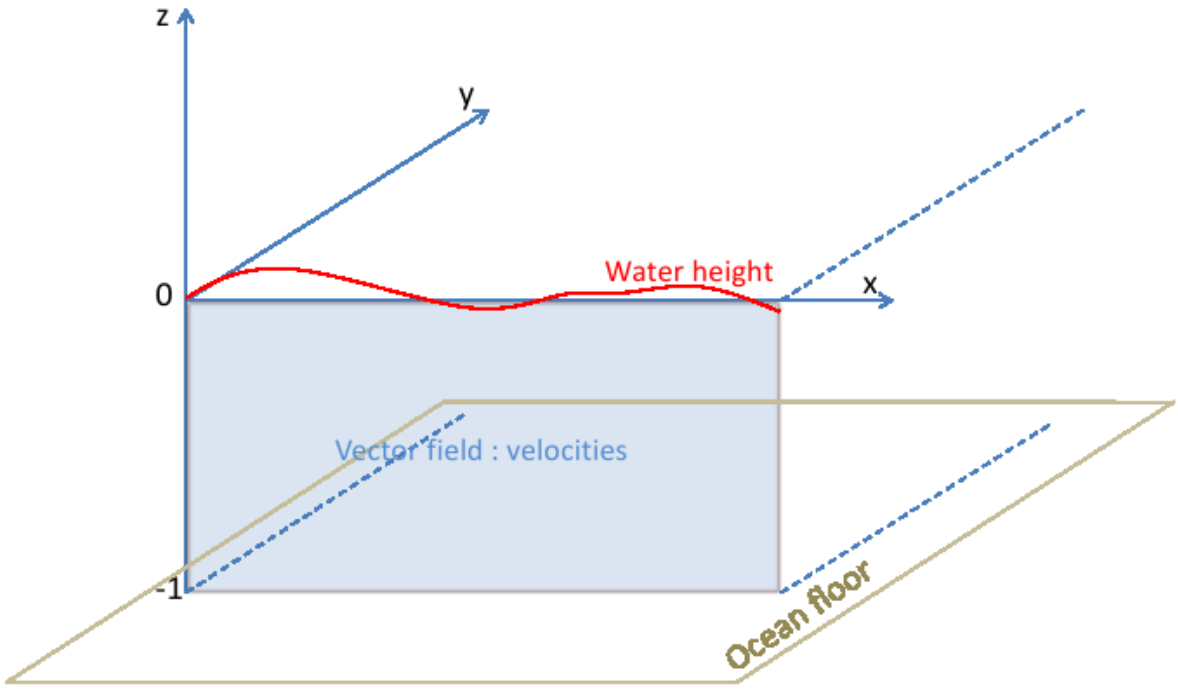


Figure 1: The studied domain

2 Theory : the Physical Model

2.1 The Equations

Before coding, it's first of all necessary for us to have a model. This modeling always comes from physical, phenomenological laws following a major observation of nature and these behaviours. Here is a question of interpreting the primitive equations of the ocean. Thus, we are going to consider the primitive equations of the ocean in the domain $(x, y, z, t) \in \mathrm{R}_x \times \mathrm{R}_y \times$[-H(x,y), $\xi(x, y, t)]_z \times \mathrm{R}_t^+$:

$$\begin{cases} \partial_t \mathbb{U}_h + \mathbb{U}_h \cdot \nabla_h \mathbb{U}_h - \nu \Delta \mathbb{U}_h + \frac{2}{\rho_0} \overrightarrow{\Omega} \wedge \mathbb{U}_h + \frac{1}{\rho_0} \nabla_h p &= 0 \\ \nabla_h \cdot \mathbb{U}_h + \partial_z w &= 0 \\ \partial_z p &= -\rho g \\ \rho &= \rho(z, T, S) \\ \partial_t T + \mathbb{U}_0 \cdot \nabla T - \nu_T \Delta T &= Q_T \\ \partial_t S + \mathbb{U}_0 \cdot \nabla S - \nu_S \Delta S &= Q_S \end{cases}$$

- **The parameters :** g (the gravity), ν (the viscosity), $\overrightarrow{\Omega}$ (the earth rotation vector), ν_T and ν_S (the diffusion coefficients).
- **The unknowns :** ($\mathbb{U}_h$,w) = (u, v, w) (the 3d-velocity), p (the pressure), ρ (the density), T (the temperature) and S (the salinity).

2.2 Hypothesis of the problem

In order to simplify the equations, and so the calculations to do in the java code, we can do some hypothesis. To begin, we can neglect the influence of the diffusion coefficients (ν_T and ν_S) and suppose that the density is constant ($\rho = \rho_0 = 1$). Moreover, we can consider that the vertical velocity w is a function of the horizontal velocity $\mathbb{U}_h$, and that the pressure p is a function of the water height ξ. To finish, we can consider that the problem is dimensionless, that is to say we can introduce the following dimensionless quantities :

$$\boxed{\begin{pmatrix} (x, y) = L(\tilde{x}, \tilde{y}) & t = (\frac{L}{\mathbb{U}})\tilde{t} \\ \xi = H\tilde{\xi} & z = H\tilde{z} \\ \mathbb{U}_h = \mathbb{U}\tilde{\mathbb{U}_h} & \mathbb{U}_0 = \mathbb{U}\tilde{\mathbb{U}_0} \end{pmatrix}}$$

2.3 Boundary and initial conditions

The solution of the problem must verify several conditions.

