CFD with OpenSource Software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

Solvers for Boussinesq shallow water equations

Developed for OpenFOAM-4.0x

Author:
Dimitrios KOUKOUNAS
Department of Mechanics and
Maritime Sciences,
Chalmers University of Technology
dimkou@student.chalmers.se

Peer reviewed by:
Daniel HUMMEL
Mohammad ARABNEJAD

Licensed under CC-BY-NC-SA, https://creativecommons.org/licenses/

Disclaimer: This is a student project work, done as part of a course where OpenFOAM and some other OpenSource software are introduced to the students. Any reader should be aware that it might not be free of errors. Still, it might be useful for someone who would like learn some details similar to the ones presented in the report and in the accompanying files. The material has gone through a review process. The role of the reviewer is to go through the tutorial and make sure that it works, that it is possible to follow, and to some extent correct the writing. The reviewer has no responsibility for the contents.

January 5, 2018
Learning Outcomes

In this project the reader will learn:

How to use it

• How to simulate the evolution of a gaussian hump inside a rectangular tank using the Abbott and the weakly non linear Nwogu equations

The theory of it

• The basic assumptions behind the Boussinesq approximation
• The mathematical approach in deriving the Nwogu equations

How it is implemented

• How the NwoguFoamRK and the AbbottRK solvers are implemented from scratch

How to modify it

• How to change the parameters of the analysis
• How to run the rectangular tank tutorial
• How to post-process the results to obtain useful figures and plots
# Contents

Introduction 3

1 Theory 4  
1.1 The Boussinesq approximation .............................................. 5  
1.2 The Nwogu equations ............................................................. 8  
1.3 The standard Boussinesq model .............................................. 9

2 Implementation 10  
2.1 General ................................................................. 10  
2.2 The NwoguFoamRK.C file ...................................................... 11  
2.3 The AbbottRK.C file ........................................................... 13

3 Tutorial setup 14  
3.1 The Gaussian hump problem .................................................... 14  
3.2 Tutorial Setup ................................................................. 15  
3.3 Running the code .............................................................. 21

4 Post-processing and Results 22  
4.1 Post-processing ............................................................... 22  
4.2 Results ........................................................................... 23  
4.3 Discussion and Conclusions .................................................. 28

Study Questions 29

Appendix A 31
Introduction

This report describes the steps towards a Finite Volume modeling of the Boussinesq non linear shallow water equations. The goals of this work are to:

- Describe the basic concept of the Boussinesq approximation of shallow water waves
- Develop a solver from scratch in order to model terms of special complexity and to use a a 4th order Runge Kutta time integration scheme
- Make a utility to initialize the surface elevation field
- Create a rectangular tank tutorial in order to model a Gaussian hump disturbance as it can be seen in figure 1

Figure 1: A Gaussian hump in the modelled space
Chapter 1

Theory

In the following, the fundamentals of Boussinesq approximation for shallow water waves are discussed in the first subsection. Consequently, the development of the solver NwoguFoamRK follows, with special attention in presenting the various challenges met in the discretization process for the Nwogu equations and the solutions to them.

Dispersion and shallow water waves

One of the most important aspects of the behaviour of water waves is frequency dispersion, i.e. the property that harmonic components of different wavelengths propagate at different velocities. Dispersion helped the explanation of the basic behavioural properties as well as more complex wave phenomena in deep water waves while its importance in the understanding of shallow water waves is equally crucial. Indeed, special wave cases such as the cnoidal waves arise from the interaction between depth-varying dispersion effects and non-linearities arising from high steepness (height to wavelength ratio). Looking at the linear waves which are solutions of the Laplace potential flow equation with the linearized free surface kinematic and dynamic boundary conditions, the dispersion relation for finite depth $h$ is given by

$$\omega^2 = \kappa g \tanh(\kappa h), \quad c_p = \frac{\omega}{\kappa} = \frac{1}{\omega} \tanh(\kappa h)$$

In the equation $1.1$, $c_p$ is the propagation (phase) velocity, $\omega$ is the angular frequency, $h$ is the depth and $\kappa = \frac{2\pi}{l}$ is the wave-number with $l$ being the wavelength. This formula indicates the effect of depth in dispersion. As it can be seen, when the quantity $\kappa h$ approaches infinity, i.e. in case of very short waves or very large depth, the equation becomes unaffected by the depth. On the other hand, in the limit $\kappa h$ going to zero it has been proven that all the waves propagate with the constant velocity $\sqrt{gh}$ independently of their wavelength which means that the dispersion disappears.

In order to understand how dispersion can affect the wave elevation profile we can now think about the process of wave shoaling, i.e. the propagation over a progressively smaller water depth. Although real ocean waves are far more complex than a linear sinusoidal one, they can be considered as being constituted by the superposition of countably or uncountably infinite number of sinusoidal components. A simplified model is to consider a more complex wave as a superposition of linear components each of which has a constant wavelength, amplitude and propagation speed according to the linear dispersion formula described above (in the case of a non linear wave higher order interactions between these parameters as well as between the different components appear). When such a wave model approaches progressively smaller depth, the longest components which propagate faster, are first being slowed down when reaching smaller depth areas. The shorter following components, which are still left behind in larger depth, feel less effect of the depth and are "catching" (in a loose sense) the longest ones. The result of this phenomenon is a shorter steeper overall wave profile, a fact that increases the nonlinear interactions between the components.
After the aforementioned, one can expect the propagation mechanism as well as the dynamic equilibrium in a wave-train propagating over finite depth, to be affected by both nonlinearity and dispersion effects. This fact still holds in the special case of shallow water waves even if the dispersion effect progressively reduces as the depth to wavelength ratio reduces. In order to describe to what extent each effect participates in the dynamic equilibrium of the wave, two important non-dimensional parameters should be introduced. These are the nonlinearity and dispersion parameters \( \delta = \alpha_0 / h_0 \) and \( \mu = \kappa_0 h_0 \), respectively. In these definitions, \( h_0 \) is the depth (or a characteristic depth in the varying depth case), \( k_0 \) is a characteristic wavenumber and \( \alpha_0 \) is a characteristic wave amplitude.

After these considerations, the classical Airy’s Shallow Water Model is accurate in the limiting case of very shallow water waves with very large wavelength to depth ratio \( \frac{l}{d} > 10 \), where the dispersion is negligible. On the other hand, such model’s accuracy is rapidly compromised when applied in problems with lower \( \frac{l}{d} \). This led to extensive research over the years resulting in a big variety of models, all of which taking into account dispersion, with the most important ones being the Boussinesq family of equations, the Korteweg-De Vries equation and the Green-Naghdi model. These are all non linear models based on the assumption of \( O(\delta) < 1 \).

