Temporal and spectral analysis of simulated and experimental Boussinesq type waves

by

Jordan R. Scarrott

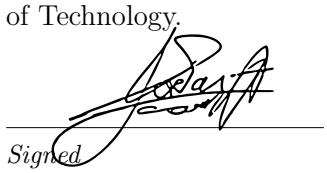
Thesis presented in fulfilment of the requirements for the degree of Masters in Engineering: Electrical Engineering in the Faculty of Engineering at the Cape Peninsula University of Technology

Supervisor: Dr Kessie Govender

June 2024

Declaration

I, Jordan Ross Scarrott, declare that the contents of this thesis represent my own unaided work, and that the thesis has not previously been submitted for academic examination towards any qualification. Furthermore, it represents my own opinions and not necessarily those of the Cape Peninsula University of Technology.



Signed

Date: November 7, 2024

Abstract

In this project we are interested in using computers to predict the characteristics of ocean waves where they meet the shore. Waves in the ocean play an important role in a number of areas as follows: Out in the deep sea they impact shipping and fishing. The breaking of waves along the coast results in erosion and beach line changes. The breaking of waves also creates huge forces on shoreline structures and can be very destructive. Thus a knowledge of these waves and their characteristics is useful for beach management and protecting coastal structures and harbours.

In order for numerical models to be valid, they must be comparable to real world experimental equivalents. This is particularly true for complex phenomena like Boussinesq beach waves. The Boussinesq equation is highly nonlinear and is further complicated by the boundary conditions that need to be satisfied. In this work we aim to numerically solve the equations for water waves propagating along a narrow and long tank in which a sloping bottom is introduced at one end. The purpose of the sloping bottom is to create/simulate breaking waves. By doing so we aim to determine the domain of validity for the chosen numerical scheme.

As a precursor for solving the Boussinesq equation, we first attempt to numerically solve the classical one dimensional wave equation, followed by the numerical solution of the Korteweg-de Vries (KdV) equation. In solving these two partial differential equations we set the scene for the more complex Boussinesq equation. In solving each of these equations we set up the discretisation procedure, followed by actual implementation of the discretised equation using MATLAB.

In the numerical solution of the one dimensional wave equation and the KdV equation, the accuracy of the numerical techniques is assessed by comparing them to analytical solutions or results published in the open literature. In the case of the Boussinesq equation, the accuracy of the results are assessed by comparing them to a real experiment conducted in a wave tank. In both the numerical and experimental cases, we examine the changes in wave profile, wave speed, and spectral content of the waves as they move

from water of constant depth into a region containing a sloping beach where breaking occurred.

Acknowledgements

Firstly I would like to thank my supervisor, Dr Kessie Govender, for his years of mentorship not only during my masters degree but since my first day at the CPUT campus. He is the reason I have wanted to do my masters degree since my first year of university and it has been a privilege to have been under his supervision. All of his students admire his keen insight and life long curiosity and I can only hope these qualities to be contagious. I would like to thank my amazing wife, Mieke, for her unwavering support as we both worked through our respective masters degrees while working full time during our first year of marriage. I am so proud of everything we have achieved together and it has only brought us closer. I would like to thank all of my family and friends who have inevitably suffered the stress of this degree together with us. I would like to thank the French South African Institute of Technology (FSATI) for their financial support during the first two years of my masters degree.

Contents

List of Figures

List of Algorithms

Acronyms

ABPC Adams-Bashfourth Predictor Corrector

ABMPC Adams-Bashfourth-Moulton Predictor Corrector

AI Artificial Intelligence

CFD Computational Fluid Dynamics

CFL Courant Friedrichs Lewy

DNS Direct Numerical Simulation

FDM Finite Difference Method

FEM Finite Element Method

FFT Fast Fourier Transform

iFFT Inverse Fast Fourier Transform

KdV Korteweg-De Vries

ML Machine Learning

PDE Partial Differential Equation

SPH Smoothed Particle Hydrodynamics

WoS Walk on Spheres

WoSt Walk on Stars

Chapter 1

Introduction

1.1 Background information

In the modern world computers are an integral part of our lives. Computers are used for sending and receiving email messages, drafting documents, accessing the internet etc. Even our smart cell phones are computers. Business and banking institutes rely on computers for securely storing and processing personal and financial information. Recent advancements in everything from self driving autonomous vehicles, drones, and farming machinery have greatly benefitted from advancements in computing technology. Computing has been used to enhance the capabilities of existing technology in every area of our lives from real-time weather data applications on our phones to the precision controls of aeroplanes that are informed by physical weather models. Even the oceans are modelled and used to optimise the routes taken by freighter ships.

During the COVID-19 pandemic, for example, computers were extensively used to predict the propagation and spreading of the coronavirus. Computers have been used in modelling and predicting weather phenomena for many years. In this project we are interested in applying computers to solve wave propagation in the ocean. Waves in the ocean play an important role in a number of areas as follows: Out in the deep sea they directly impact shipping and fishing. Along the coast, the breaking of waves results in erosion and beach line changes. Further, the breaking of waves creates huge forces on coastal structures. Thus a knowledge of these waves and their characteristics are useful for beach management and protecting coastal structures and harbours.

There are many advantages to using computers to model waves propagating up a sloping beach, as opposed to conducting laboratory experiments.

Laboratory experiments involving beach waves require large experimental apparatus which can be time consuming to build and maintain and can also only model a small set of shoreline conditions. Numerical models allow the study of waves that are described by very different sets of parameters, giving repeatable and reproducible results. This gives the researcher a high degree of control over what is naturally a very chaotic phenomenon and can allow the investigation of wave phenomena on very different scales, from small scale surface waves to large ocean and coastal scenarios. This gives numerical methods a naturally superior range of applicability that is simply not possible in physical experiments.

Wave propagation is usually described mathematically using partial differential equations (PDEs). In this thesis I will focus mainly on the following PDEs:

1. The standard 1D wave equation which is described by:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}, \tag{1.1}$$

   where u represents the displacement of some quantity and x and t are the spatial and time variables respectively. This equation describes waves such as those on a string or vibrations along a metal rod.

2. The Korteweg-de Vries equation (KDV) which is described by:

$$\frac{\partial u}{\partial t} + \eta u \frac{\partial u}{\partial x} + \mu^2 \frac{\partial^3 u}{\partial x^3} = 0, \tag{1.2}$$

   where u is the height of the wave, x is a spatial dimension, t is time, and η and μ are two real constants. This equation describes waves such as those in a plasma and also in shallow water.

3. The Boussinesq equation which is described by:

$$\eta_t + \nabla\cdot(h+\eta)\,\mathbf{u} + \nabla\cdot\left\{\left(\frac{z_\alpha}{2} - \frac{h^2}{6}\right) h\nabla\left(\nabla\cdot\mathbf{u}\right) + \left(z_\alpha + \frac{h}{2}\right) h\nabla\left[\nabla\cdot(h\mathbf{u})\right]\right\} = 0, \tag{1.3}$$

$$u_t + g\nabla\eta + (\mathbf{u}\cdot\nabla)\,\mathbf{u} + z_\alpha\left\{\frac{z_\alpha}{2}\nabla\left(\nabla\cdot u_t\right) \;+\; \nabla\left[\nabla\cdot(h\mathbf{u}_t)\right]\right\} = 0, \tag{1.4}$$

   where η is the water surface elevation, $\mathbf{u} = (u, v)$ is the horizontal surface velocity at arbitrary depth z_α, h is water depth, and g is the

gravitational acceleration. The subscript t refers to time derivatives of the respective quantity. This equation describes, for example, waves propagating in shallow water and it is the equation which will be our primary focus.

The emphasis in this thesis will be the numerical solution of the above equations, as opposed to their derivation or analytical solutions to these.

1.2 Aims and Objectives

The aim of this project is to simulate the dynamics of classical fluids. In particular, we will examine water waves propagating in a narrow long tank and approaching a sloping beach. In order to achieve this aim the following objectives need to be completed:

1. Solve the 1D wave equation numerically as an introduction to numerical techniques.

2. Solve the Korteweg-De Vries equation numerically for shallow water waves.

3. Solve the Boussinesq equation for shallow water waves.

4. Compare results of Boussinesq simulation with existing experimental data.

The following research design and methodology was adopted to achieve the above mentioned aims and objectives:

1. Undertake a thorough study of how to use MATLAB to write programs to solve PDEs numerically.

2. Review and implement current techniques used for solving different types of PDEs numerically (compare Direct Numerical Simulation (DNS), Smoothed Particle Hydrodynamics (SPH) and Machine Learning (ML) techniques).

3. Interpret and understand the theory behind each equation that is simulated.

1.3 Outline of the thesis

The thesis is organised as follows:

In Chapter 2 an overview of various numerical techniques are provided. These include: Finite Difference Methods (FDM), Finite Element Methods (FEM), Smoothed Particle Hydrodynamics (SPH), AI based methods, and Monte-Carlo methods.

In Chapter 3 I expand on the numerical analysis presented in Chapter 2. Specifically the numerical solution of the standard 1D wave equation is discretised and solved numerically. Further in this chapter I provide some methods of assessing the stability of the numerical solutions.

In Chapter 4 the numerical solution of the Korteweg de Vries equation is examined and its interaction with a standard cyclic boundary condition is investigated. We then discuss methods for tracking the technique's stability using the laws of conservation of momentum and conservation of energy and we comment on the unique interactions of KdV waves with each other.

In Chapter 5 I describe the discretisation of the Boussinesq equation using a potential flow model. We then solve these equations numerically using a predictor-corrector method. The derivation of these equations, and specifically the conversion to and from velocity potentials is discussed in detail.

Before examining the numerical solution of the Boussinesq equation, I digress in Chapter 6 to examine data from a real experiment conducted by my supervisor and his PhD student (Mukaro, Govender and Mccreadie, 2013). The experiment involved waves propagating in a long tank and moving up a sloping beach where breaking occurred. I examined the changes in the wave profile, wave speed and spectral content of the waves as they move up the beach. This analysis will be used as a base line in which to examine the numerical solution of the Boussinesq equation.

In Chapter 7 I numerically simulate the experiment discussed in Chapter 6. The analysis of the numerical results focuses on the same aspects of the waves as the experimental analysis including changes in wave profiles, wave speed, and spectral content of the waves. Comparisons are then drawn between the two sets of results. The algorithms used for these analyses are included and discussed in the text. Further, the simulation allows us to generate and analyse data at points along the beach that were outside of the range of the experimental setup.

Note that each chapter contains detailed citation of references, which represents the literature that was studied.

Chapter 2

Numerical Methods

2.1 Introduction

In this chapter we will investigate various approaches to solving partial differential equations numerically. We will define the types of problems we are aiming to solve and choose an appropriate class of techniques which we will then use to solve an example equation. Doing so will allow us to go into more detail about the specific complexities of numerical simulations and the common features they share.