Initial conditions :

- Condition 1 : $\mathrm{U}_h(\cdot, 0) = \mathrm{U}_{h,i}$ in Ω,
- Condition 2 : $\xi(\cdot, 0) = \xi$ in w.

Boundary conditions :

- Condition 1 : $\frac{\partial u}{\partial n} = \frac{\partial v}{\partial n} = 0$ in $\Gamma 1$,
- Condition 2 : $\partial_z\, \mathbb{U}_h(x, y, -1, t) = \partial_z\, \mathbb{U}_h(x, y, 0, t) = 0$ in $\Gamma 2$.

Comment : the last condition represents the homogeneous Neumann boundary conditions. $\Gamma 1$ is the floor ($z = -1$) and the surface ($z = 0$) of the ocean, and $\Gamma 2$ corresponds to the sides of the studied domain.

Moreover, to simplify the problem, we can introduce some characteristic quantities, to know :

$\hookrightarrow \epsilon = \frac{\mathbb{U}}{fL}$: the Rossby number,

$\hookrightarrow Re = \frac{\mathbb{U}L}{\nu}$: the horizontal Reynolds, number

$\hookrightarrow Re' = \frac{H^2}{L^2 Re}$: the vertical Reynolds number,

$\hookrightarrow Fr = \frac{\mathbb{U}}{\sqrt{gH}}$: the Froude number.

We chose to exhibit the Rossby number as a small parameter since we are interested in long-time oceanographic circulation for which the Rossby number is typically of magnitude 10^{-2}. The values of Reynolds and Froude numbers vary with respect to the turbulent processes and to the height of the area that is considered respectively.

2.4 An exact solution

For this problem, we don't have a solution which checks the equations written in the previous page. So, we are going to find a solution which checks the initial and the boundary conditions. Finally we have the possible following solutions :

$$\begin{cases} u(\text{t,x,z}) &= t\cdot\sin(\frac{\pi x}{3})\cdot\cos(\pi z) \\ v(\text{t,x,z}) &= t\cdot\sin(\frac{2\pi x}{3})\cdot\cos(\pi z) \\ h(\text{t,x}) &= t\cdot\ \sin(\frac{\pi x}{3}) \end{cases}$$

Thus, injecting these functions in the primitives equations, we don't obtain the expected 0-result, but appears a secund member. We will call f_1, f_2 and f_3 the functions which are respectively solutions of the u-primitive equation, the v-primitive equation and the h-primitive equation :

$$f_1(t,x,z) \ = \ t\Big((\tfrac{10\pi^2}{9})\cos(\pi z)\cdot\sin(\tfrac{\pi x}{3})+\tfrac{\pi}{3}\cos(\pi z)\cdot\cos(\tfrac{\pi x}{3})\text{-}\tfrac{cos(\pi z)}{ee}\ \cdot\sin(\tfrac{2\pi x}{3})\Big)+cos(\pi z)\cdot\sin(\tfrac{\pi x}{3})$$

$$f_2(t,x,z) \ = \ t\Big((\tfrac{13\pi^2}{9})\cos(\pi z)\cdot\sin(\tfrac{2\pi x}{3})+\tfrac{2\pi}{3}\cos(\pi z)\cdot\cos(\tfrac{2\pi x}{3})+\tfrac{cos(\pi z)}{ee}\ \cdot\sin(\tfrac{\pi x}{3})\Big)+cos(\pi z)\cdot\sin(\tfrac{2\pi x}{3})$$

$$f_3(t,x) \ = \ \tfrac{\pi t}{3}\ \cos(\tfrac{\pi x}{3}).$$

Finally, we obtain a new system of equations, $\tilde{E}_p$, whom u, v, h are solutions :

$$\boxed{\begin{cases} \left\{\partial_t + \mathbb{U}_0\cdot\nabla_h - \frac{\Delta_h}{Re} - \frac{\partial_z^2}{Re'} + \frac{C}{\epsilon}\right\}\mathbb{U}_h + \frac{\nabla_h\xi}{Fr^2} &= (f_1,f_2) \\ \partial_z\mathbb{U}(x,y,-1,t) = \partial_z\mathbb{U}(x,y,0,t) &= 0 \\ \{\partial_t + \mathbb{U}_0\cdot\nabla_h\}\,\xi + \nabla_h\cdot\overline{\mathbb{U}}_h &= f_3 \end{cases}}$$

$$\text{where C} = \overline{\mathbb{U}}_h = \begin{pmatrix}\overline{u}\\ \overline{v}\end{pmatrix} := \tfrac{1}{H}\int_{-H}^{0}\mathbb{U}_h\,dz.$$

3 The Set of Equations : the Mathematical Model

3.1 The space discretization of the domain

We first describe the space discretization of the domain. We consider a regular cartesian grid of $n_x \times n_z$ points and we apply a finite volume method. We introduce the horizontal space step Δx and the vertical space step Δz. Here we are going to deal with the horizontal velocity and we are not going to compute a 3d pressure but a 2d water height. Note the all velocities can be computed on the same cells since we consider a 2d (x, z) problem for the horizontal velocity (u, v) in the (x, y) plane. We thus have to introduce two types of finite volume meshes (Figure 1).