1.1 The Boussinesq approximation

First, let’s pose the problem of the non linear wave propagation over finite varying (or constant) depth mathematically, using the potential flow theory. We seek a solution of the Laplace differential equation with the fully non linear free surface kinematic and dynamic boundary conditions as well as the bottom boundary condition.

\[
\phi_{zz} + \nabla \phi = 0 \\
g\eta + \phi_t + \frac{1}{2} \left( \nabla \phi \right)^2 + \frac{1}{2} \phi_z^2 = 0, \quad z = \eta \\
\eta_t + \nabla \phi \cdot \nabla \eta - \phi_z = 0, \quad z = \eta \\
h_t + \nabla \phi \cdot \nabla h + \phi_z = 0, \quad z = -h
\] (1.2)

In the above equations, \( \phi \) is the velocity potential of the flow, \( \eta \) is the wave elevation from the still surface, \( h \) is the (variable) depth and \( \nabla = \frac{\partial}{\partial x} + \frac{\partial}{\partial y} \). In this model and for the following discussion the origin is located at the still water surface. It is also convenient to express the system of differential equations and BCs in non-dimensional form in order to be able to judge the importance of each term in accordance with the variation of the nonlinearity and dispersion parameters. If we select a characteristic wave amplitude \( \alpha_0 \), a characteristic wavenumber \( \kappa_0 \), a depth \( h_0 \) and a velocity potential \( \phi_0 \) the variables \( x, y, z, \eta \) and \( \phi \) are non-dimensionalized as

\[
(\hat{x}, \hat{y}) = \kappa_0 (x, y), \quad \hat{\eta} = \frac{\eta}{\alpha_0}, \quad \hat{z} = \frac{z}{h_0}, \quad \hat{h} = \frac{h}{h_0}
\]

If we further take into account a characteristic velocity \( c_0 = \sqrt{gh_0} \), then

\[
\hat{t} = (\kappa_0 gh_0) t, \quad \hat{\phi} = (\frac{\alpha_0}{\kappa_0 h_0 \sqrt{gh_0}})^{-1} \phi
\]

And the system of equations becomes
\[ \phi_{zz} + \mu^2 \nabla^2 \phi = 0 \]
\[ \hat{\eta} + \hat{\phi}_t + \frac{1}{2} \delta (\nabla \hat{\phi})^2 + \frac{1}{2} \frac{\delta}{\mu^2} \hat{\phi}_z^2 = 0, \quad z = \delta \hat{\eta} \]
\[ \hat{\phi}_t + \delta \nabla \hat{\phi} \cdot \nabla \hat{\eta} - \frac{1}{\mu^2} \hat{\phi}_z = 0, \quad z = \delta \hat{\eta} \]
\[ \mu^2 \nabla \hat{\phi} \cdot \nabla \hat{h} + \hat{\phi}_z = 0, \quad z = -\hat{h} \] (1.3)

The key assumptions behind the Boussinesq approximation are the following:

- long wave assumption (i.e. wavelength is large compared to the depth)
- small nonlinearity parameter i.e. \( O(\delta) << 1 \)
- dispersion effects equally important with the non-linear interaction effects, i.e. \( O(\mu^2) \approx O(\delta) \)

After these assumptions, all the terms in the non linear boundary conditions are equally important and should be taken into account. Moreover, a process should be followed in order from the original system of equations 1.2 or 1.3 to derive a system of two differential equations one for the wave elevation and the other either for the velocity potential \( \phi(x, y, z, t) \) or the velocity \( \mathbf{u}(x, y, z, t) \). In addition, it is more convenient and computationally efficient to eliminate the dependency on depth which can be done by either averaging or integrating the equation over the depth, or alternatively by selecting a reference water depth \( z_0 \) in which the 2D velocity \( \mathbf{u}_d(x, y, t) \) to be computed.

For the derivation of the Boussinesq models, we can proceed in two ways. Either by starting from the system of equations 1.2 and applying the perturbation approach or starting from the non dimensional system, 1.3. In the former case the dependence of the velocity on depth is assumed to be quadratic, while in the latter approach the velocity potential is expanded in power series with respect to the depth. The most important steps of the latter approach are going to be presented here, following the process by Chen and Liu presented by Wei and Kirby, [1]. The hats will be dropped for the non dimensionalized variables for convenience.

The velocity potential is expanded in power series with respect to depth

\[ \phi(x, y, z, t) = \sum_{n=0}^{\infty} (z + h)^n \phi_n(x, y, t) \] (1.4)

Consequently, the gradient and laplacian of the velocity potential become

\[ \nabla \phi = \sum_{n=0}^{\infty} (z + h)^n \nabla \phi_n + \sum_{n=1}^{\infty} n(z + h)^{n-1}(\nabla h) \phi_n = \sum_{n=0}^{\infty} (z + h)^n (\nabla \phi_n + (n + 1)(\nabla h) \phi_{n+1}) \] (1.5)

\[ \nabla^2 \phi = \sum_{n=0}^{\infty} (z + h)^n \nabla^2 \phi_n + \sum_{n=1}^{\infty} n(z + h)^{n-1} \nabla h \nabla \phi_n + \sum_{n=1}^{\infty} n(z + h)^{n-1}(\nabla^2 h) \phi_n \]
\[ + \sum_{n=1}^{\infty} n(z + h)^{n-1} \nabla h \nabla \phi_n + \sum_{n=2}^{\infty} n(n - 1)(z + h)^{n-2}(\nabla h)^2 \phi_n \] (1.6)

Equation 1.6 after some rearrangements becomes:

\[ \nabla^2 \phi = \sum_{n=0}^{\infty} (z + h)^n (\nabla^2 \phi_n + 2(n + 1) \nabla h \nabla \phi_{n+1} + (n + 1) \phi_{n+1} \nabla^2 h + (n + 2)(n + 1) \phi_{n+2}(\nabla h)^2) \] (1.7)
Derivation with respect to $z$ gives:

$$
\phi_z = \sum_{n=0}^{\infty} (n+1)(z+h)^n \phi_{n+1}
$$

(1.8)

$$
\phi_{zz} = \sum_{n=0}^{\infty} (n+2)(n+1)(z+h)^n \phi_{n+2}
$$

(1.9)

In continue, the above formulas are substituted first in the bottom boundary condition to give a relation between $\phi_0$ and $\phi_1$

$$
\mu^2 \nabla h \left( \sum_{n=0}^{\infty} (z+h)^n (\nabla \phi_n + (n+1) \nabla h \nabla \phi_{n+1}) \right) + \sum_{n=0}^{\infty} (n+1)(z+h)^n \phi_{n+1} = 0
$$

For $n=0$

$$
\mu^2 \nabla h (\nabla \phi_0 + \nabla h \phi_1) + \phi_1 = 0
$$

$$
\phi_1 = -\frac{\mu^2 \nabla h \nabla \phi_0}{1 + \mu^2 (\nabla h)^2}
$$

(1.10)