When replicating a physical system in a simulation we are naturally most concerned about real-world accuracy. Does the simulation accurately and reliably predict phenomena that we see in the physical world? If it does, a simulation then conforms to all the same principles of a classical experiment, as long as it takes into account all of the effects it is trying to model. However, the simulation space contains limitations that we do not have in the real world. For many cases, the real world can be considered to consist of continuous quantities. However when we simulate it, we are forced to compute discrete quantities. The process of solving equations at specific intervals in time and space is called discretisation.

The problems we are trying to solve with numerical computing are often too complex to be solved analytically, and can in many cases be solved through experimentation. However, real world experiments are costly and time consuming to run. The problems of ocean waves specifically are good candidates for numerical simulation because we can solve these equations with a great degree of control over the experiment itself.

Before going into detail about the equations that will form the subject of this thesis, let us briefly review some modern numerical simulation techniques.

2.2 Overview of numerical simulation techniques

2.2.1 Finite difference methods

Finite difference methods take the simplest and perhaps most obvious approach to solving differential equations. The simulation space is divided into a grid of points at which computations can take place and the number of these points is determined by a spatial discretisation step size and the total size of the domain. In its simplest form, this grid is regularly spaced in all directions, and does not change with time. The concept of breaking a space down into finite differences was described by Brook Taylor in the 1700's, and his work on the Taylor series is the place from which many modern finite difference techniques are derived. It was used to solve the first Boussinesq equations for water waves on a beach, and newer techniques can solve coupled Schrodinger-Boussinesq equations using the same principles (Deng and Wu, 2021; Peregrine, 1967). This shows that while it is the oldest of the numerical methods, it is used extensively in modern literature.

One of the most obvious inefficiencies of this method is that computation is spread evenly throughout the simulation space. If more detailed computation is required in a specific region, the entire space is forced to be computed at a higher resolution. To combat this dilemma many techniques have been devised to subdivide regions that require more details into grids of higher resolution as shown in Figure 2.1. While this type of spatial optimization is more typical of finite element methods, it is heavily researched and applicable to all types of numerical simulations (Catmull and Clark, 1978; Chong, 1978; Li, Wei and Zhang, 2019; Sederberg, Zheng, Sewell and Sabin, 1999).

2.2.2 Finite element methods

In finite difference methods the simulation space is populated by a finite number of computation points. These points are distributed throughout the space with areas of high complexity having a correspondingly higher density of these points. In this way, finite difference methods aim to optimize the number of computation points needed to successfully approximate a solution. As the simulation evolves, the distribution of these points can be adjusted in a process called remeshing. Generally a greater density of computation points is assigned to areas with larger spatial or temporal derivatives. Much effort has been dedicated to this in recent years to make highly efficient, highly scalable numerical techniques (Ando, Thurey and Wojtan, 2013; Xiao

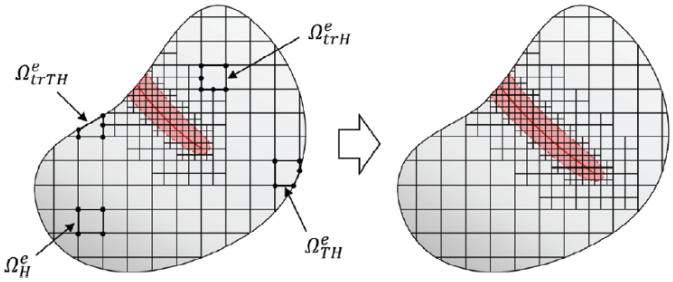


Figure 2.1: Work by Ho-Young Kim and Hyun-Gyu Kim showing a remeshing technique called adaptive trimmed hexahedral (TH) mesh refinement where a higher mesh density is achieved surrounding the location of a crack in a material by subdividing the parts of the grid into smaller regions (Kim and Kim, 2021).

et al., 2020). This idea can be seen in Figure 2.1 where meshes are fitted to the specific geometry of the problem being solved so as to optimize the computation of the solution. One modern technique skips the issue of mesh generation entirely with a novel Monte Carlo based geometry processing technique (Sawhney and Crane, 2020). This technique will be discussed in Section 2.2.4.

2.2.3 Smoothed Particle Hydrodynamics (SPH)

Smoothed Particle Hydrodynamics is a particle based numerical technique that developed out of astrophysics in the 1990s. The first versions of the technique suffered from weak convergence characteristics, but have since matured to have fourth order convergence accuracy (Lind and Stansby, 2016). SPH approaches are typically more complex to implement and more computationally demanding than finite difference and finite element methods, but have the advantage of being highly parallelizable and effective on parallel processing units (Lind, Rogers and Stansby, 2020). This allows the number of simulated particles in some studies to exceed one hundred million even back in 2010 (Maruzewski, Le Touzé, Oger and Avellan, 2010). SPH has proven to be a versatile numerical method and has been used to solve problems in many domains including the non-linear Schrödinger equation from Quantum Mechanics (Mocz and Succi, 2015). These techniques have the less-

acknowledged advantage of being somewhat more intuitive to understand and implement. Instead of dealing only with grids of data, this technique lends itself naturally to conventional object oriented software principles by modelling "particles" directly. This can make this type of software more scalable and maintainable for the ones implementing it at scale.

SPH is known as a meshless or mesh-free method. One begins with a number of particles that are analogous to the grid points of mesh based methods - that is, they act as points at which computations can be carried out. These particles have mass, experience forces and are free to move about in the simulation space. Forces are computed between neighbouring particles subject to a smoothing function. The smoothing function defines the strength of the interactions and is often chosen to be a Gaussian function centred on the specified particle as shown in Figure 2.2.

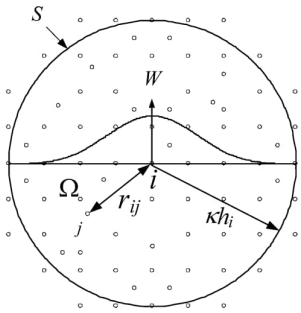


Figure 2.2: Diagram showing two dimensional SPH simulation in a domain Ω with surface S. All field calculations are computed for each particle i within the *support domain* with radius κh_i. These values are averages over all j using the smoothing function S. Here r_{ij} denotes the distance between the particle i and the subject particle in particle group j (Liu and Liu, 2010).

This smoothing function is chosen to taper off to zero at a specific length. This is called the smoothing length and effectively defines a radius around the particle within which to compute neighbouring particle searches and interactions. As an optimization this smoothing length can be variable (Zeng, Wu, Deng, Zhu and Chi, 2021). Mesh-free methods have the intrinsic advantage that computation is limited to areas of the simulation space that actually contain fluid. This completely avoids the problems of remeshing found in Sections 2.2.1 and 2.2.2.

Despite these advantages, SPH remains a relatively complex CFD technique requiring more effort to characterise boundary conditions, achieve reliable convergence rates, and manage spurious pressure wave fluctuations

(Lind et al., 2020).

2.2.4 AI based techniques

The approach taken by the Artificial Intelligence pioneers, Deepmind, attempts to merge two highly complex fields of computer science (Pfaff et al., 2021). The first being general purpose AI, and the second being highly efficient, highly accurate numerical techniques.

In this approach an AI model is trained on real world data then asked to extrapolate a new, unseen set of data. These methods often involve a finite element based spatial discretisation coupled with a learning based temporal extrapolation technique. The unique approach of Pfaff et al. (2021) uses an AI to predict the change in a system by encoding the system's state in a graph network. They call this framework *MESHGRAPHNETS*. Figure 2.3 below shows the results obtained from the same model for four very different physical problems.

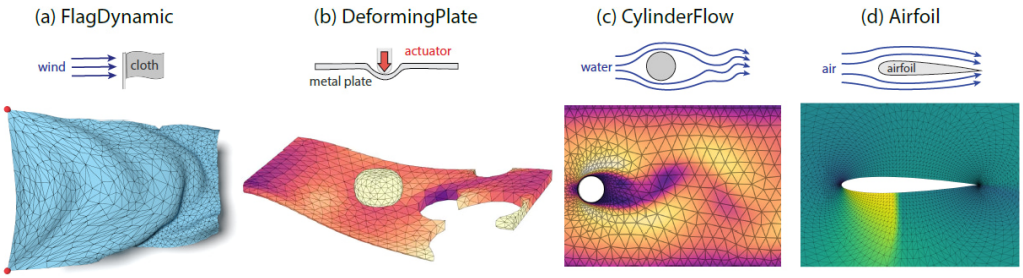


Figure 2.3: State of the art techniques can predict the evolution of vastly different physical systems. The technique by Pfaff et al. (2021) is shown to accurately simulate the interactions between (a) wind and cloth, (b) a metal plate and actuator, (c) turbulent flow of water around a cylinder in 2D, and (d) airflow around an airfoil in 2D.

The AI propagates learned changes through the network and the results are decoded and fed back into the system to compute the next state using Eulerian integration. This method combines finite element and AI based methods and incorporates a remeshing algorithm to optimize the mesh. AI based methods tend to be used primarily for film and computer graphics where visual appeal is valued over physical accuracy. However, *MESHGRAPHNETS* has the ability to accurately predict the evolution of a wide range of physical systems including aerodynamics, fluid dynamics, cloth, and structural me-

chanics while outperforming other particle and grid based methods. AI based simulations have reached a point where they can be unconditionally stable, and in some cases orders of magnitude more efficient than direct numerical simulations (Holden et al., 2019; Sanchez-Gonzalez et al., 2020).

2.2.5 Monte Carlo methods

As mentioned previously, one novel technique takes a Monte-Carlo approach to solving partial differential equations. A paper by Sawhney and Krane (2020) relies on a technique called Monte-Carlo integration which states that the integral of a function f can be estimated simply by randomly sampling the domain. The integral will then be the average value of the samples weighted by the probability distribution from which those samples are drawn.

For an arbitrary probability distribution we have

$$F_N = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)}, \tag{2.1}$$

where f is the function to be integrated and F_N is the approximation of its integral, N is the number of samples, p is the probability distribution function of the random numbers, and X is a random sample. The probability distribution used can be chosen so that samples are more frequently chosen at "important"[1] locations along the function. Monte-Carlo based methods avoid the use of meshes entirely and are therefore known as *mesh-free*. Equation 2.1 forms the foundation of Monte-Carlo based techniques and is unique amongst other numerical techniques in that it allows one to introduce many tools from the field of statistics into its simulations.

Monte-Carlo techniques have been largely unexplored in the last half-century as a tool for solving PDEs. Recently, however, there is a renewed interest in these techniques as described in an exciting paper by Sawhney and Krane (2020). Their paper develops ideas that were first proposed in 1956 in a paper titled *Some Continuous Monte Carlo Methods for the Dirichlet Problem* (Muller, 1956). In this paper, Muller derived an algorithm called *Walk on Spheres (WoS)* that combines two statistical principles, known as Kakutani's Principle (Kakutani, 1944) and the Mean Value Property (Axler, Bourdon and Wade, 2013) in order to solve PDEs. With reference to Figure 2.4, Muller showed that the solution to an elliptic PDE at a point x

[1] This idea is known as importance sampling and is analogous to the job of remeshing in other aforementioned numerical techniques in that it optimizes the position of computation points.

within a domain is equal to the average of boundary values reached by recursively choosing a random point on a sphere centred at x_i until reaching the boundary.