The first one is a 2d finite volume mesh and is related to the computation of the velocities. For i $= 0 \cdots n_x - 1$ and j $= 0 \cdots n_z$, we denote $I = i + jn_x$. The cells of this first mesh will be denoted $C_I = X_I + (-\frac{\Delta x}{2}, \frac{\Delta x}{2}) \times (-\frac{\Delta z}{2}, \frac{\Delta z}{2})$, where the points X_I stand for $X_I = (0, -H) + (i\Delta x, j\Delta z)$ (they are represented by a black circle in Figure 1).

The second grid is a 1d finite volume mesh devoted to the computation of the water height. The cells of this second mesh will be denoted $c_{i+\frac{1}{2}} = x_{i+\frac{1}{2}} + (-\frac{\Delta x}{2}, \frac{\Delta x}{2})$, where the points $x_{i+\frac{1}{2}}$ stand for $x_{i+\frac{1}{2}} = (i + \frac{1}{2})\Delta x$ (they are represented by a circle with a number inside in Figure 1).

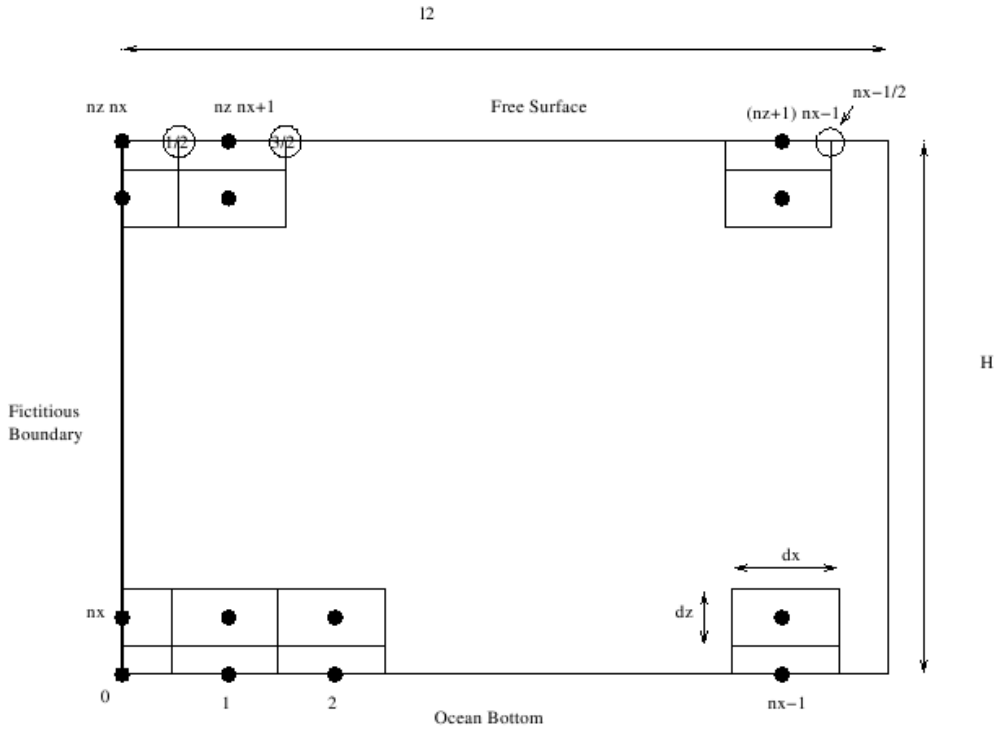


Figure 2: Space discretization of the domain

3.2 A Crank-Nicholson Scheme for the velocity

Let us now consider the discretization of the equations. We integrate the (u, v)-equation on the time-space cell $[t_k,t_{k+1}] \times C_I$. We compute the interface fluxes at time $t_{k+\frac{1}{2}}$ by using classical centered formulas. We recover the well-known Crank-Nicholson scheme. It is known to be second order accurate and conditionnaly stable in the L norm under a CFL type condition on the time step $\Delta t_k = t_{k+1} - t_k$. This strategy is applied for all the velocity nodes such that the neighbouring nodes are included inside the considered subdomain. The discrete relations are :

u : the abscissa of the velocity :

$$u_{I,k} - \frac{\Delta t}{2} \left\{ u_0 \ \frac{u_{I+1,k} - u_{I-1,k}}{\Delta x} - \frac{1}{Re} \ \frac{u_{I+1,k} - 2u_{I,k} + u_{I-1,k}}{2\Delta x} - \frac{1}{Re'} \ \frac{u_{I+1,k} - 2u_{I,k} + u_{I-1,k}}{2\Delta z} - \frac{1}{\epsilon} \ v_{I,k} + \frac{1}{Fr^2} \ \overline{D_{x1}} \ \xi_{j,k} \right\}$$

v : the ordinate of the velocity :

$$v_{I,k} - \frac{\Delta t}{2} \left\{ u_0 \ \frac{v_{I+1,k} - v_{I-1,k}}{\Delta x} - \frac{1}{Re} \ \frac{v_{I+1,k} - 2v_{I,k} + v_{I-1,k}}{2\Delta x} - \frac{1}{Re'} \ \frac{v_{I+1,k} - 2v_{I,k} + v_{I-1,k}}{2\Delta z} + \frac{1}{\epsilon} \ u_{I,k} \right\}$$