Substitution of the formulas 1.7 and 1.9 into the Laplace equation yields:

$$
\mu^2 \left( \sum_{n=0}^{\infty} (z+h)^n (\nabla^2 \phi_n + 2(n+1) \nabla h \nabla \phi_{n+1} + (n+1) \phi_{n+1} \nabla^2 h + (n+2)(n+1) \phi_{n+2} (\nabla h)^2) + \sum_{n=0}^{\infty} (n+2)(n+1)(z+h)^n \phi_{n+2} \right) = 0
$$

$$
\phi_{n+2} = -\frac{\mu^2 (\nabla^2 \phi_n + (n+1) \nabla h \nabla \phi_{n+1} + (n+1) \nabla (\phi_{n+1} \nabla h))}{(n+2)(n+1)(\mu^2 (\nabla h)^2 + 1)}
$$

(1.11)

So, the horizontally varying components $\phi_n(x,y,t)$ exhibit a recursive relationship between them, meaning that we can calculate one component from its predecessors. Moreover, at every step of the process multiplication by $\mu^2$ takes place so every component is smaller compared to its predecessors if $\mu^2 < 1$. Lets now calculate $\phi_2$ and $\phi_3$

$$
\phi_2 = -\frac{\mu^2 (\nabla^2 \phi_0 + \nabla h \nabla \phi_1 + \nabla (\phi_1 \nabla h))}{2(\mu^2 (\nabla h)^2 + 1)}
$$

$$
\phi_3 = -\frac{\mu^2 (\nabla^2 \phi_1 + 2\nabla h \nabla \phi_2 + 2\nabla (\phi_2 \nabla h))}{6(\mu^2 (\nabla h)^2 + 1)}
$$

(1.12)

If we assume the gradient of the bathymetry to be small, i.e $(\nabla h) \simeq O(\mu^2)$, after substituting $\phi_1$ into the expression for $\phi_2$

$$
\phi_2 = -\frac{\mu^2 (\nabla^2 \phi_0 - \mu^2 \nabla h \nabla (\nabla h \nabla \phi_0)) - \mu^2 \nabla (\nabla \phi_0 (\nabla h)^2))}{2} = -\frac{\mu^2}{2} \nabla^2 \phi_0 + O(\mu^4)
$$

(1.13)
As it can be seen from the expression for $\phi_3$ in the equation 1.12, all the terms in the numerator are of order $O(\mu^2)$ and are multiplied by $\mu^2$ which leads to $\phi_3 \simeq O(\mu^4)$. So, if we want to keep only the terms of the power series for $\phi$ which are of order at max $O(\mu^2)$ then

$$\phi(x, y, z, t) = \phi_0(x, y, t) - (z + h)\mu^2 \nabla h \nabla \phi_0(x, y, t) - (z + h)^2 \frac{\mu^2}{2} \nabla^2 \phi_0(x, y, t)$$ (1.14)

In this formula, $\phi_0$ is the potential at the bottom. We can calculate the potential at arbitrary depth $z_a$ from the above formula as

$$\phi_a(x, y, z = z_a, t) = \phi_0 - \mu^2(z_a + h)\nabla h \nabla \phi_0 - \frac{\mu^2}{2}(z_a + h)^2 \nabla^2 \phi_0(x, y, t)$$

Substituting the above equation for $\phi_0$ in the equation 1.14 and keeping only the terms up to $O(\mu^2)$ results as follows

$$\phi = \phi_a + \mu^2(z_a + h)\nabla h \nabla \phi_a + \frac{\mu^2}{2}(z_a + h)^2 \nabla^2 \phi_a$$

$$- \mu^2(z + h)\nabla h \nabla \phi_a - \frac{\mu^2}{2}(z + h)^2 \nabla^2 \phi_a + O(\mu^4)$$

$$\phi = \phi_a + \mu^2(z_a - z)\nabla h \nabla \phi_a + \frac{\mu^2}{2}(z_a^2 - z^2)^2 \nabla^2 \phi_a + \mu^2(z_a - z)h \nabla^2 \phi_a + O(\mu^4)$$

So, finally we get

$$\phi = \phi_a + \mu^2(z_a - z)\nabla (h \nabla \phi_a) + \frac{\mu^2}{2}(z_a^2 - z^2)^2 \nabla^2 \phi_a + O(\mu^4)$$ (1.15)

### 1.2 The Nwogu equations

By substituting the above equation 1.15 for $\phi$ in the Laplace equation with the system of BCs 1.3, then integrating the Laplace equation with respect to the depth we obtain the so-called Nwogu fully non linear shallow water equations which belongs to the family of extended Boussinesq models. Then, in order to simplify the problem we can neglect terms of $O(\delta \mu^2)$, thus arriving in the Nwogu weakly non linear system. The non dimensional Nwogu weakly nonlinear shallow water equations are as follows

$$\mathbf{u}_{at} + \delta \mathbf{u}_a \cdot \nabla \mathbf{u}_{at} + \nabla \eta + \mu^2 \frac{z_a^2}{2} \nabla \mathbf{u} \cdot \mathbf{u}_{at} + z_a \nabla \mathbf{u} \cdot h \mathbf{u}_{at} = 0$$ (1.16)

$$\mathbf{M} = (h + \delta \eta) \mathbf{u}_a + \mu^2 h \left( \frac{z_a^2}{2} - \frac{h^2}{6} \right) \nabla \nabla \mathbf{u}_a + \mu^2 h (z_a + \frac{h}{2}) \nabla h \mathbf{u}_a$$ (1.17)

$$\frac{\partial \eta}{\partial t} = -\nabla \mathbf{M}$$ (1.18)

In these equations $h$ is the depth, $\mathbf{u}_a$ is the 2D (in the horizontal x and y directions) velocity vector in the reference depth $z_a$, $\eta$ is the surface elevation, $\delta$ is the nonlinearity parameter and
\[ \mu^2 \] the dispersion parameter. The depth \( z_a \) can be selected arbitrarily but according to Chen and Liu, [1] the optimum value for it is \(-0.531h\). This reference depth is thus used in the present project.