With reference to Figure 2.4, we randomly choose a point x_0 within the domain Ω. We then find the largest possible sphere centred at this point that fits within the domain. This circle will be touching the boundary of the domain at least at one point. We then pick a random point x_1 on the surface of this sphere and use this as the centre of our next sphere. This process is repeated until the randomly chosen point x_k is within some minimum distance $\partial\Omega$ from the domain boundary. The quantity simulated (for example the temperature of a material) is then sampled at this boundary point. Using Equation 2.1 the a solution can be computed at the point x_0 by doing a number of random walk on spheres starting at x_0. This process is then repeated any number of times at random places in the domain. This technique is unique in that the entire domain does not need to be considered in order to find a solution at a specific point, meaning that a rough initial solution can be quickly found and can then be progressively refined in successive iterations. This is very useful for simulations running on complex and time consuming geometries.

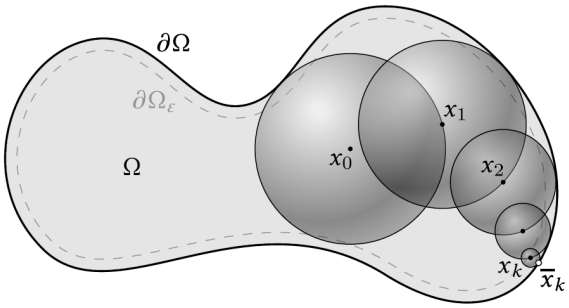


Figure 2.4: The walk on spheres algorithm recursively chooses a random point on the largest sphere as possible at point x_i within the simulation space until it comes within some minimum distance of the boundary.

In a comparison between a FEM technique and their WoS technique on a highly detailed geometry, the authors found that while the FEM took 14 hours and used 30GB of memory to solve the Poisson equation, their approach used 1GB of memory and took less than a minute to compute. While this approach has obvious benefits for many applications, it is however, limited to elliptic equations with Derichlet boundary conditions (Sawhney and Crane, 2020). In 2023 the authors published a paper about an evolution of

the WoS algorithm which they call *Walk on Stars (WoSt)* which can solve elliptic PDE's with Neumann boundary conditions as well as the normal Derichlet boundary conditions (Sawhney, Miller, Gkioulekas and Crane, 2023). It remains however unable to support the solution of the more complex, nonlinear hyperbolic equations that are dealt with as the subject of this thesis.

Something very important to note is that while this technique cannot directly solve fluid equations, it has one application of particular interest to fluid simulations. This is that it is able to perform Helmholtz-Hodge Decomposition which is one of the fundamental theorems of fluid dynamics (Bhatia, Norgard, Pascucci and Bremer, 2013). This offers a new and exciting alternative for this specific part of many existing numerical techniques.

2.3 Summary

In this chapter an overview of the various numerical schemes for solving PDEs have been provided. These include finite difference methods, finite element methods, smooth particle hydrodynamics, Al based techniques, and Monte-Carlo based techniques. The advantages and disadvantages were also discussed.

In the next chapter we will proceed with a worked example. The finite difference technique that will be used in this example will form the foundation of the numerical solution of the Boussinesq equation. We will use the wave equation as a case study to investigate this technique and its various properties and how they can be used to solve more complex equations.

Chapter 3

Numerical analysis of the wave equation

3.1 Introduction

In this chapter I lay out the basic framework for how PDEs are solved numerically. I begin with the wave equation and schemes for discretising it. Thereafter, I show how these techniques can be used to solve more advanced equations. I will also discuss the more practical considerations of the simulations. The handling of boundary and initial conditions, computational molecules/stencils, and techniques of stability analysis is discussed in this chapter.

3.2 Discretising the Wave Equation

Suppose we have a one dimensional rod and we want to examine the vibrations propagating along it or the propagation of surface water waves in a long tank. These waves can be described by the wave equation

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}, \tag{3.1}$$

where u is the height of the wave and c is a measure of the speed of the wave. Here we have x as the spatial dimension and t as the time dimension. Many numerical methods exist for solving for u in this equation with their own pros and cons. The method shown here is known to be one of the most desirable because of its stability, efficiency and simplicity - the explicit methods. With this type of method we attempt to solve for u directly instead of one of its derivatives. This avoids having to use an extrapolation technique to integrate

this derivative to solve for u. This reduces the complexity of the simulation and results in faster and more stable solutions.

At this point it is good to establish a naming convention for our derivations. When solving for the quantity, u, along the surface of a one dimensional bar of length, l, we can define the discretisation as follows. Derivatives of u in the x direction are defined as u_x and u_{xx} for the first and second order derivatives, and similarly for the time derivatives u_t and u_{tt}. We split up the length of the bar into J pieces each of length Δx, where the j^{th} region is $j\Delta x$ distance along the bar as shown in Figure 3.1. We can then solve the equation for any time $k\Delta t$ in the future up to time K. The disturbance/displacement u from its equilibrium position at time t at position j, k would be $u(j,k)$ and its first derivative, $u_x(j,k)$.

The problem now is to rewrite Equation 3.1 in terms of these discrete points. We want to find the value of u at $(x,t) = (j,k+1)$. That is, the height of the wave at this point on the next iteration. Using the Taylor series in Equation 3.2 we can get an expression for $u(j,k+1)$ in terms of $u(j,k)$ and its derivatives:

$$u(j,k+1) = u(j,k) + u_x(j,k)\Delta x + \frac{1}{2}u_{xx}(j,k)\Delta x^2. \tag{3.2}$$

This allows us to solve for the first and second derivatives u_x and u_{xx} as follows. Dividing by Δx and ignoring the u_{xx} term we get the **forward difference** equation:

$$u_x(j,k) \approx \frac{u(j+1,k) - u(j,k)}{\Delta x}. \tag{3.3}$$

Similarly we can derive the **backward difference** equation:

$$u_x(j,k) \approx \frac{u(j,k) - u(j-1,k)}{\Delta x}. \tag{3.4}$$

We can use the Taylor series in a similar way to find the **centred difference** for the second derivatives to be

$$u_{xx}(j,k) \approx \frac{u(j+1,k) - 2u(j,k) + u(j-1,k)}{\Delta x^2}. \tag{3.5}$$

In the same way, one can get a similar expression for the time derivative. Substituting Equation 3.5 and a similar expression for $u_{tt}(j,k)$ into Equation 3.1 we get

$$\frac{u(j,k+1)-2u(j,k)+u(j,k-1)}{\Delta t^2}=c^2\frac{u(j+1,k)-2u(j,k)+u(j-1,k)}{\Delta x^2}. \tag{3.6}$$

Solving for $u(j,k)$ and letting $r=\frac{c\Delta t}{\Delta x}$, we get

$$u(j,k+1)=2u(j,k)\left(1-r^2\right)+r^2\left[u(j+1,k)+u(j-1,k)\right]-u(j,k-1). \tag{3.7}$$

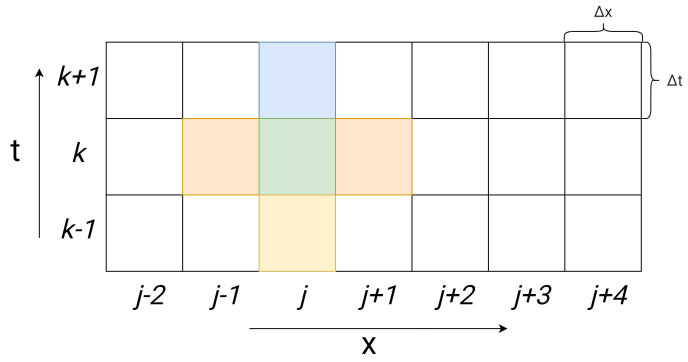


Figure 3.1: An example of a discretised simulation space showing the length of the bar separated into regular intervals in j and computed each iteration k using the computational molecule described in Figure 3.2. The bar is separated lengthwise into segments each of length Δx shown from left to right, and three iterations through time are shown each of length Δt.

Now we have a way to explicitly compute the value of u at any point on a discrete grid. This is the basic method for discretising PDEs and more advanced techniques often still draw from these fundamental concepts. There are many ways to discretise an equation and not all such derivations provide stable results. However, much effort in the scientific community has gone into finding the methods that do and these methods are examined in the relevant sections for each equation.

In this section we have discretised the wave equation and shown a basic scheme for computing the next value of u given any j. We can see that the scheme developed takes into account the current value of u, $u(j,k)$, the value to the left, $u(j-1,k)$, the value to the right, $u(j+1,k)$, and the

previous value of u, $u(j, k-1)$. This is known as a three level scheme and its computational molecule is shown below in Figure 3.2.

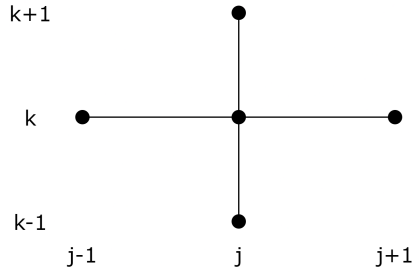


Figure 3.2: The computational molecule for the discretised wave equation in Equation 3.7.

Since the solution of the next time iteration is dependent on the results of the previous two iterations, we need two initial conditions in order to compute the first new iteration. These initial conditions, along with the concept of boundary conditions, will be introduced in Section 3.3.

3.3 Initial and Boundary Conditions

Boundary conditions describe what happens at the edges of our simulation space and, in the real world, describe how the system interacts with its surroundings. Boundaries can simply be the borders of the simulation space or in more complex simulations they can describe the borders of foreign objects within the space. In our simulations they serve two functions. They describe what happens at the borders of the space, and they occupy a gap of cells that the discretisation is not able to fill.

When discretising an equation we often end up with a solution describing $u_{j,k+1}$ in terms of $u_{j,k}$ and its left and right neighbours. On their own, these schemes cannot compute the values at the edges of our simulation space as they depend on values that lie outside of it. This is shown in Figure 3.3. When boundary conditions are added we can solve the equation as shown in Figure 3.4.

Boundary conditions can have various physical meanings. Below I separate them into two main types. Those that are dependent on the state of the system and those that are not.

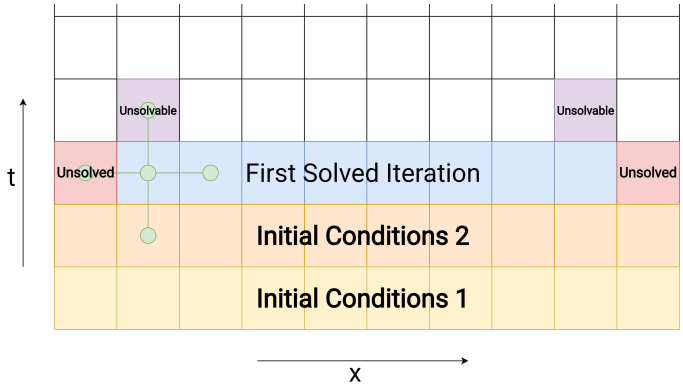


Figure 3.3: Due to the nature of the computational molecule, we cannot simply compute the value of u at the boundaries. This prevents us from solving further cells in the grid.