3.3 An Upwind Scheme for the height of the water

Let us now consider the equation of the height of the ocean. We integrate it on time space cells $[t_k,t_{k+1}] \times c_{i+\frac{1}{2}}$, except for $i = 0$ where we need to use the transmission conditions. We compute the interface fluxes by using classical explicit upwind formulas. The resulting scheme is known to be first order and also conditionnaly stable under a CFL type condition. The related formula is :

$$\xi_{i+\frac{1}{2},k+1} = \left\{ 1 - \frac{\Delta t}{\Delta x} \ u_0 \right\} \xi_{i+\frac{1}{2},k} + \frac{\Delta t}{\Delta x} \ u_0 \ \xi_{i-\frac{1}{2},k} - \frac{\Delta t \Delta z}{\Delta x} \left(\overline{u}_{i,k}^{+,n+1} - \overline{u}_{i-1,k}^{+,n+1} \right)$$

$$\text{where } \overline{u}_{i,k}^{+,n+1} = \frac{u_{j,k}}{2} + \sum_{j=1}^{n_z-1} u_{j(n_x+1),k} + \frac{u_{n_z(n_x+1),k}}{2}.$$

4 Programming

4.1 Platforms used

During the project, we could have to do numerous computations, in particular to obtain the essential matrixes to visualize the solutions of the equations. That's why, at the beginning of the three weeks, we thought that we will use the ***MATLAB*** software. Indeed, the code proposed was written with Matlab. Matlab is a numerical computing environment and programming language. It allows an easy manipulation of matrix, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs in other languages.

This last point is very important, because thanks to Matlab we can create a graphic interface in Java : libraries written in Java can be directly called from Matlab and many Matlab libraries are implemented as wrappers around Java libraries. However, calling Matlab from Java is more complicated, but can be done with Matlab extension which is sold separately by MathWorks (which is a privately held, mid-size, multi-national corporation which is specialized in technical computing software).

We did not use Matlab because of the price of the software and the additional price for the extension. Hence, we have decided to use ***SCILAB***, because it looks like Matlab, especially regarding a lot of functions and the syntax which are similar. Scilab is a high level programming language in that : most of its functionality is based around the ability to specify many computations with few lines of code. Unlike Matlab, Scilab is available for download at no cost. It is an open source.

Moreover, we can launch remote Matlab/Scilab scripts and visualize results textually with ProActive Interface. The main aim of the ProActive Interface is to equip Scilab and Matlab with a generic interface to Grid computing. This extension allows the deployment of Matlab/Scilab instances on several notes of the grid, to use these instances like computing engines and submitting of Matlab/Scilab tasks over the grid.

To continue, in a Java code, it is possible to call Scilab functions, thanks to the Javasci package proposed by Scilab. Nevertheless, to do an Applet with this package is complicated for our team and for the users -to see the next subsection. As a matter of fact, to configure the Applet on the Internet, a user must install .dll files for Windows or .so files for Linux, some jar files and must update his JRE (Java Runtime environment). This configuration is not easy, so we have decided with Mr. Dreyfuss to translate all the Matlab code in Java.

After the previous decision, we decided to download the ***Java IDE NetBeans***, the 6.5 version. The Netbeans GUI Builder greatly reduces the learning curve and development time needed to produce professional quality Java GUIs, and in particular to create Graphical User Interface with the GUI application named ***Matisse***. Furthermore, NetBeans has the characteristic to be portable. That's why we could use whatever operating system, to know ***Windows XP*** for one of us -Adrien, and ***Linux*** for François.

4.2 Packages loaded

We used and have downloaded a lot of packages. In this section we present the packages and why we have or not used them.

Previously, we have specified the use of ***Javasci package***. It is a Scilab tool to interface Scilab functions to Java. The advantage of this package is the creation of matrixes, and the possibility to do computations with Scilab on Java code. To use this package correctly, it is necessary to import the package and a few files with an extension .dll (dynamic-link library with Windows), .so (shared object with Unix), .c and .h.

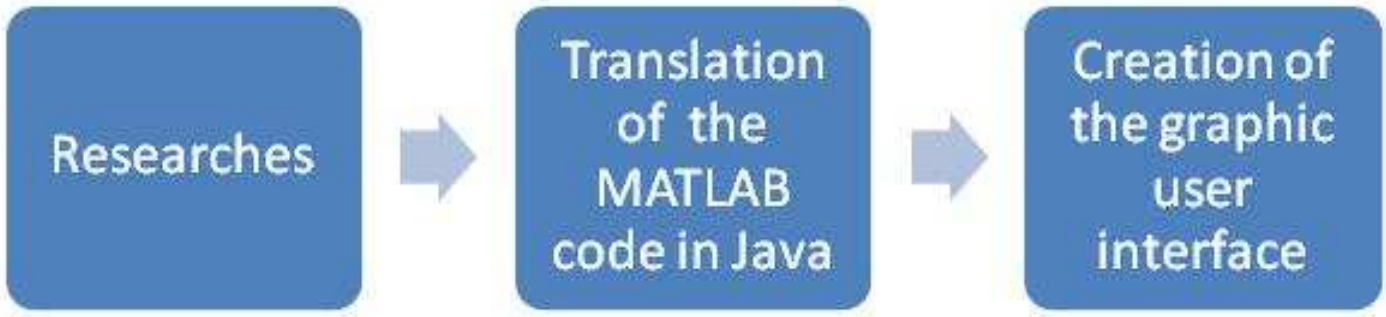
To link dynamic libraries is usually handled by linking to an import library when building or linking to create an executable file. Then, the created executable contains an import address table by which all .dll function calls are referenced. At run-time, the import address table is filled with appropriate addresses that point directly to a function in the separately-loaded 'dll'. It is the same procedure over Unix with files .so.