### 1.3 The standard Boussinesq model

The first work in introducing dispersion in the shallow water equations is accredited to Boussinesq. In more recent days, using the perturbation asymptotic approximation, Peregrine managed to improve the properties of the Boussinesq model. In the same manner, Madsen and Sorrensen added some additional third order terms to improve the model even more, resulting in the following extended Boussinesq system of equations

\[
\begin{align*}
\mathbf{u}_t + \mathbf{u} \cdot \nabla \mathbf{u} + g \nabla \eta &= \frac{h}{2} \nabla \nabla \cdot (h \mathbf{u}_t) + (B h^2 - \frac{h^2}{6}) \nabla \nabla \cdot \mathbf{u}_t + B h^2 g \nabla (\nabla^2 \eta) \\
\eta_t &= -\nabla [(h + \eta) \mathbf{u}] 
\end{align*}
\]  

(1.19)  

(1.20)

In these equations if we set \( B = 0 \) we obtain the Abbott equations while if \( B = \frac{1}{15} \) we obtain the Madsen Sorrensen model. In these equations \( \mathbf{u} \) represents the depth averaged velocity. Moreover, it can be seen that if we neglect the right hand side terms of equation 1.19 we obtain the Airy’s non linear shallow water model, thus the additional right hand side terms are the ones which account for dispersion.

In this project we set \( B = 0 \) modeling the Abbott equations. In the same solver if one sets the \( B = \frac{1}{15} \) the solver turns to the Madsen Sorrensen model which should give results very close to the Abbott and Nwogu.
Chapter 2

Implementation

2.1 General

In this project, two solvers are developed from scratch to model the slightly non linear Nwogu equations and the Abbott equations. The files contained in the solver folders are presented in figure 2.1.

For the development of both solvers, the finiteVolume and meshTools libraries should be included in each solver’s Make/options file. This inclusion is as follows:

```
EXE_INC = \n   -I$(LIB_SRC)/finiteVolume/lnInclude \n   -I$(LIB_SRC)/meshTools/lnInclude
```

```
EXE_LIBS = \n   -lfiniteVolume \n   -lmeshTools
```

Moreover, the path of the bin files of each solver should be declared in each solver’s Make/files file. For the NwoguFoamRK solver this is

```
NwoguFoamRK.C
```

```
EXE = $(FOAM_USER_APPBIN)/NwoguFoamRK
```
And for the AbbottRK
AbbottRK.C

\texttt{EXE} = $(\text{FOAM\_USER\_APPBIN})/AbbottRK$

The file \texttt{createFields.H} contains the definition of the three important fields, namely the velocity field "Ua", the acceleration field "Uat" and the surface elevation field "eta". These are the same for the two solvers so the \texttt{createFields.H} files of both solvers are identical. Moreover, the \texttt{readCoefficientsAndConstants.H} and the \texttt{readGravitationalAcceleration.H} files are also identical for the two solvers. The former reads a dictionary called "coefficientsAndConstants" where all the analysis parameters are defined. The latter is used to read only the gravitational acceleration $g$ from the dictionary "gravitationalProperties".

In the rest of this chapter, a careful look inside the \texttt{NwoguFoamRK.C} and \texttt{AbbottRK.C} files will take place. The included .H files are not going to be presented here, but they can be found in the appendix A.

2.2 The NwoguFoamRK.C file

Both solvers in this project are implemented using a fourth order Runge-Kutta method of 2N storage. Since this part of the code is identical for both solvers and it is the core of the present approach, it is going to be discussed first.

The RK loop is implemented as follows

```c
countRK = 0;
for (countRK=0; countRK<5; countRK++)
{
    UaRes = rk4a[countRK]*UaRes;
    etaRes = rk4a[countRK]*etaRes;

    UaRes = UaRes + dt*CalculateRHSUa( Uat, eta, Ua, g, h, timeDerCoef);
    etaRes = etaRes + dt*CalculateRHSeta( eta, Ua, za, h, timeDerCoef);

    Ua = Ua + rk4b[countRK]*UaRes;
    eta = eta + rk4b[countRK]*etaRes;

    eta.correctBoundaryConditions();
    Ua.correctBoundaryConditions();
}
```

In this piece of code, the functions "CalculateRHSU" and "CalculateRHSeta" calculate the time derivatives $\frac{\partial u_a}{\partial t}$ and $\frac{\partial \eta}{\partial t}$ respectively. The variable "countRK" is used to count the Runge-Kutta steps and it was initialized in the beginning of the main() function as an integer number. The Runge-Kutta coefficients "rk4a" and "rk4b" were initialized in the beginning of the main() function as float arrays of rational numbers. The interested reader is directed to the Appendix A for the complete code with all the variable declarations.

At the end of the loop, the fields $u_a$ and $\eta$ are subjected to correction of boundary conditions. This is in order to update their patch fields manually (using the specified BCs and the field of neighbour cells) because, for example, when the $\eta$ is calculated its patch field is not updated since it is solved explicitly. The same problem would apply also in $u_{at}$ if it would be calculated explicitly but this is not the case as it will be seen later. When this $\eta$ is then used in the above code to update the $\eta$
field in time, only the cell values of $\eta$ field are updated and not its patch field. Then, when this $\eta$ is used to calculate the $\eta_t$ of the next RK step, then there is an error in calculation of divergence and gradients in the near-patch cells. Another important fact in the above implementation is that the required $u_{at}$ in the "CalculateRHSU" function, is the value of the previous time step, not the one of the previous RK step. This probably compromises the $4^{th}$ order nature of the time integration scheme and it is should be a subject of future study.

The CalculateRHSU function

The calculation of $u_{at}$ takes place in this function using an implicit scheme. The implementation of this function was challenging due to some technical difficulties arising from the equations. A careful look at the equation 1.16 reveals two problems. First, the quantity of interest $u_{at}$ appears also in complex terms of the form $\nabla(\nabla \cdot u_{at})$ and second, these terms are gradient of divergence of the $u_{at}$ field and thus difficult to be discretized using an implicit FV scheme. To overcome this problem, the following idea is applied:

$$u_{at} + \left( \frac{\sigma^2}{2} + h\sigma_{at} \right) \nabla^2 u_{at} = -g\nabla \eta - \frac{1}{2} \nabla(u_a \cdot u_a) - \left( \frac{\sigma^2}{2} + h\sigma_{at} \right) \nabla \cdot u_{at}$$

(2.1)

As it can be seen from this formula, the term containing $\nabla \cdot u_{at}$ can be decomposed into two terms i.e. the laplacian $\nabla^2 u_{at}$ which is held in the left side discretized implicitly and the remaining term in the right side which is the difference $\nabla \cdot u_{at} - \nabla^2 u_{at}$ to be discretized explicitly. For that reason, for the calculation of the $u_{at}^{n-1}$ value of the previous time step should be introduced as an argument in the CalculateRHSU function. Moreover, the velocity transport term $u_a \nabla u_a$ has been modeled as $\frac{1}{2} \nabla (u_a \cdot u_a)$ because

$$u_a \nabla u_a = \frac{1}{2} \nabla (u_a \cdot u_a) + u_a \times \text{rot}(u_a)$$

Due to the inviscid, irrotational flow assumption, the second term in the right hand size of the above equation is zero. The OpenFOAM implementation of these ideas is as follows:

```cpp
solve( fvm::Sp(C,Uat) + (sqr(za)/2 + za*h)*fvm::laplacian(Uat) ==
    - 0.5*fvc::grad(Ua & Ua) - g*fvc::grad(eta)
    - (sqr(za)/2 + za*h)*fvc::grad(fvc::div(Uat))
    + (sqr(za)/2 + za*h)*fvc::laplacian(Uat) );
```

In this implementation, the coefficient C is a dimensionedScalar with all dimensions set to zero and a value of 1.