1. Independent Boundary Conditions

   (a) Function of an external system
   The boundary conditions can change entirely independent of the rest of the system, according to some function $f(x)$. Physically for the wave equation this corresponds to waves being forcefully generated at the boundaries of the system. For example by a wave generator in a wave tank.

   (b) Constant / Fixed
   These boundary conditions keep the same value through the whole duration of the simulation. This can be used to simulate reflection of waves off of walls. Also used as the bottom boundary in Navier-Stokes simulations of wall bounded flow to simulate the No-Slip Boundary condition (Sengupta and Bhaumik, 2019).

2. dependent Boundary Conditions

   (a) Cyclic
   With this boundary condition waves approaching a boundary on one side reappear at the other boundary moving in the opposite direction. Can be thought of as a connection directly from one end to the other. This has been used by Zabusky and Kruskal in their original numerical solution of the Korteweg-De Vries equation (Zabusky and Kruskal, 1965).

(b) Open ended
The boundary maintains the same value as its nearest neighbour. Physically this can simulate water waves slipping up a wall as the waves strike the wall or waves that reach the end of a string that is not attached to anything.

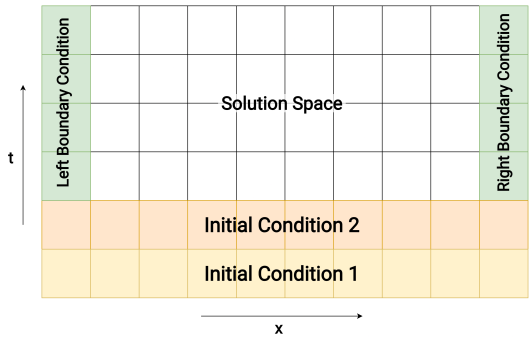


Figure 3.4: The entire simulation space for the 1D wave equation. This includes the first and second initial conditions, the left and right boundary conditions, and the solution space where the explicit numerical scheme will be computed. The first two iterations are explicitly specified.

As shown in Section 3.2, the explicit scheme for solving the wave equation is a three level scheme. To compute $u_{j,k+1}$ we need the data from the previous two iterations in time. This means we need two initial conditions: the state of the system at $t = t_0$ and at $t = t_0 + \Delta t = t_1$. These are shown in Figure 3.3.

In its analytical form, the one dimensional wave equation has two sets of boundary conditions, one for each of its bases, x, and t:

$$u(0, t) = 0, \tag{3.8}$$

and

$$u(L, t) = 0, \tag{3.9}$$

where u is the height of the wave from its resting position, t is time, and L is the distance from the left boundary to the right. These equations state that the disturbance/displacement at the left and right boundaries must always remain at the resting position. For all time t, the displacement is zero at $x = 0$ and $x = L$. These equations are in fact just one set of possible

equations for the boundaries. In the case where the wave is being driven by an external input from the left, the left boundary could be based on some time dependent function, $f(t)$:

$$u(0,t) = f(t). \tag{3.10}$$

In the same way that we are unable to solve the wave equation on the left and right spatial boundaries, we are also limited in the time domain. For this reason we require that the initial state must be defined by

$$u(x,0) = g(x), \tag{3.11}$$

where $g(x)$ is some known function. We then also have the option to define initial conditions for derivatives of u. While it is possible for these to be initialized as being zero, it is common for them to be defined based on some function that relates to u:

$$\frac{\partial u}{\partial t} = h(u,x). \tag{3.12}$$

Equations 3.8, 3.9, and 3.10 translate trivially to their discretised counterparts. For the left and right boundaries in a simulation, u is simply set to zero each iteration. This is normally done directly after the data for the iteration in question has been computed. This ensures that the boundary conditions are prepared for use in the following iteration. The initial conditions for derivatives in Equation 3.12 can similarly be set in this fashion. For Equation 3.11 we simply set the values of the entire first iteration to the value of $g(x)$. This then gives us the information we need to compute subsequent iterations.

3.3.1 Corner points

A common boundary condition for water wave equations is the no-slip condition (Blazek, 2001; Wei and Kirby, 1995). It states that a fluid at a boundary does not have any velocity in the direction along the surface of the boundary, i.e. any velocity perpendicular to the boundary normal is always zero. This is stated generally as

$$\nabla u \cdot \boldsymbol{n} = 0, \tag{3.13}$$

where u is the water surface height and $\boldsymbol{n}$ is the normal to the wall surface. In a 2 dimensional simulation this kind of boundary condition results in two equations being applied to the corner points of the simulation space. One for the x direction and another for the y direction. For the bottom left corner of a 2D space we have

$$u_x(x, 0) = 0, \tag{3.14}$$

because of the bottom boundary condition and

$$u_y(0, y) = 0, \tag{3.15}$$

due to the left hand boundary. For the corner point at $(0, 0)$ we have a velocity vector $v(0, 0)$ described by

$$v(0, 0) = (0, 0). \tag{3.16}$$

Similarly this is applied to all other corner points.

3.4 Stability Analysis

Every discretisation has an accompanying truncation error - a measure of the error that arises from replacing derivatives in an equation with discrete approximations (Logan, 1987). In addition, details below the resolution of our discretisation grid are lost and those that occur on time scales shorter than Δt are also lost. There is therefore a need for strict stability analysis of each numerical technique to determine how well they represent reality. Below, an analysis is conducted to find the *Courant-Friedrichs-Lewy (CFL) stability condition* for the wave equation which determines under which conditions the numerical solutions converge. When finding a CFL condition our aim is to determine under what exact conditions our discretisation is valid for this solution. It can be useful then to solve for some quantity involving our discretisation variables, Δx and Δt.

Beginning with Equation 3.1, we assume a solution to the PDE of the form:

$$u(j, n) = M^n \cos \alpha x(j). \tag{3.17}$$

The above choice of trial function is based on a method developed by John von Neumann for examining stability (Long, 2006*a*). The quantity M above is called the magnification factor and we want this factor to be less than one. This will ensure that the solution does not grow with time. The symbol n refers to the time increments. The goal then is to determine the condition when $|M| < 1$. The $\cos \alpha x$ is just a convenient function for the initial condition which we can control the values of. Substituting the above into Equation 3.7 (our discretised wave equation) we obtain

$$M^2 - \left(2 - 4r^2 \sin^2 \frac{\alpha \Delta x}{2}\right) M + 1 = 0. \tag{3.18}$$

We remind the reader that here, r is a quantity indicating the speed of the waves, scaled by a constant dependent on our discretisation: $r = \frac{c\Delta t}{\Delta x}$. Solving for M we get

$$M = 1 - 2r^2 \sin^2 \frac{\alpha \Delta x}{2} \pm 2r \sin \frac{\alpha \Delta x}{2} \sqrt{r^2 \sin^2 \frac{\alpha \Delta x}{2} - 1}. \tag{3.19}$$

There are two cases to consider: if $r \leq 1$ and $r > 1$. Beginning with the former we can see that the right hand term becomes imaginary when $r^2 \sin^2 \frac{\alpha \Delta x}{2} - 1 \leq 0$. In such a case we get

$$M = 1 - 2r^2 \sin^2 \frac{\alpha \Delta x}{2} \pm i2r \sin \frac{\alpha \Delta x}{2} \sqrt{r^2 \sin^2 \frac{\alpha \Delta x}{2} - 1}. \tag{3.20}$$

Taking the absolute value we can remove i from the equation

$$|M| = \sqrt{\left(1 - 2r^2 \sin^2 \frac{\alpha \Delta x}{2}\right)^2 + \left(4r^2 \sin^2 \frac{\alpha \Delta x}{2} \left(1 - r^2 \sin^2 \frac{\alpha \Delta x}{2}\right)\right)} = 1. \tag{3.21}$$

Here we have one solution for $|M|$, although it is not immediately obvious how much this says about the constraints of our simulation. Reminded of our goal of solving for some quantity containing Δx and Δt we look at the second of the two cases in Equation 3.19.

Considering the case where $r > 1$, we choose α such that $\alpha \Delta x = \pi$. Thus the *sin* terms in Equation 3.19 become 1. Then taking the negative sign in Equation 3.19 gives

$$M = 1 - 2r^2 \sin^2 \frac{\alpha \Delta x}{2} - 2r \sin \frac{\alpha \Delta x}{2} \sqrt{r^2 \sin^2 \frac{\alpha \Delta x}{2} - 1}. \tag{3.22}$$

Thus $|M| > 1$. Consequently solutions of the difference equation remain bounded for all α only in the case described by:

$$r = \frac{c\Delta t}{\Delta x} \leq 1, \tag{3.23}$$

which is the *Courant-Friedrichs-Lewy (CFL) stability condition* for hyperbolic equations which can be written as

$$\frac{\Delta x}{\Delta t} \geq c. \tag{3.24}$$

Physically this CFL condition means that the numerical solution cannot proceed at a slower rate than the speed of the fastest waves in the simulation. This result allows us to choose values for Δx and Δt that produce accurate results.

Here we have produced a reliable mechanism for tracking the convergence of our numerical scheme. This was done through careful mathematical analysis of the expected solution of the equations. For more complex systems, CFL conditions may be notably more difficult to determine, and in such cases other methods of stability analysis are used. In later chapters we will see that we can take advantage of the laws of conservation of mass and momentum, or other predictable parameters, to keep track of the stability of a system and is an important area of research today (Yan, Zheng, Lu and Zhang, 2022).

The results of this simulation of the 1D wave equation are discussed in the next section.

3.5 Numerical Results

Using the explicit scheme shown in Equation 3.7, we can compute the solution to the wave equation along the length of the bar at each time interval. For our initial conditions we begin with two wave peaks travelling in opposite directions. We arbitrarily define the initial condition as the sum of two hyperbolic functions given by $u(x, t_0) = f(x, t_0, b) + \frac{1}{2} f(x, -t_0, 3b)$ where $f(x, t, b) = \cosh^{-2}(x - t - b)$ and x is the axis of propagation of the wave, t is time, and b is a constant controlling a spatial offset for the waves. As can

be seen this gives us two wave peaks with one having an offset three times that of the other. In order to differentiate the two peaks, the second wave was arbitrarily chosen to have half the amplitude of the other. The inverse hyperbolic cosine function (or hyperbolic secant function) is often used in the literature for investigating stability and dispersive properties of solitary wave propagation (Schember, 1982; Wei and Kirby, 1995). After one iteration, a time Δt has passed and the second initial condition has become $u(x, t_1) = f(x, t_1, b) + \frac{1}{2} f(x, -t_1, 3b)$. The boundary conditions are $u(0, t_0) = u(1, t)$ and $u(l, t) = 0$, where l is the length of the bar. The discretisation parameters are $\Delta t = 0.02, \Delta x = 0.025$. We evolve the simulation over 600 iterations through 12 seconds of time, filling in the solution space at $t = (k + 1)\Delta t$ each iteration. Cross sections of the results are plotted in Figure 3.5.