However, in the Javasci package there are Java's native methods. Simply put, a native method is the Java interface to-non-Java code. It is the Java's link to the «outsider world ». More specially, a native method is a Java method whose implementation is provided by non-Java code, most likely C. Thus, we must import the files with the extension .c, and .h too. We have tried to import this package, but it misses already a file, and the package is not identified. Even if, it is possible to do an Applet (see the web site www.raditha.com/Java/jni/), Mr. Dreyfuss pointed out the difficulties to launch this kind of Applet with native methods.

Finally, as we decided to translate the Matlab code in Java, we have used others Java packages. To create matrixes, we have used the ***UJMP package***. The Universal Java Matrix Package is an open source Java library that provides sparse and dense matrix classes, as well as a large number of calculations for linear algebra like matrix multiplications or the inverse of a matrix, and every matrix can be visualized in a JFrame by invoking the showGUI() method. The advantages of this package are his resources, but also the possibility to create sparse or/and dense matrix.

To create interactive graphics we have used the ***SGT package*** and the ***JFreeChart package***. The Scientific Graphics Toolkit facilitates an easy development of independent platform Java applications to produce highly interactive, flexible, publication quality, object oriented graphics of scientific data. Features include user settable or automatically scaled axes, sophisticated, automatically self-scaling time axes, labels as movable, customizable objects, automatic generation of legends to explain data being displayed... The advantage of this package is the possibility to draw vector field. Secondly, JFreeChart is a free Java chart library that makes it easy for developers to display professional quality charts.

4.3 Evolution of the Project



First week: this week was devoted to the research. Indeed, during this period we have done three kinds of researches. As we have never created neither a graphic user interface or an Applet, we have discovered the «swing » and «awt » libraries proposed by Java, thanks to the web site www.siteduzero.com.

Then, as in a first time we could want to use Scilab, we have searched information to encapsulate the Scilab code in a Java code, and to do the link between the two programming languages. We have also discovered the platform NetBeans.

Second week: this week was devoted to the translation of the code Matlab in Java. Namely, we have created different classes in Java, having the same characteristics and giving the same results as the Matlab scripts. It was a difficult step because the execution of the Java code is less efficient than Matlab or Scilab execution, and a lot of Java package were not complete. And, at the end of this week, the code was not correct.

Third week: this week was devoted to the creation of the «Graphic User Interface » -GUI. But, before beginning the interface we were to finish the translation of the Matlab code in Java. Moreover, we have written the report.

4.4 Problems

We have met lots of problems. Firstly, there was the problem of the software, because we have not Matlab in our laptops. Then, we have discovered that with Scilab it is difficult to execute an Applet on the Internet, because the user must upload and download different files.

During the translation of Matlab code in Java, we have encountered a lot of difficulties. Indeed, with UJMP, it is hard to create matrixes, and there are some methods of it which return the erroneous results. For example, our main difficulties concern the inversion of a matrix or the solving of a linear system $Ax = b$ with A a matrix and x and b two vectors. To solve this problem we have try to create our own method. We have coded several algorithms, therefore Gaussian elimination.

In linear algebra, Gaussian elimination is an efficient algorithm for solving systems of linear equation, finding the rank of a matrix, and calculating the inverse of an invertible square matrix.

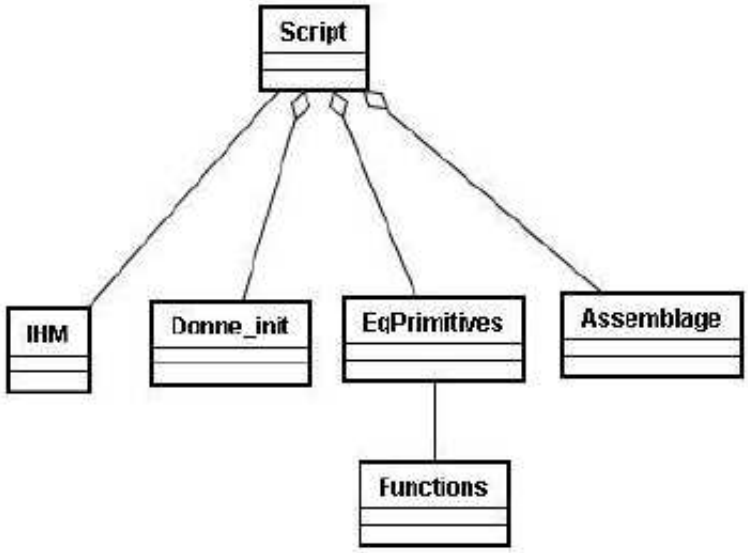
4.5 Mathematics and Programming

4.5.1 Java language versus Matlab

All the code in our project is written in Java language, and is the mirror of the Matlab code. Indeed, we have decided to keep the same names for the functions and for the parameters. It is possible to find differences between each kind the code but, the results are the same. These differences appeared when in Matlab code there are functions of Matlab which are called (like feval, speyes, sparse, etc), or in some calculations in particular, with vectors are simplify thanks to the tools of Matlab.