The CalculateRHSEta function

This function calculates the $\eta_t$ explicitly. Substitution of the equation 1.17 into 1.18 yields an equation for $\eta_t$, and its OpenFOAM implementation is as follows:

```cpp
dimensionedScalar C1("C1", dimensionSet(0,0,-1,0,0,0,0), 0);
volScalarField etat ("etat", C1*eta );
solve
( fvm::Sp(timeDerCoef,etat) == - fvc::div( (h + eta)*Ua
    + h*(sqr(za)/2 - sqr(h)/6)*fvc::grad(fvc::div(Ua))
```
2.3. THE ABBOTT.RK.C FILE

The solve command solves a diagonal system to calculate the \( \eta_t \). If instead we substitute the \( \text{Sp} \) term in the left hand side with just \( \eta_t \) and remove the solve command we should normally arrive in the same result. The constant \( \text{timeDerCoef} \) which is used in the implicit discretization of the source term is a dimensionedScalar with all dimensions set to zero and value 1.

2.3 The AbbottRK.C file

This application is implemented in the same way like the NwoguFoamRK one, but instead, using the Abbott equations. Thus the only difference is the implementation of the functions "CalculateRHSU" and "CalculateRHSeta".

The CalculateRHSU function

The same implicit implementation is used also to calculate the \( u_t \) field

\[
\begin{align*}
  u_t - (B + \frac{1}{3})h^2 \nabla^2 u_t &= -g \nabla \eta - \frac{1}{2} \nabla (u \cdot u) + (B + \frac{1}{3})h^2 \nabla \cdot u_t \\
  &\quad - (B + \frac{1}{3})h^2 \nabla^2 u_t + Bgh^2 \nabla (\nabla^2 \eta)
\end{align*}
\]  

(2.2)

The OpenFOAM implementation is as follows:

```cpp
solve( fvm::Sp(C,Uat) - (B+(1.0/3.0))*sqr(h)*fvm::laplacian(Uat) == 
  - 0.5*fvc::grad(Ua & Ua) - g*fvc::grad(eta) 
  + (B+(1.0/3.0))*sqr(h)*fvc::grad(fvc::div(Uat)) 
  - (B+(1.0/3.0))*sqr(h)*fvc::laplacian(Uat) 
  +B*g*sqr(h)*fvc::grad( fvc::laplacian(eta) ) );
```

In this implementation, the name Ua is used to describe the volVectorField \( u \) and the Uat is used for \( u_t \). Also \( B = \frac{1}{15} \).

The CalculateRHSeta function

This calculates explicitly the \( \eta_t \)

\[
\eta_t = -\nabla[(h + \eta)u]
\]

(2.3)

OpenFOAM implementation:

```cpp
dimensionedScalar C1("C1", dimensionSet(0,0,-1,0,0,0,0), 0); volScalarField etat ("etat", C1*eta);

solve ( fvm::Sp(timeDerCoef,etat) == - fvc::div( (eta+h)*Ua) );
```

In the above code, the dimensionedScalar C1 is used to initialize the volVectorField etat field from the eta one with the correct dimensions. The C1 corrects the dimensions. This initialization similarly takes place in the case of NwoguFoamRK solver.
Chapter 3

Tutorial setup

3.1 The Gaussian hump problem

Problem description

In this project, a basic tutorial of wave evolution inside a rectangular tank is simulated using the solvers for Nwogu and Abbott equations. The rectangular tank has dimensions 10m x 10m and depth of 0.5 m. The problem domain was discretized using 10cm x 10cm elements in order to have exactly the same settings as the ones in FUNWAVE test case described by Wei and Kirby (1998), [1]. This is important in order to compare the results with the ones given in the article for FUNWAVE in order to validate the Finite Volume solvers. Moreover, a simulation with denser mesh is carried out in order to look at the behaviour of the solution as it will be mentioned in the "Results" subsection. In order to simulate the problem of Gaussian hump evolution inside a rectangular tank the following parameters should be specified

• the gravitational acceleration "g"
• the depth "h"
• the longitudinal extent of the domain, "length"
• the transverse extent of the domain, "breadth"
• the reference depth for the Nwogu method "za". This depth plays no role in the case if the Abbott solver is used.
• the height of the gaussian hump "alpha0"
• the shape coefficient of the gaussian hump, "beta"
• the dimensionedScalar dUadtCoefficient which is equal to 1 and is used by the solver to construct the implicit etat source term.

These parameters are being defined in two dictionaries, the "gravitationalProperties" for g and the "coefficientsAndConstants" for the rest of the parameters.

Initial condition

The shape of the initial disturbance in the water surface is the one described as the Gaussian Hump and is given by the following formula:

\[
\eta = a_0 \exp\left[-\beta ((x-x_0)^2 + (y-y_0)^2)\right]
\]  (3.1)
In this equation, the height of the disturbance $\alpha_0$ is set to 0.1m and the shape parameter $\beta = 0.4$. Also $x_0 = 5m$, $y_0 = 5m$ is the center of the hump, to coincide with the center of the domain. The fields of velocity $u_a$ and time derivative of velocity $u_{at}$ are initially set to uniform $[0 0 0]$ vector field.

**Boundary conditions**

The boundary condition for $\eta$ is the zero gradient and for the velocity field is set to slip which means that the velocity component vertical to the wall is set to zero. For the variable $u_{at}$ the slip boundary condition is also used.

**Time step**

In the present analysis, a time step of 0.01 is used which is a bit smaller than the one used by Wei and Kirby in the tests of FUNWAVE software, [1]. Moreover, the solvers developed here admit constant value of time step which is introduced in the controlDict dictionary. Alternatively, the CFL criterion can be used in order to calculate the maximum time step for stability using the relation

$$C = \frac{|u| \Delta t}{\delta x} \leq C_{max}$$  \hspace{2cm} (3.2)

where $C$ is the Courant number and $C_{max}$ depends on the problem, usually $C_{max} = 1$.

### 3.2 Tutorial Setup

The tutorial folder contains 3 sub-folders, namely 0, constant, and system. The tree structure of the case folder BoussinesqTutorial is shown in the figure 3.1.

**Figure 3.1: The file tree for the case BoussinesqTutorial**

It should be mentioned here that if one wants to run the case with the Abbott solver, the above structure as well as the files is exactly the same, a thing that was a target of the project for flexibility. The "0" folder should contain the initial data of 3 fields-i.e. $\eta$, $U_a$ and $U_{at}$. Those are presented in the following.