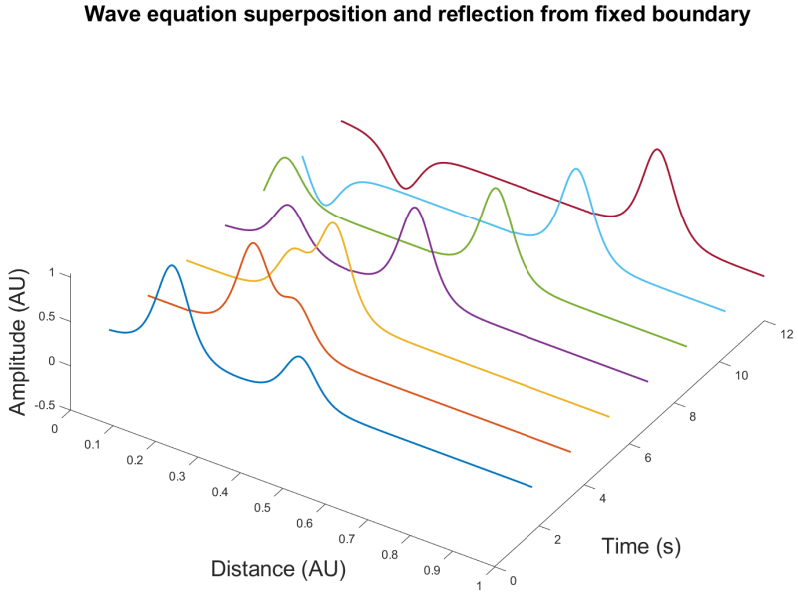
Figure 3.5 shows the interaction of two oppositely moving wave fronts. At time $t = 0s$, the two wave fronts are far enough away from each other so that we can consider them to have no overlap. At $t = 2s$ and $t = 3s$ the two waves are superimposed. The waves have to retain their original amplitudes before and after the interaction. The smaller of the two waves then reaches the fixed left boundary between $t = 8s$ and $t = 10s$ and reflects off it and inverts its amplitude and velocity.

Next, the paths of the two waves can be seen in a heat map shown in Figure 3.6. From here we can clearly see the paths of the waves and the point of their interaction in space-time. The interaction of the two waves does not alter their phase. This is important to note as the types of waves investigated later in this study, do not act in this way. Interactions between KdV waves can result in phase shifts and the state of the waves after an interaction can depend on the relative velocity of the individual waves. Waves in these models can reflect or transmit through each other based on their relative velocities. These will be discussed in more detail in the relevant sections to follow.

3.5.1 Comparison with analytical solution

In order to measure the accuracy of our numerical solution we can compare it against an analytical solution. Our initial conditions describe two wave fronts moving in opposite directions. The large wavefront moving to the right and the smaller wavefront moving left towards the reflective fixed point boundary.

Our initial condition can be described by $u(x, t) = f(x) + g(x)$ where f is the smaller wavefront and g is the larger. We can define the effect of a left boundary condition by adding another term to describe an inverted, oppositely directed wavefront that meets f at the boundary. Our function



$$\begin{array}{c|c} u(x,t_0) = f(x,t_0,b) + \frac{1}{2}f(x,-t_0,3b) & u(0,t) = u(1,t) \\ u(x,t_1) = f(x,t_1,b) = f(x,t_0+dt,b) & u(l,t) = 0 \\ dt = 0.02 & dx = 0.025 \\ b = 3 & \end{array}$$

Figure 3.5: The path of two waves according to the wave discretised wave equation where $f(x,t,b) = \cosh^{-2}(x-t-b)$. In the figure above, u represents the amplitude on the vertical axis and x and t are distance and time, respectively, on the horizontal axes.

then becomes $u(x,t) = f(x) - f(-x) + g(x)$ which can fully describe the reflection for all $0 \leq x < \infty$. Using this function we can directly compute the height of the wave surface at any point in time to use as a comparison for our numerical solution.

In Figure 3.7 the numerical and analytical solutions are plotted over one another. It can be seen visually that the solutions are essentially identical, but for one small area of difference. Figure 3.8 shows a zoomed plot over the region $9 \leq x \leq 13$ for the case of $n = 600$ in Figure 3.7d. In Figure 3.8 we see the only visible differences. In the numerical solution there is a distortion in the trailing edge of the large wavefront. This high frequency oscillation is

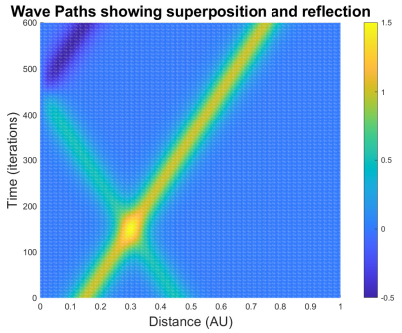


Figure 3.6: The path of two waves according to the discretised wave equation.

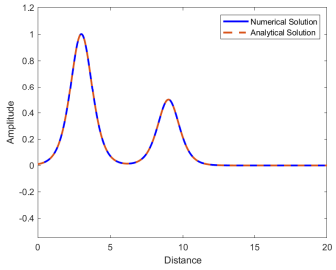
a telltale sign of a common problem encountered in numerical simulation.

This numerical error can be traced back to the setup of the initial conditions of this simulation. At the start of the simulation the left side of the larger peak trailed off towards the left boundary where it approached some number close to but greater than zero. Since the left boundary condition specified the left most point as being zero, this caused a jump from the left most point to its neighbouring point that was large enough to cause a visible oscillation.

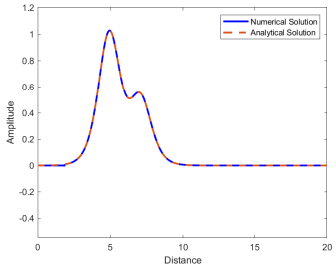
We can draw two useful conclusions from this observation. One, that initial conditions can conflict with boundary conditions. And two, that there is some limit within which the simulation can tolerate these types of inconsistencies. In this case, the oscillation did not seem to grow during the simulation, but also did not diminish visibly and was ultimately insignificant for this this time period. It is however important to acknowledge that unwanted oscillations can appear in a simulation despite having correctly derived a numerical technique as well as its initial and boundary conditions. That alone is not sufficient to guarantee a working simulation. In cases where smaller oscillations are relevant, these types of errors cannot be present.

3.6 CFL Condition Violation

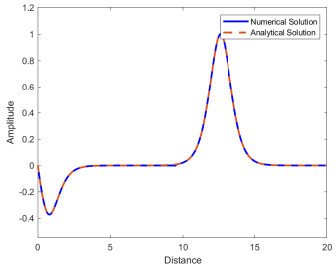
Until now we have seen results from successful simulations. It is useful, however to know what to expect when the simulation is unstable so as to know how to correct for it. What kind of results can we expect when the CFL condition is violated? Let us choose a value of Δt such that the CFL condition is not violated and then run the simulation multiple times, scanning through a range of Δt until it is. To simulate the effect of $\frac{\Delta x}{\Delta t} < c$, we first



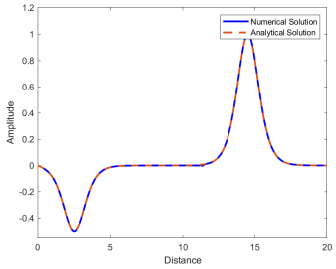
(a) Numerical and analytic wave equation solutions at n = 0 iterations.



(b) Numerical and analytic wave equation solutions at n = 100 iterations.



(c) Numerical and analytic wave equation solutions at n = 500 iterations.



(d) Numerical and analytic wave equation solutions at n = 600 iterations.

Figure 3.7: Comparison of Numerical and Analytic solutions

determine c.

Our initial conditions specify two independent wave fronts moving in opposite directions described by

$$\begin{aligned} u(x,0) &= \cosh^{-2}(x - t_0 - b) + \frac{1}{2}\cosh^{-2}(x + t_0 - 3b) \\ u(x,1) &= \cosh^{-2}(x - t_1 - b) + \frac{1}{2}\cosh^{-2}(x + t_1 - 3b), \end{aligned} \tag{3.25}$$

where $t_1 = t_0 + \Delta t$. Looking at the time dependent components of the wave positions, we can see they move at velocities of

$$\frac{-t_0 - \Delta t + t_0}{\Delta t} = -1,$$

and

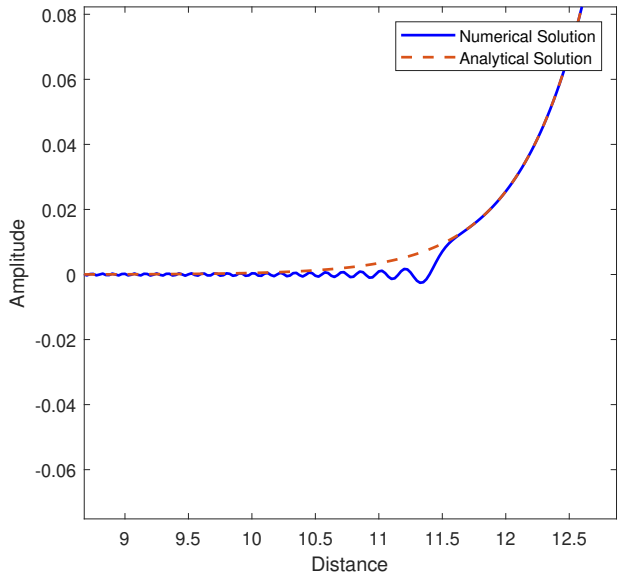


Figure 3.8: Comparison with the analytical solution at n=600 iterations

$$\frac{t_0 + \Delta t - t_0}{\Delta t} = 1,$$

respectively. The CFL condition becomes

$$\frac{\Delta x}{\Delta t} \leq 1.$$

In order to illustrate the effect of violating stability conditions we would like to plot the solution of the wave equation for varying values of Δx and Δt such that $\frac{\Delta x}{\Delta t}$ approaches 1 from above. We can then see how strict the effect of this condition is by proceeding for values that are slightly less than 1. We could alter either of these variables and we choose to alter the value of Δt since altering Δx means we need to recompute the wave equation on grids of different resolutions for each value of Δt. To prevent the need for interpolating between functions of differing resolutions, we simply alter Δt instead.