The translation of the Matlab code is not easy, and the final Java code is not really efficient. For example, our Java machine calculate slowly matrix with an important dimension (99x10) contrary to Matlab or Scilab. A solution proposed by Mr. Dreyfuss was to create sparse matrix. This kind of matrix allows saving only the values different of zero, and the corresponding index. In this case the Random Access Memory (RAM) is less used. Moreover, the calculations between sparse matrixes are quicker. Unfortunately, we have not enough time to implement and to develop this idea.

To understand our code, we represent the architecture of it, thanks to this classes diagram. In this diagram we have written only the name of the classes.



There are 6 codes. The « IHM.java » is our interface and we present the GUI result on the next section. The file Script.java can be considered like the main code. In this code we recover the two matrixes characterizing height and velocities during time. Script.java calls three other objects : Donne_init, EqPrimitives and Assemblage.

⇒ Donne_init : initialization of the several parameters and matrixes,
⇒ EqPrimitives : calculations of the primitive equations to find U (matrix velocities) and a (matrix heights),
⇒ Assemblage : calculations of different matrixes to obtain the two matrixes Ap and Am created in blocks,
⇒ Functions : generation of different functions.

4.5.2 Graphic User Interface (GUI)

Matrixes : our goal is to visualize in a « GUI » the velocity and the height of the ocean –**see the appendix**. Before presenting the interface, we are going to present the structure of the velocity and of the height obtained with the codes. In reality the velocity and the height are characterized by matrixes.

⇒ **The velocity** is represented by the matrix 'U' below, whom size is $2N \times n_t$:

$$U = \begin{pmatrix} u_{0,0} & u_{0,1} & u_{0,2} & \dots & u_{0,n_t} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ \vdots & \vdots & \vdots & u_{i,j} & \vdots \\ \vdots & \vdots & \vdots & \dots & \vdots \\ u_{N-1,0} & u_{N-1,1} & u_{N-1,2} & \dots & u_{N-1,n_t} \\ v_{0,0} & v_{0,1} & v_{0,2} & \dots & v_{0,n_t} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ \vdots & \vdots & \vdots & v_{i,j} & \vdots \\ \vdots & \vdots & \vdots & \dots & \vdots \\ v_{N-1,0} & v_{N-1,1} & v_{N-1,2} & \dots & v_{N-1,n_t} \end{pmatrix}$$

Comment : the velocity U is determined by a group of vectors for each time ; each column represents one time. For one time, there is one vector on one knot of the grid ; each vector is represented by its u-velocity, whom value is given by the term $u_{i,j,t}$, and its v-velocity, whom value is given by the term $v_{i,j,t}$, with the trinomial (i, j, t) given.

⇒ **The height** is represented by the matrix 'a' below, whom size is $\mathrm{n}_x \times n_t + 1$:

$$a = \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \dots & a_{0,n_t} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ \vdots & \vdots & a_{i,j} & \dots & \vdots \\ \vdots & \vdots & \vdots & \dots & \vdots \\ a_{n_x-1,0} & a_{n_x-1,1} & a_{n_x-1,2} & \dots & a_{n_x-1,n_t} \end{pmatrix}$$

Comment : to create an animation we draw all the points of columns functions of time. In a first time we draw on a graphic the point of the first column. We obtain the height of the ocean, namely a curve at time 0 (this value zero corresponds to the index of the column). Each value of a column represents the height at time t, at the abscissa determine by the second index of values. Then this first curve is drawn, and we can draw the value of the second column, etc.

Paramaters : in our GUI, a user must fill several parameters to initiate the animations. He must put only numbers. It is possible to write number with point like '1.23' for instance. If this condition is not respected an information message appeared, and the field is automatically filled by default values. It is important to manage the data given by a user. Indeed, if a user writes a letter instead of a number, an error will be generated, and the applet will have certainly problems while the execution. To reassure our applet we have decided to use regular expressions, named «regex ».

Regular expressions are a way to describe a set of strings based on common characteristics shared by each string in the set. They can be used to edit or manipulate text and data. We have learnt a specific syntax to create our expression. The regex syntax is supported by the java.util.regex API and the code has the following form :

```
jTextField1.getText().matches("\\d*\\.?\\d*")
```

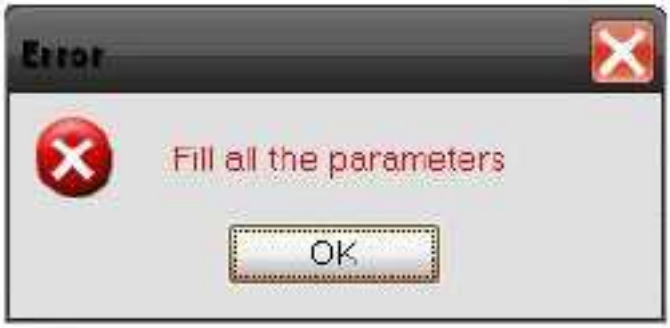
The matches method checks if the text in the jTextField has the good form, given by the regex between the bracket of the matches method :

↪ \\d* means that we can write zero or several numbers
↪ \\.? means that we can put zero or one point

Straight away, we present the parameters. In our GUI, a user must fill five files corresponding to five parameters :

- n_x : this parameter represents the number of points along the X-axis,
- n_z : this parameter represents the number of points along the Z-axis,
- n_t : this parameter represents the number of step-time,
- ee : this parameter is the Rossby number,
- u_0 : this parameter is the initial velocity.