For $\eta$: 

\[ \]
3.2. TUTORIAL SETUP

FoamFile
{
    version 2.0;
    format ascii;
    class volScalarField;
    location "0";
    object eta;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions [0 1 0 0 0 0];

internalField uniform 0;

boundaryField
{
    sides
    {
        type zeroGradient;
    }
    inlet
    {
        type zeroGradient;
    }
    outlet
    {
        type zeroGradient;
    }
    frontAndBack
    {
        type empty;
    }
}

// ************************************************************************* //

NOTE: The initial definition of eta field is also uniform zero. In order to generate the initial data, the "GaussHump" utility should be run as it is going to be described later. This utility generates a folder "1" with the updated eta field. This folder should be renamed to "0" and then the problem can be simulated. For the rest of files, Ua and Uat, the vector fields remain uniform as follows. For Ua:

FoamFile
{
    version 2.0;
    format ascii;
    class volVectorField;
    location "0";
    object Ua;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions [0 1 -1 0 0 0];
internalField  uniform (0 0 0);

boundaryField
{
  sides
  {
    type  slip;
  }
  inlet
  {
    type  slip;
  }
  outlet
  {
    type  slip;
  }
  frontAndBack
  {
    type  empty;
  }
}

// ************************************************************************* //

And for Uat:

FoamFile
{
    version 2.0;
    format ascii;
    class volVectorField;
    location "0";
    object Uat;
}

// *----------------------------------------------------------------------- //

dimensions  [0 1 -2 0 0 0 0];

internalField  uniform (0 0 0);

boundaryField
{
  sides
  {
    type  slip;
  }
  inlet
  {
    type  slip;
  }
  outlet
  {

In the above setup, see the type "empty" in the front and back patches since we solve for a 2D problem.

In the blockMeshDict dictionary, a standard box shaped block should be constructed with dimensions 10x10 meters. The vertical dimension is set to 0.1 in this case.

In the fvSolution dictionary, the solvers for etat, Ua and Uat systems should be specified. This is done as follows:

```
FoamFile
{
    version 2.0;
    format ascii;
    class dictionary;
    object fvSolution;
}
```

```
// ************************************************************************* //
```

```
solvers
{
    Uat
    {
        solver PCG;
        preconditioner DIC;
        tolerance 1e-10;
        relTol 0.01;
    }
    etat
    {
        solver PCG;
        preconditioner DIC;
        tolerance 1e-10;
        relTol 0.01;
    }
    Ua
    {
        solver PCG;
        preconditioner DIC;
        tolerance 1e-10;
        relTol 0.01;
    }
}
```

```
// ************************************************************************* //
```
In the fvSchemes dictionary, the discretization schemes for the gradient, the divergence and the laplacian should be specified as follows

```plaintext
FoamFile
{
    version 2.0;
    format ascii;
    class dictionary;
    object fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

gradSchemes
{
    default Gauss linear;
}

divSchemes
{
    default Gauss linear;
}

laplacianSchemes
{
    default Gauss linear uncorrected;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

The controlDict dictionary is as follows

```plaintext
FoamFile
{
    version 2.0;
    format ascii;
    class dictionary;
    object controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

application NwoguFoamRK;
startFrom startTime;
startTime 0;
stopAt endTime;
endTime 50;
deltaT 0.01;
writeControl timeStep;
```
writeInterval 10;
purgeWrite 0;
writeFormat ascii;
writePrecision 6;
writeCompression off;
timeFormat general;
timePrecision 6;
runTimeModifiable true;

Finally, in order to specify the analysis parameters, the following dictionaries should be created. For the gravitational acceleration the "gravitationalProperties" dictionary:

```
FoamFile
{
    version 2.0;
    format ascii;
    class dictionary;
    object gravitationalProperties;
}

// *************************************************** //
g  
    [0 1 -2 0 0 0 0] 9.81;
// *************************************************** //
```

For the rest of parameters, the "coefficientsAndConstants" dictionary

```
FoamFile
{
    version 2.0;
    format ascii;
    class dictionary;
    location "0";
    object coefficientsAndConstants;
}