The results in Figure 3.9 show an exponentially growing high-frequency noise on all functions where the value of Δt violates Equation 3.24. To

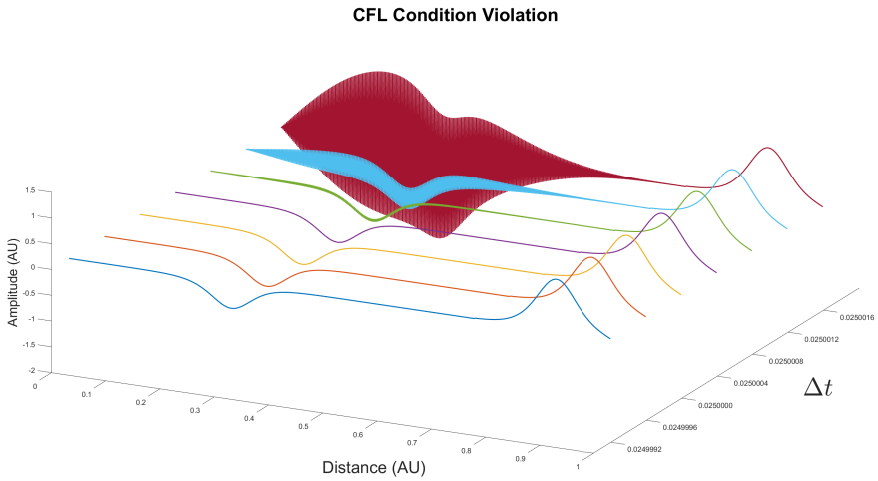


Figure 3.9: A series of plots showing an exponentially growing numerical error on the trough of a wave front. The figure shows how a large numerical error can appear over an extremely small time frame.

generate this amount of unphysical oscillations the CFL only had to be violated by 0.0016%. Let us take a closer look at the error in the results from $\Delta t = 0.0249992 \rightarrow 0.0250008$. The error is computed as $f_{(\Delta t)} - f_{(0.0249992)}$. This is shown in Figure 3.10. We can clearly see that any violation at all of the CFL condition leads to unusable results after a few iterations and also that we can get as close as we need to the CFL condition without these errors appearing. It is, therefore, useful to know that we can safely use space-time resolutions in the limit of the CFL condition as long as we stay enough out of range of the condition that it cannot accidentally be violated by the machine errors[1].

This demonstration makes a valuable point about the importance of being sure about the stability of a numerical solution. Any error in a fluid simulation diffuses throughout the space and can both affect the quality of results and also be difficult to debug.

In many cases, the accuracy of a numerical solution can be checked simply be comparing it to the analytic solution of the equation. In this thesis we are, however, investigating the numerical solutions of equations who have no known analytic solution. This means we need to use other techniques to

[1]Machine errors are errors that arise from rounding associated with the computation of floating point numbers (Kreyszig, 2011).

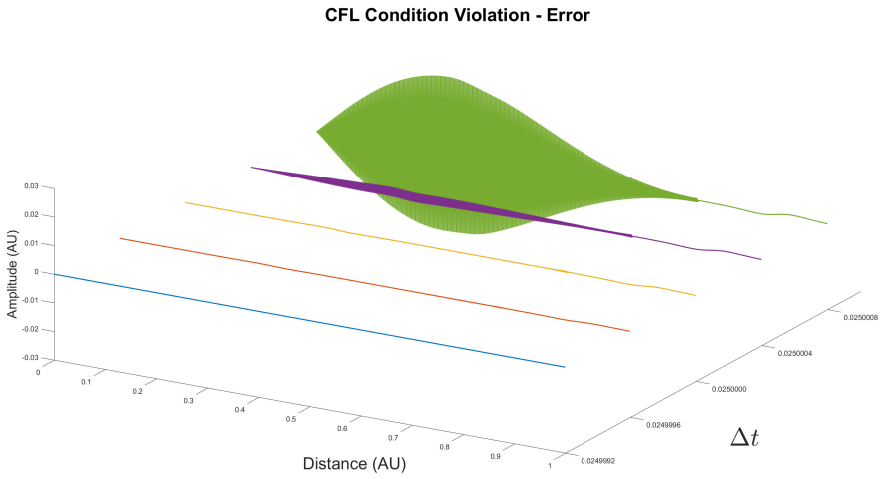


Figure 3.10: CFL Violation Error Zoomed. Zoomed in section of the error around the CFL violation showing significant errors only where $\Delta t > \Delta x$.

check the accuracy and stability of our results. For example we can use the constants of motion as indicators for the stability of a scheme (Bridges and Reich, 2006). Tracking these over time to confirm whether they deviate from a constant value can be a good indication that the solution is accurate.

3.7 Summary

In this chapter we investigated an explicit numerical technique for solving PDEs. We used it to solve the wave equation in one and two dimensions. We were able to confirm the stability of the numerical scheme and use it to predict future states of the system. In the next chapter we will move on to techniques for solving the more complex Korteweg-De Vries equation.

Chapter 4

The KdV Equation

4.1 Introduction

In this chapter we will introduce the Korteweg-de Vries (KdV) equation and its discretisation before comparing its numerical solutions with previous works. We will then investigate methods of analysing the success of our simulations without comparing them to analytic solutions or real world data.

Due to the explosion of research into numerical techniques, the software for solving PDEs has evolved to a point where each component of these algorithms has become its own subfield of computer science. A single fluid simulation can consist of dozens of complex interdependent algorithms. These include mesh generation, remeshing, temporal extrapolation and interpolation, tridiagonal matrix solvers, parallel computing, spectral analysis, boundary condition modelling, linearization of equations, stability analysis and rendering. Each of these parts are also free to develop independently of the others in their own fields. In order to be a good numerical analyst you will need to understand the basic concepts in each of these fields. To be exceptional you must master all of them. It is important therefore to learn the lessons of software engineering and apply the principle of separation of concerns. Programs can be separated into functional layers with very specific purposes. Each layer should be treated as a black box with simple inputs and outputs from those layers. This approach naturally lends itself to object oriented programming if the researcher wishes to remain sane. This also makes debugging much more effective. This is extremely important in a field where mistakes propagate throughout the entire simulation space and become extremely difficult to debug.

One intuitive property of water waves is that they are self dampening. Over time the energy of a wave is dispersed throughout the medium through

which it propagates. As a wave moves through a medium energy is lost and therefore the amplitude of the wave decreases. There is however a type of wave whose amplitude and wavelength is entirely self sustained over large distances (Brauer, 2000). It was first studied mathematically in the mid 19th century by the Scottish engineer, John Scott Russell and is now known as the soliton. This surprising phenomenon has been observed in seemingly unrelated places such as plasmas, anharmonic crystals, and blood vessels in the human body (Ali, Saha and Chatterjee, 2017; Askar, 1982; Chen et al., 2020; Elgarayhi et al., 2013).

Solitons have since been modelled mathematically by the KdV equation, also known as the non-linear shallow water wave equation. This is done by integrating the Navier Stokes equation over the depth of the fluid. While the Navier Stokes equation describes the velocity of a fluid, the KdV equation is an equation describing the position (or height) of a water surface and is given by Equation 4.1 below

$$\frac{\partial u}{\partial t} + \eta u \frac{\partial u}{\partial x} + \mu^2 \frac{\partial^3 u}{\partial x^3} = 0, \tag{4.1}$$

where u is the height of the wave, x is a spatial dimension, t is time, and η and μ are two real constants. This equation has solitary wave solutions called solitons that are caused by a delicate balance between the equation's nonlinear and dispersive terms. The nonlinear term in this equation refers to the second term, $\eta u \frac{\partial u}{\partial x}$. It describes the self-interaction of the different components of the wave and is what allows this equation to describe soliton behaviour (Bridges and Reich, 2006).

While in the domain of classical physics, the KdV equation describes water waves, in the quantum world it can describe something quite different. Due to the fact that solitons do not lose their shape over time they have even been modelled as particles as well as waves. They have in fact been found to exist in superfluids by reducing the Gross-Pitaevskii equation[1] to a KdV equation using the reductive perturbation method[2] (Carretero-González et al., 2017). This method in fact shows that many equations can be broken down into KdV equations. This further points at the fact that KdV equations are actually part of a much broader class of equations that apply to a wide variety of fields.

[1]The Gross-Pitaevskii equation is a non-linear wave equation describing the ground state of a quantum system of multiple identical bosons. It is also conveniently known as the non linear Schroedinger equation (Antoine, Bao and Besse, 2013).

[2]The reductive perturbation method is a method for solving non linear hyperbolic systems of equations by converting them to a single non linear equation (Taniuti, 1974).

4.2 Discretisation

We begin by investigating a suitable numerical scheme for solving the KdV equation. The first to successfully solve the KdV equation numerically was an explicit finite difference scheme developed in 1965 by Zabusky and Kruskal at Bell Labs (Zabusky and Kruskal, 1965). Their technique solved the equation in a stable manner up until a point where it produced unrealistic oscillations that caused the simulation to diverge rapidly from real solutions. It produced accurate results up until a point where unrealistic oscillations caused results to become unusable. Modern explicit techniques have been developed that do not have this problem (Feng and Mitsui, 1998; Wang et al., 2008). In a comparison of finite difference and Chebyshev methods, the Chebyshev-collocation method was shown to be more efficient than finite difference schemes for short time integrations. However, it was concluded that it was not the best candidate in terms of stability due to its spectral properties (Skogestad and Kalisch, 2009). Due to the simplicity, stability, and relative efficiency of the explicit scheme proposed by Wang et al. (2008), it has been chosen here for solving the KdV equation.

The KdV equation is discretised from

$$\frac{\partial u}{\partial t} + \eta u \frac{\partial u}{\partial x} + \mu^2 \frac{\partial^3 u}{\partial x^3} = 0, \tag{4.2}$$

where u is the height of the wave, and η and μ are two real constants, into the form shown by Wang et al. (2008) in Equation 4.3.

$$\begin{aligned} \frac{1}{2}\left(\frac{u_{(j-1,k+1)} - u_{(j-1,k)}}{\Delta t} + \frac{u_{(j+1,k)} - u_{(j+1,k-1)}}{\Delta t}\right) \\ = -\eta \frac{u_{(j+1,k)} + u_{(j,k)} + u_{(j-1,k)}}{3} \frac{u_{(j+1,k)} - u_{(j-1,k)}}{2\Delta x} \\ - \frac{\mu^2}{2\Delta x^3}\left(u_{(j+2,k)} - 2u_{(j+1,k)} + 2u_{(j-1,k)} - u_{(j-2,k)}\right), \end{aligned} \tag{4.3}$$

where $u_{(j-1,k+1)}$ is the future value of u at $j-1$ that we would like to compute at time iteration $k+1$, etc. Each cell is separated by a distance Δx. We use the above equation to find the value of u at a particular cell j (or $j-1$) using the values of neighbouring cells from the previous time iterations. Rearranging and solving for $u_{(j-1,k+1)}$ we can solve the KdV equation in a stable and efficient manner using appropriate initial conditions. When solving this equation it can be useful to check the accuracy of the solution against analytic ones such as (Brauer, 2000). However, since later equations examined in this work have no such analytical solutions, we would like to

investigate other methods of measuring numerical accuracy such as tracking the constants of motion over time. This is discussed in further detail in Section 4.4. In Figure 4.1, the computational molecule for this three level scheme is shown.

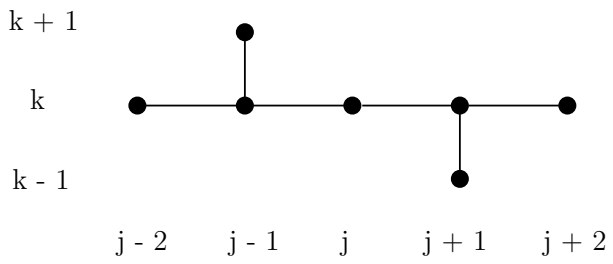


Figure 4.1: Computational molecule/stencil for the explicit numerical scheme for the KdV equation shown by Wang et al. (2008).