Moreover, if you enter bad parameters or forget to fill a field, then a message appeared. These are the different dialog boxes that you can encounter on the next page :



↪ If you have have forgotten to fill a field

↪ If you have given bad parameters. Then the differents fields are filled with default values.

↪ A user must click on the next button until the simulation is finished namely the value of the parameter n_t is reach.

4.5.3 Using of the GUI

The walk that a user must follow to use our interface is explained thanks to an activities diagram.

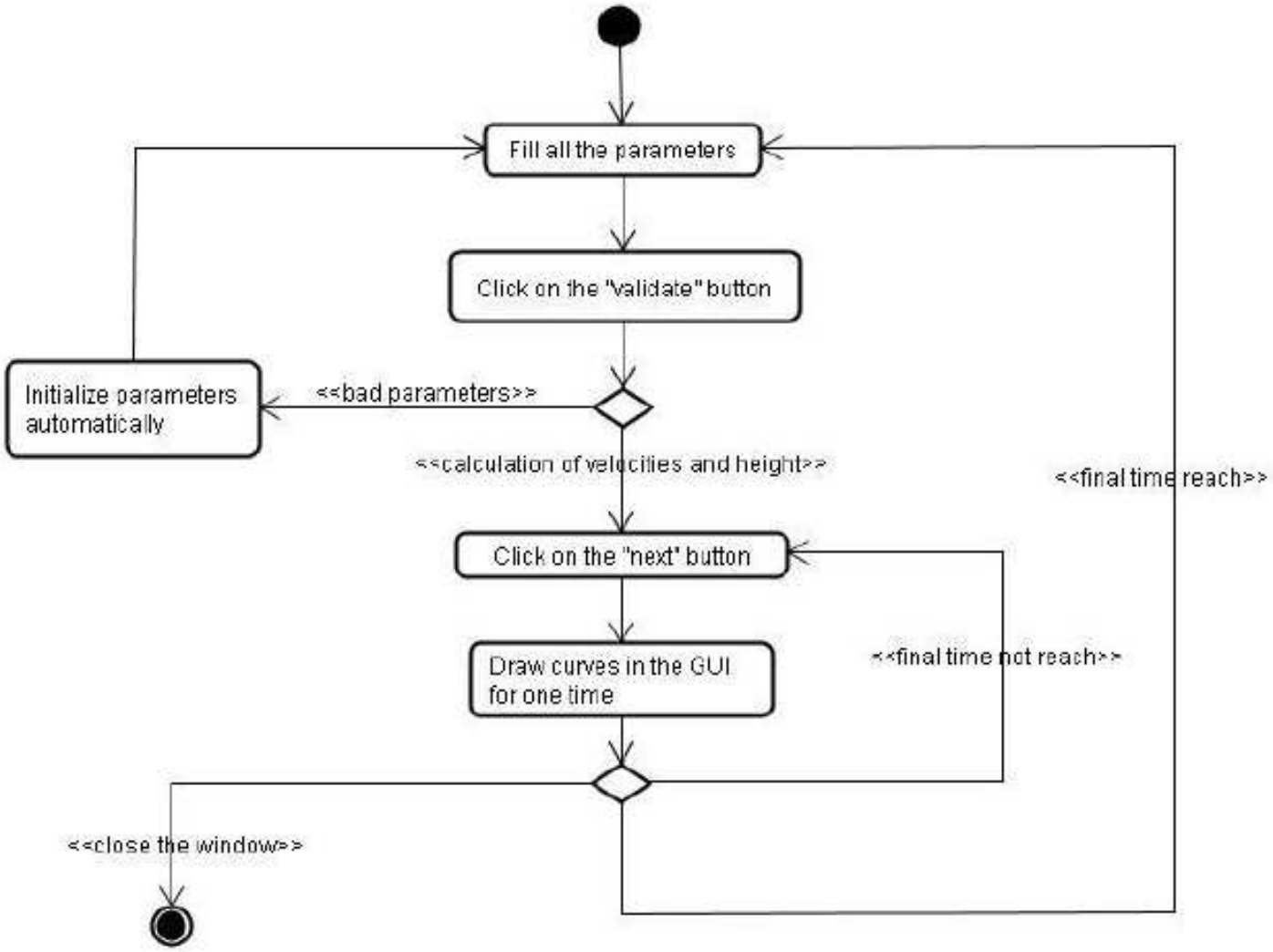


Figure 3: The activities diagram

5 Future and Conclusion

The monodomain model, that we have attempted to imitate, doesn't allow to have right previsions, because of a bad knowledge of boundary conditions. In a large domain, like the Ocean, the researchers hope for improving these results. To do that, the best manner is to divide the domain in several subdomain, and then find the solution in each subdomain. Thus, they call on efficient domain decomposition methods : the Schwarz waveform relaxation type algorithms.