// *************************************************** //
dUadtdCoefficient  
dUadtdCoefficient  [0 0 0 0 0 0 0] 1;
za  
    za  [0 1 0 0 0 0 0] -0.265;
h  
    h  [0 1 0 0 0 0 0] 0.5;
alpha0  
    alpha0  [0 1 0 0 0 0 0] 0.1;
beta  
    beta  [0 0 0 0 0 0 0] 0.4;
length  
    length  [0 1 0 0 0 0 0] 10;
breadth  
    breadth  [0 1 0 0 0 0 0] 10;
// *************************************************** //
```
The aforementioned dictionaries "gravitationalProperties" and "coefficientsAndConstants" are located in the "Constant" folder. In order for the reader to reproduce the analysis which is presented in this report, the values of the various parameters should be set as follows:

- length 10 m
- breadth 10 m
- h 0.5 m
- za -0.265 m
- alpha0 0.1
- beta 0.4
- time step 0.01
- dUadtCoefficient 1

3.3 Running the code

In order to start the simulation of the problem, the following steps are required:

- Run the blockMesh utility to construct the mesh.
- Specify the desired values for alpha0 and beta of the gaussian hump in the "coefficientsAndConstants" dictionary.
- In controlDict set: startTime 0, endTime 1, deltaT 1, writeInterval 1.
- Run the setGaussHump utility which writes the folder "1" with the updated eta field.
- Delete the folder "0" and rename the folder "1" to "0". Also delete the "uniform" sub-folder in this folder.
- Reset the controlDict parameters to the desired values.
- Run the tutorial.
Chapter 4

Post-processing and Results

4.1 Post-processing

In order to have a better figure of the simulated process of wave evolution over time, we can produce a series of wavy plots of the surface elevation over space. Moreover, contour plots are necessary for the comparisons with already published data which are in contour form.

First, the field "eta" should be loaded in paraView.

Wavy surface

- Go to the field selection toolbar and select eta
- Go to filters > recent > Slice
  - In the slice properties tab, select "Z Normal" and unselect the "show plane" option, then click "Apply".
- Go to Filters > Recent > Warp By Scalar.
  - In the warp properties tab, set the " Scalars" to "eta", the "Scale Factor" to 20, the "Coloring" to "eta", then click "Apply".

Contour plots

- Go to the field selection toolbar and select eta.  
- Go to the Calculator and calculate the scalar field $\text{Result} = \text{eta}/0.1$.  
- Go to Filters > Recent > Slice.  
  - In the "Scalars" select the "Result", in the "Coloring" select "Solid Color", then click "Apply".  
- Go to Filters > Recent > Contour  
  - In the contour properties tab, in the "Contour By" select "Result", click to select the "Compute Scalars" option, then in the window "Isosurfaces" add the desired contours (in this report are the 0.01, 0.1 and 0.2), then click "Apply".  
  - Go again in the contour properties tab, under "Coloring" select the "Solid Color", under the "Styling" adjust the "Line Width" as you want. Here the line width is set to 3mm. Press "Enter" and the contour plot is updated.
4.2 Results

The evolution of the free surface elevation in the wave basin at initial condition as well as after 10, 20, 30, 40 and 50 seconds is presented in the figures 4.1 and 4.2.

Moreover, the contour plots of $0.01H_0$, $0.1H_0$ and $0.2H_0$ were plotted in order to check their pattern with the given ones from the FUNWAVE software. The contours, which have very similar shape with the FUNWAVE ones, are presented in the figures 4.3 to 4.7.

Another way to test the validity of the models is to compare the midpoint elevation time histories obtained by the NwoguFoamRK and the AbbottRK solvers. Moreover, comparison with the analytic solution of the problem and the solution of the linearized Nwogu equations can be done and it takes place in the figures 4.8, 4.9 and 4.10.
4.2. RESULTS

CHAPTER 4. POST-PROCESSING AND RESULTS

Figure 4.2: Surface elevation at different times

(a) 40 sec

(b) 50 sec

Figure 4.3: Contour plots at 10 and 20 seconds

(a) 10 sec, OpenFOAM

(b) 10 sec, FUNWAVE, Wei and Kirby [1]

(c) 20 sec, OpenFOAM

(d) 20 sec, FUNWAVE, Wei and Kirby [1]
4.2. RESULTS

CHAPTER 4. POST-PROCESSING AND RESULTS

Figure 4.4: Contour plots at 30 seconds

(a) 30 sec, OpenFOAM
(b) 30 sec, FUNWAVE, Wei and Kirby [1]

Figure 4.5: Contour plots at 40 seconds with element size 10x10 cm

(a) 40 sec, OpenFOAM
(b) 40 sec, FUNWAVE, Wei and Kirby [1]
4.2. RESULTS

CHAPTER 4. POST-PROCESSING AND RESULTS

Figure 4.6: Contour plots at 40 seconds with element size 5x5 cm

(a) 40 sec, OpenFOAM
(b) 40 sec, FUNWAVE, Wei and Kirby [1]

Figure 4.7: Contour plots at 50 seconds with element size 10x10 and 5x5 cm

(a) 50 sec, OpenFOAM denser mesh
(b) 50 sec, FUNWAVE, Wei and Kirby [1]
(c) 50 sec, OpenFOAM denser mesh
(d) 50 sec, FUNWAVE, Wei and Kirby [1]
4.2. RESULTS

CHAPTER 4. POST-PROCESSING AND RESULTS

Figure 4.8: midpoint elevation, OpenFOAM, non linear terms are included

Figure 4.9: midpoint elevation, OpenFOAM, linearized

Figure 4.10: midpoint elevation, FUNWAVE linearized Nwogu(dashed line), standard Bouss.(dotted) and analytical (solid), source: Wei and Kirby [1]
4.3 Discussion and Conclusions

- In figures 4.3 and 4.4 a very good agreement between the results obtained by the Nwogu OpenFOAM implementation and the FUNWAVE software [1],[2] can be pointed out until 30 seconds.

  NOTE: In the figures of the references [1] and [2], the rectangular tank is presented to be of different extent (20x20m). Despite that, in the article, a 10x10 m wave tank is described and for that reason in this project the actual description of the problem was used (i.e the 10x10 m case).

- In the figures 4.5 and 4.7 (a),(b) a larger difference can be observed. This is mainly because in FUNWAVE, a 4th order central difference scheme is used for spatial discretization while in OpenFOAM 2nd order accurate FV scheme is used. This difference can be eliminated by using a denser mesh which is actually the case as it is shown in figures 4.6 and 4.7 (c),(d).

- In figure 4.8, the midpoint elevation time series gives very good (almost perfect) correspondence between the solvers of Nwogu and Abbott equations. In addition, a very good match appears also with the FUNWAVE plots of figure 4.10 with some differences to occur because of the negligence of the nonlinear terms in the latter case.

- The Abbott and Nwogu OpenFOAM solvers were also run after having neglected the non linear terms $\nabla(u_{\eta})$ and $\nabla(u_{u}u_{u})$ and the results are shown in figure 4.9. As it can be seen, the overall time series profile for both solvers match better the FUNWAVE ones. More specifically, the Nwogu OF solution matches the analytical and Nwogu solutions better (see figure 4.10, dashed and solid lines), while the Abbott OF solution matches the standard Boussinesq model curve (see figure 4.10, dotted line) better.
Study Questions

How to use it

- What kind of water wave problems are modeled by the solvers developed in this project?

The theory of it

- What are the key assumptions behind the Boussinesq approximation?
- What the variable $u_a$ stands for in the Nwogu equations?

How it is implemented

- How the term $\nabla(\nabla U_{at})$ is modelled?
- What is the role of CalculateRHSU and CalculateRHSeta in NwoguFoam solver?

How to modify it

- How the shape of the gaussian hump can be modified?
Bibliography


Appendix A - Full code

The solver NwoguFoamRK.C

#include "fvCFD.H"
#include "OFstream.H"

// --- Functions to calculate the time derivatives of et and Ua

volVectorField CalculateRHSU( volVectorField Uat, volScalarField eta, volVectorField Ua, dimensionedScalar g, dimensionedScalar h, dimensionedScalar za, dimensionedScalar timeDerCoef )
{
    dimensionedScalar C("C", dimensionSet(0,0,0,0,0,0,0), 1);

    solve( fvm::Sp(C,Uat) + (sqr(za)/2 + za*h)*fvm::laplacian(Uat) == - 0.5*fvc::grad(Ua & Ua)
        - g*fvc::grad(eta)
        - (sqr(za)/2 + za*h)*fvc::grad(fvc::div(Uat))
        + (sqr(za)/2 + za*h)*fvc::laplacian(Uat) );

    volVectorField dUadt("dUadt", Uat);
    return dUadt;
}

volScalarField CalculateRHSeta( volScalarField eta, volVectorField Ua, dimensionedScalar za, dimensionedScalar h, dimensionedScalar timeDerCoef )
{
    dimensionedScalar C1("C1", dimensionSet(0,0,-1,0,0,0,0), 0);
    volScalarField etat("etat", C1*eta);

    solve
    ( fvm::Sp(timeDerCoef,etat) == - fvc::div( (h + eta)*Ua
        + h*(sqr(za)/2 - sqr(h)/6)*fvc::grad(fvc::div(Ua))
        + h*(za + h/2)*fvc::grad(h*fvc::div(Ua)) ) );

    return etat;
```cpp
int main(int argc, char *argv[]) {
    #include "setRootCase.H"
    #include "createTime.H"
    #include "createMesh.H"
    #include "readGravitationalAcceleration.H"
    #include "readCoefficientsAndConstants.H"
    #include "createFields.H"

    // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    int countRK;

    float rk4a[5] = { 0.0, -567301805773.0/1357537059087.0, -2404267990393.0/2016746695238.0, -3550918686646.0/2091501179385.0, -1275806237668.0/842570457699.0};

    float rk4b[5] = { 1432997174477.0/9575080441755.0, 5161836677717.0/13612068292357.0, 1720146321549.0/20902068949498.0, 3134564353537.0/4481467310338.0, 2277821191437.0/14882151754819.0};

    volVectorField UaRes("UaRes", Ua*0);
    volScalarField etaRes("etaRes", eta*0);

    Info<< "\n Starting time loop\n" << endl;

    fileName outputFile("Elevation.txt");
    OFstream os(runTime.path()/outputFile);
    os << "time series of midpoint\n";
    os << endl;

    while (runTime.loop()) {
        Info<< "\n Time = " << runTime.timeName() << nl << endl;
        dimensionedScalar dt = runTime.deltaT();
        UaRes = UaRes*0;
        etaRes = etaRes*0;

        // 3rd order Runge Kutta loop..../

        countRK = 0;
    }
}
```
for (countRK=0; countRK<5; countRK++)
{
    UaRes = rk4a[countRK]*UaRes;
    etaRes = rk4a[countRK]*etaRes;

    UaRes = UaRes + dt*CalculateRHSU( Uat, eta, Ua, g, h, za, timeDerCoef);
    etaRes = etaRes + dt*CalculateRHSeta( eta, Ua, za, h, timeDerCoef );

    Ua = Ua + rk4b[countRK]*UaRes;
    eta = eta + rk4b[countRK]*etaRes;

    eta.correctBoundaryConditions();
    Ua.correctBoundaryConditions();
}

int midPoint = mesh.findCell(point(5.0,5.0,0.05));

double elevation = eta[midPoint];

    //ofstream os(runTime.path()/outputFile);
    os << elevation;
    os << nl << endl;

    runTime.write();

        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
             << " ClockTime = " << runTime.elapsedClockTime() << " s"
             << nl << endl;
    }

    Info<< "End\n" << endl;

    return 0;
}
The AbbottRK.C file

The AbbottRK solver is implemented in the same way as the aforementioned NwoguFoamRK and differs only in the functions CalculateRHSU and CalculateRHSeta. Specifically these functions are implemented as follows:

```cpp
volVectorField CalculateRHSU( volVectorField Uat, volScalarField eta, volVectorField Ua, dimensionedScalar g, dimensionedScalar h, dimensionedScalar za, dimensionedScalar timeDerCoef )
{
    dimensionedScalar C("C", dimensionSet(0,0,0,0,0,0,0), 1);
    scalar B = 1/15;

    solve( fvm::Sp(timeDerCoef,Uat) - (B+(1.0/3.0))*sqr(h)*fvm::laplacian(Uat) == -0.5*fvc::grad(Ua & Ua) + (B+(1.0/3.0))*sqr(h)*fvc::grad( fvc::div(Uat) ) - (B+(1.0/3.0))*sqr(h)*fvc::laplacian(Uat) + B*g*sqr(h)*fvc::grad( fvc::laplacian(eta) )
    );

    volVectorField dUadt ("dUadt", Uat);

    return dUadt;
}

volScalarField CalculateRHSeta( volScalarField eta, volVectorField Ua, dimensionedScalar za, dimensionedScalar h, dimensionedScalar timeDerCoef )
{
    dimensionedScalar C1("C1", dimensionSet(0,0,-1,0,0,0,0), 0);
    volScalarField etat ("etat", C1*eta );

    solve( fvm::Sp(timeDerCoef,etat) == - fvc::div( (eta+h)*Ua) );

    return etat;
}
```
The createFields.H file

Info<< "Reading transportProperties\n" << endl;

Info<< "Reading field eta\n" << endl;
volScalarField eta
(
    IObject
    ("eta",
        runTime.timeName(),
        mesh,
        IObject::MUST_READ,
        IObject::AUTO_WRITE
    ),
    mesh
);

Info<< "Reading field Ua\n" << endl;
volVectorField Ua
(
    IObject
    ("Ua",
        runTime.timeName(),
        mesh,
        IObject::MUST_READ,
        IObject::AUTO_WRITE
    ),
    mesh
);

Info<< "Reading field Uat\n" << endl;
volVectorField Uat
(
    IObject
    ("Uat",
        runTime.timeName(),
        mesh,
        IObject::MUST_READ,
        IObject::AUTO_WRITE
    ),
    mesh
);
Auxiliary header files

The readCoefficientsAndConstants.H file

```
Info<< "\nReading coefficientsAndConstants" << endl;

IOdictionary coefficientsAndConstants
{
    IObect
    (
        "coefficientsAndConstants",
        runTime.constant(),
        mesh,
        IObect::MUST_READ_IF_MODIFIED,
        IObect::NO_WRITE
    );
}

const dimensionedScalar za(coefficientsAndConstants.lookup("za"));
const dimensionedScalar h(coefficientsAndConstants.lookup("h"));
const dimensionedScalar timeDerCoef(coefficientsAndConstants.lookup("dUadtCoefficient"));
const dimensionedScalar alpha0(coefficientsAndConstants.lookup("alpha0"));
const dimensionedScalar beta(coefficientsAndConstants.lookup("beta"));
```

readGravitationalAcceleration.H

```
Info<< "\nReading gravitationalProperties" << endl;

IOdictionary gravitationalProperties
{
    IObect
    (
        "gravitationalProperties",
        runTime.constant(),
        mesh,
        IObect::MUST_READ_IF_MODIFIED,
        IObect::NO_WRITE
    );
}

const dimensionedScalar g(gravitationalProperties.lookup("g"));
```
The setGaussHump.C utility

#include "fvCFD.H"

int main(int argc, char *argv[]) 
{
    #include "setRootCase.H"
    #include "createTime.H"
    #include "createMesh.H"
    #include "createFields.H"
    #include "readCoefficientsAndConstants.H"

    while (runTime.loop())
    {

        forAll(eta, cellID)
        {
            double x = mesh.C()[cellID].component(0);
            double y = mesh.C()[cellID].component(1);

            eta[cellID] = alpha0.value()*Foam::exp(-beta.value()*(sqr(x-0.5*length.value())
            +sqr(y-0.5*breadth.value())));
        }

        runTime.write();
    }

    return 0;
}