4.2.1 Initial and Boundary Conditions

In order to recreate the results from Zabusky and Kruskal (1965), it is important to understand the type of boundary conditions used in more detail. These boundary conditions are known as periodic or cyclic. As mentioned in Section 3.3, this means that changes can propagate freely between the left and right sides of the simulation space. This results in the computational molecule being wrapped around from one edge of the simulation space to the other. In this scheme we compute the $u_{(j-1,k+1)}$ term at each time iteration. When we compute the upper right cell in Figure 4.2, we can see that most of the values we need lie outside of the simulation space to the right. We get these values from their equivalent spots on the left hand side as if the two ends of the space were connected.

The initial conditions used by Wang et al. (2008) are given by

$$u_{(x,0)} = \cos(\pi x). \tag{4.4}$$

In order to solve Equation 4.3 we require data at time $k\Delta t$ and $(k-1)\Delta t$. Equation 4.4 gives us the data at $k-1$, but we also need to compute the values required at k. This predicament is known as the *hot-start problem*. To provide the values at k, the following second initial condition is used

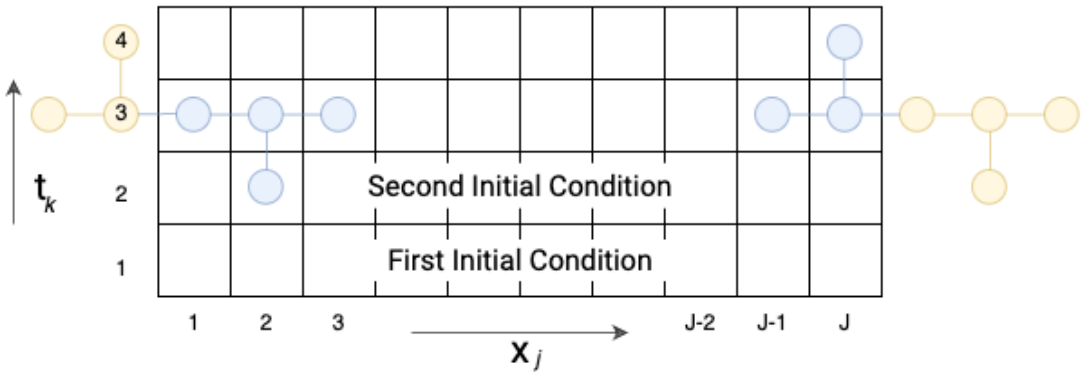


Figure 4.2: Computational molecule being wrapped around the edges of the simulation space with cyclic boundary conditions.

$$\begin{aligned}\frac{1}{\Delta t}\left(u_{(j,1)} - u_{(j,0)}\right) \\ = -\eta \frac{u_{(j+1,0)} + u_{(j,0)} + u_{(j-1,0)}}{3} \frac{u_{(j+1,0)} - u_{(j-1,0)}}{2\Delta x} \\ - \frac{\mu^2}{2\Delta x^3}\left(u_{(j+2,0)} - 2u_{(j+1,0)} + 2u_{(j-1,0)} - u_{(j-2,0)}\right). \quad (4.5)\end{aligned}$$

Equation 4.5 can then be rearranged to solve for $u_{(j,1)}$. In addition to the initial conditions, we also implement the following cyclic boundary conditions on the left and right boundaries. Figure 4.2 shows how the computational molecule/stencil overlaps the left boundary at $j = -1$ and right boundary at $j = J$. In this case the stencil is wrapped from one boundary to the other. So values of cells that are referenced outside the domain on the left are automatically mapped to cells inside the domain on the right. This mapping is defined below in the following equations. For computations near the left boundary we have

$$u(-1) = u(J) \quad (4.6)$$
$$u(-2) = u(J-1) \quad (4.7)$$
$$u(-3) = u(J-2), \quad (4.8)$$

and similarly near the right boundary:

$$u(J+1) = u(0) \tag{4.9}$$
$$u(J+2) = u(1) \tag{4.10}$$
$$u(J+3) = u(2). \tag{4.11}$$

4.2.2 Algorithm

This subsection examines the algorithm used for the explicit solution devised by Wang et al. (2008). As we can see in Algorithm 1 below, the numerical technique for solving the KdV equation is extremely simple. Since the approach of Wang et al. (2008) is an explicit one, the number of steps in the solution is limited to one mathematical operation per cell per iteration. This makes it very simple to implement and replicate. What is impressive is that this technique achieves convergence and long running stability due entirely to its derivation. Later techniques in this thesis employ the use of complex filtering techniques in order to maintain stability.

Algorithm 1: Pseudocode for solution of the 1D KdV equation.

```
Initialise variables;
Set initial conditions;
foreach iteration i in iterations n do
    foreach cell j in iteration i do
        Update boundary values;
        Compute solution u_(j,i+1);
        Compute error for iteration i;
    end
end
```

4.3 Numerical Results

As mentioned previously, we use the numerical technique devised by Wang et al. (2008) to solve the KdV equations. In reproducing the results of Wang et al. (2008), I have also been able to reproduce those of the original paper by Zabusky and Kruskal (1965). These results are shown in Figure 4.3. We begin with a cosine wave as the initial condition. In the first two frames we can see the valley of the wave move towards the left, creating a steeper incline on the right side of the wavefront. In the third frame we can see the point where the peak becomes unstable and begins to oscillate. The oscillations grow in size and begin to gain their own velocities, moving towards the left

boundary. At $t = 0.4$ s, the beginnings of these oscillations can be seen as the instabilities propagate towards the left. From here the oscillations will exit the simulation space on the left boundary and re-enter on the right boundary due to the cyclic boundary condition. In Figure 4.4 we see a snapshot of the waves at $t = 1.75$ s. The oscillations have grown into their own individual wave fronts moving in unison and have begun to superimpose with the initial wavefront.

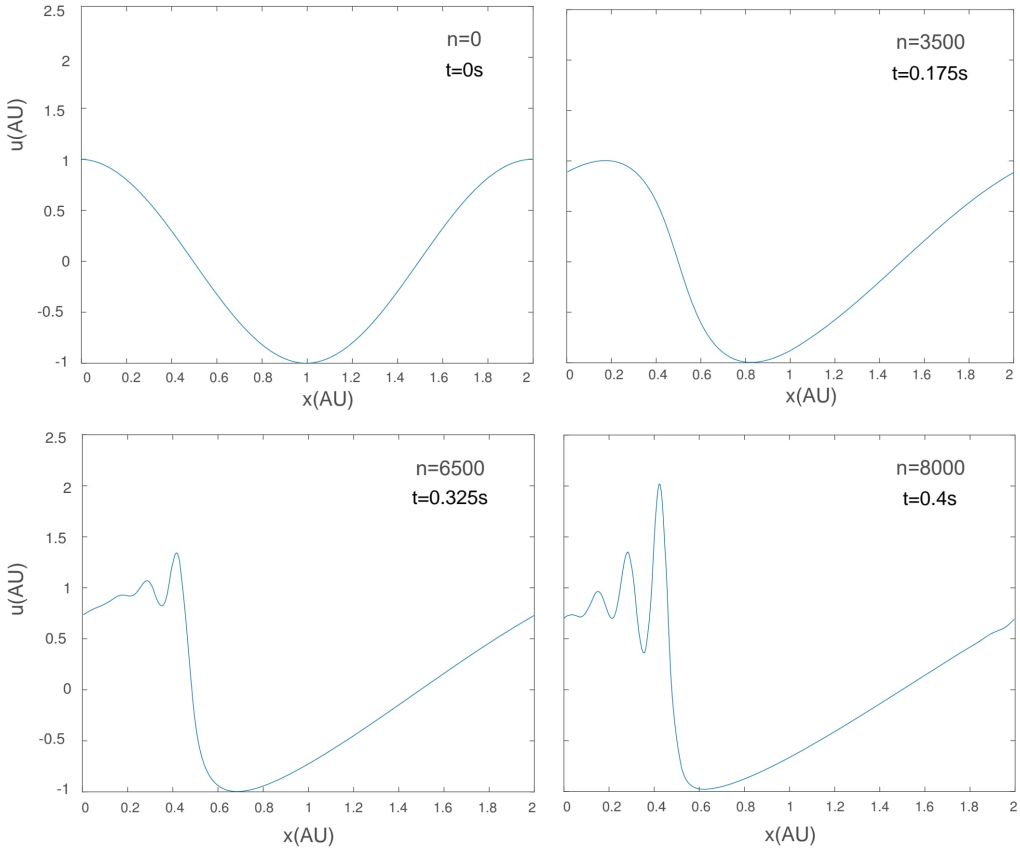


Figure 4.3: Plot of u as a function of x for various values of $t = 0$ s (top left), 0.175 s (top right), 0.325 s (bottom left) and 0.4 s (bottom right). The following parameters used to generate this result: $u(x, 0) = \cos(\pi x)$, Periodic/cyclic boundary conditions, $dt = 0.00005$, $dx = 2/399$.

As the process progresses, this becomes further complicated as they mix throughout the space. In Figures 4.5 we can see the final state reached by

Wang et al. (2008) next to my recreation of those same results shown in Figure 4.6. This set of results shows the long term stability of the numerical scheme. While the simulation by Zabusky and Kruskal (1965) produced unusable results at $t = 6.1275$ s, their numerical scheme is still unconditionally stable at $t = 40$ s and far beyond. Take note that this scheme uses no form of error correction during the computation of the solution. An analysis of the error involved in this scheme follows this section, but this information is not used in any way to enhance the simulation itself, nor does the the scheme employ any kind of filtering techniques. This approach is common place in more complex fluid solvers due to the large number of erroneous high frequency spectral components that are unavoidable.

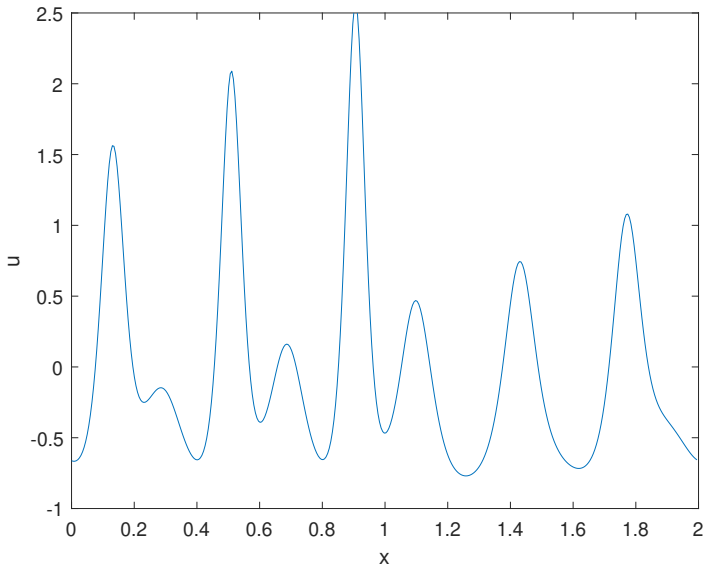


Figure 4.4: Plot of u as a function of x at $t = 1.75s$. The following parameters were used to generate this result: $u(x, 0) = \cos(\pi x)$, Periodic/cyclic boundary conditions, $dt = 0.00005$, $dx = 2/399$, $n = 35000$.