The heart of the classical Schwarz method is to solve the problem on the whole domain thanks to an iterative procedure where a problem is solved on each subdomain by the use of boundary conditions that contain the information coming from the neighbouring subdomains. The correlation between all the solutions optimize the general and final solution.

6 References

6.1 Books and Articles

E. Audusse, P. Dreyfuss, B. Merlet. Optimized Schwarz waveform relaxation for Primitive Equations of the Ocean. January 16, 2009.

Antoine Rousseau. Études Théoriques et Numériques des Équations Primitives de l'Océan sans Viscosité. June 15, 2005.

6.2 Internet Websites

- Matlab : Help online (*www.mathworks.com*)
- Scilab : Help online (*www.scilab.org*)
- Java : Editor, API and Virtual Machine (*java.sun.com*)
- Javasci : The Javasci package (*cermics.enpc.fr/cours/AP/scilab/doc/javasci/package-summary.html*)
- NetBeans : Using of NetBeans 6.5 (*www.netbeans.org*)
- SGT : Help for the Scientific Graphics Toolkit (*www.epic.noaa.gov/java/sgt/*)
- UJMP : Help for the Universal Java Matrix Package (*www.ujmp.org*)

7 Appendix

Presentation of our applet which let the users visualize the velocities and the height of the ocean, with the monodomain technic, on a website.

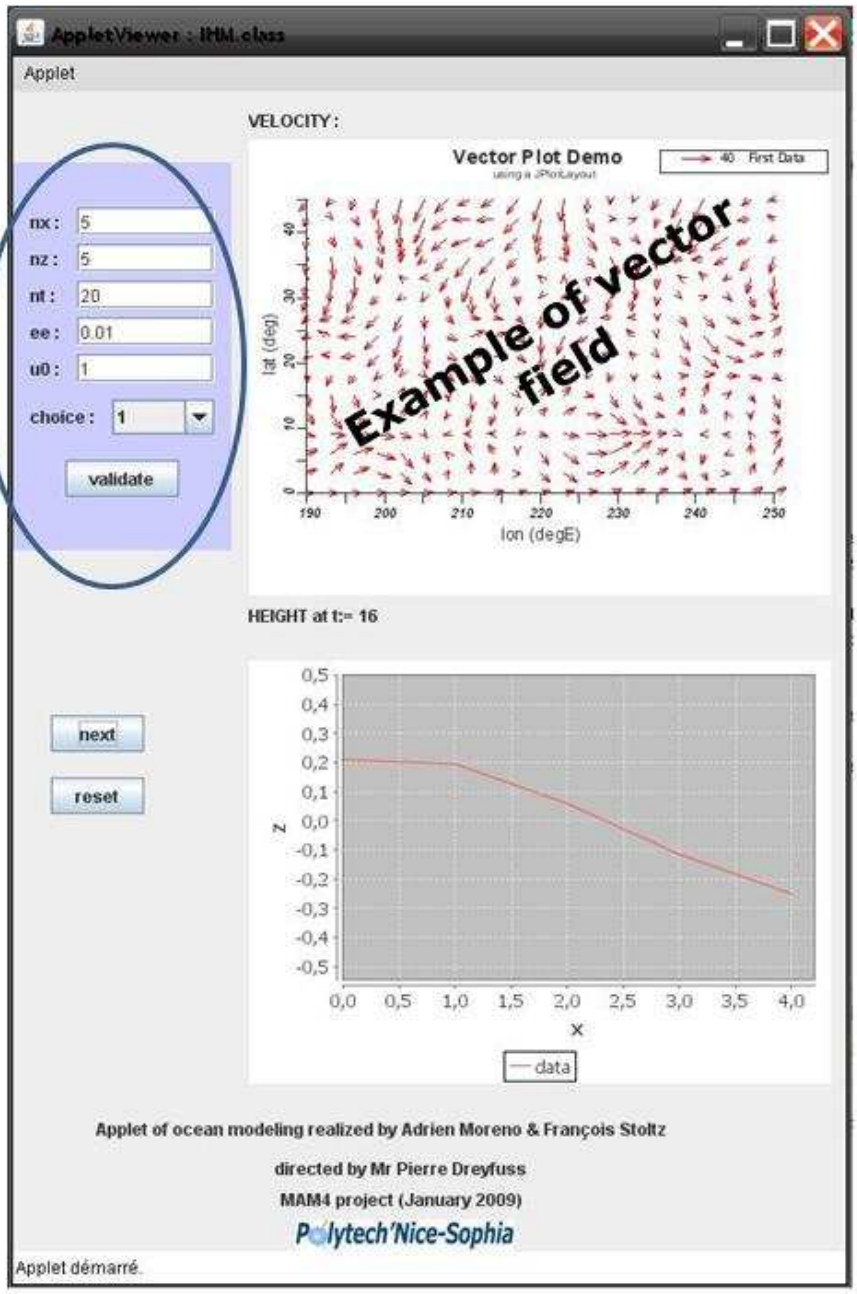


Figure 4: Our applet