In Figure 4.7 I show the paths taken by the individual wave peaks through time. This figure shows a very interesting interaction between the wavefronts. Each peak appears to reach some stable horizontal velocity indicated by the angle of the path that is drawn in the figure. As an example of this acceleration we can see the left most wavefront at $t = 0$ s begins to move to the left before reversing its direction entirely and stabilising on some velocity. We can also observe that at the points where the larger waves (shown as

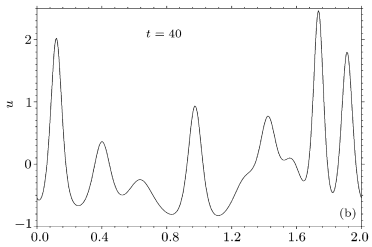


Figure 4.5: The KdV equation solution at $t = 40$ s by Wang et al. (2008).

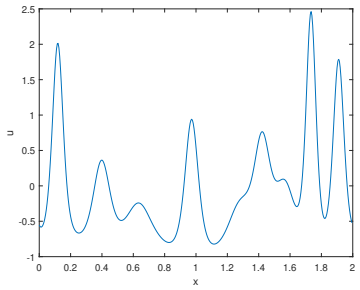


Figure 4.6: My simulation results at the same time of $t = 40$ s.

brighter shades of yellow) superimpose upon other waves they appear interact, causing a temporary change in their velocities during the interaction. This can bee seen as a break in the path that is traced by a given wavefront. While other types of wave phenomena also superimpose upon one another, they do not, in the process, affect each others velocity. This, however, is what appears to happen when solitons interact with one another, where each interaction results in a corresponding phase shift. The literature surrounding soliton interactions is rich and growing continuously. Similar soliton interactions have been thoroughly documented, specifically in the field of optics. Aitchison et al. (1991) document both attractive and repulsive interactions between spatial optical solitons. More recently Sun et al. (2023) conducted a thorough investigation of three-soliton interactions. In contrast, consider Figure 3.6 in Chapter 3 showing the wave paths traced in a solution of the wave equation where the superposition of waves does not alter their phase at all.

4.4 Stability Analysis

As mentioned previously, determining the success of a numerical simulation is of utmost importance. While a thorough analysis of the CFL condition[3] is always useful for determining the usable domain of the simulation and whether or not the scheme is convergent, it is not the sole predictor of the success of a numerical scheme. Determining the success of a simulation can be done by comparing the solutions with experimental results, analytical

[3]The Courant–Friedrichs–Lewy condition that defines the conditions under which convergence of the solution is guaranteed.

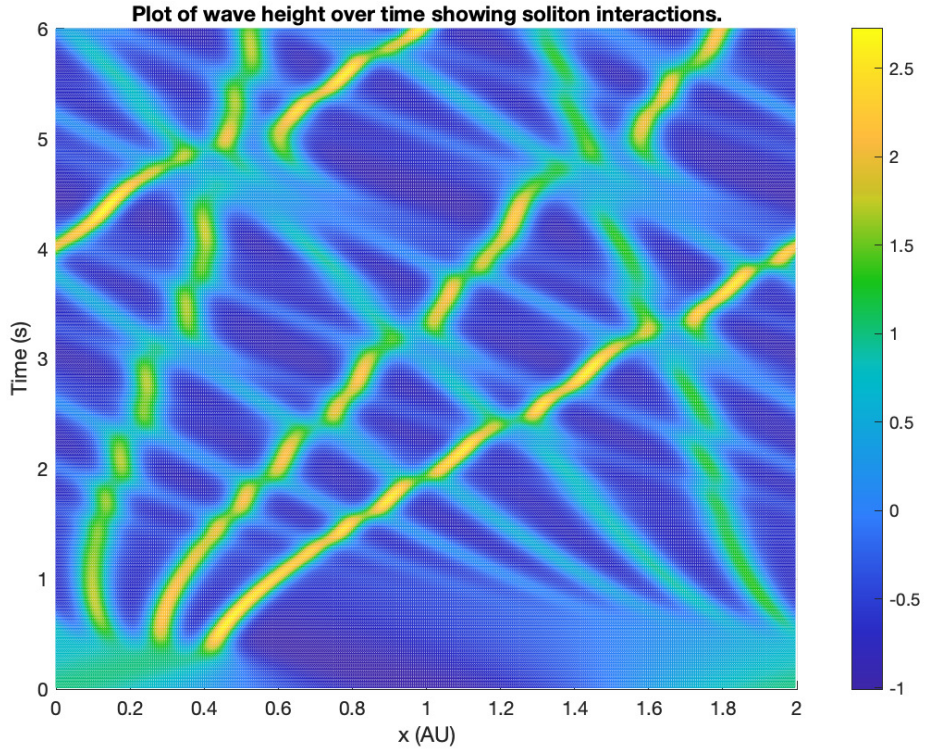


Figure 4.7: Plot of u as a function of x for $t = 0$ s to $t = 6$ s showing illustrating the paths followed by each wavefront. The following parameters were used to generate this result: $u(x, 0) = \cos(\pi x)$, Periodic/cyclic boundary conditions, $dt = 0.00005$, $dx = 2/399$, $n = 120000$.

solutions (where available) or by tracking the constants of motion of the system over time. For the Korteweg-de Vries equation under periodic boundary conditions, we know that the system has at least three physical constants of motion. They are the conservation laws of the system:

$$\begin{aligned} F_1(u) &= \int_0^2 u dx \\ F_2(u) &= \frac{1}{2}\int_0^2 u^2 dx \\ F_3(u) &= \int_0^2 \left\{ \frac{1}{2}\mu^2 u_x^2 - \frac{1}{6}u^3 \right\} dx, \end{aligned} \tag{4.12}$$

where $F_1(u)$ is the momentum conservation law, $F_2(u)$ is the energy conservation law, and $F_3(u)$ is the Hamiltonian functional for the Hamiltonian form of the KdV equation (Wang et al., 2008). The discrete versions of these equations are shown in Equation 4.13

$$\begin{aligned} F_1^h(\mathbf{u}) &= \sum_{i=1}^{n} u_i h \\ F_2^h(\mathbf{u}) &= \frac{1}{2}\sum_{i=1}^{n}\left(\frac{u_i + u_{i-1}}{2}\right)^2 h \\ F_3^h(\mathbf{u}) &= \sum_{i=1}^{n}\left\{\frac{1}{2}\mu^2 |\Delta + u_i|^2 - \frac{1}{6}u_i^3\right\} h, \end{aligned} \tag{4.13}$$

where $\Delta + u_i = (u_{i+1} - u_{i-1})/2h$ and h is the spatial step size. The above set of equations can be used to determine an estimate of the error and is given by the following equations

$$\begin{aligned} errorF_2(j\Delta t) &= F_2^h\left(u^j\right) - F_2^h\left(u^0\right) \\ errorF_3(j\Delta t) &= F_3^h\left(u^j\right) - F_3^h\left(u^0\right) \end{aligned} \tag{4.14}$$

By tracking any one of these errors in each iteration we can get an idea of the success of our simulation. In Figure 4.8 we can see that the $F_2(u)$ error remains relatively constant at around 10^{-2} throughout the simulation even after 200000 iterations. Throughout this duration the scheme is stable matching the results by Wang et al. (2008).

In Figure 4.9, below I show the unstable numerical solution of the KdV equation using what is known as the multi-symplectic six-point scheme. This result is drawn directly from the paper being discussed by Wang et al. (2008). This error is very similar to the errors considered in Chapter 3, Figure 3.9 and illustrates the way in which most compounding numerical errors accrue in the types of simulations considered in this thesis.

4.5 Summary

In this chapter the KdV equation was discretised and solved numerically using initial conditions of those of Wang et al. (2008). The results compare favourably with those of Wang et al. (2008). In the next chapter we will move on to discretising the Boussinesq equation.

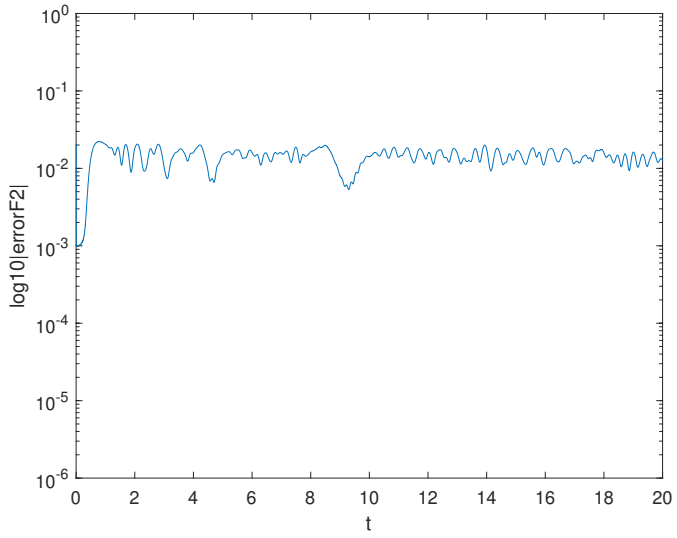


Figure 4.8: Plot of the $\log_{10}|errorF2|$ showing the conservation of energy of the system. The following parameters were used to generate this result: $u(x,0) = \cos(\pi x)$, Periodic/cyclic boundary conditions, $dt = 0.0001$, $dx = 0.01$, $n = 200000$. The results are smoothed using a moving mean of 100 data points.

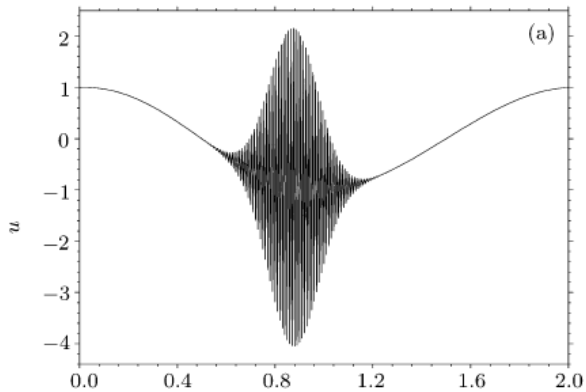


Figure 4.9: Plot of the unstable solution of the six point scheme from Wang et al. (2008).

Chapter 5

The Boussinesq Equation

5.1 Introduction

In this chapter we will give an introduction to the Boussinesq equation. This will be followed by numerical simulations and comparisons with experimental data. In the previous chapter, we discussed the Korteweg-De Vries (KdV) equation. We saw that while it can model many wave phenomena, it is best suited to modelling wave solitons. We move on now to the Boussinesq equation which is considered as the generalization of the KdV equation. It models solitary water waves travelling in multiple directions and caters for the effects of variable water depth on the evolution of waves (Kong and Wang, 2010; Scalerandi, 1997). It has been used for this purpose to simulate the evolution of deep sea waves and tsunamis (Adytia and Groesen, 2010). The Boussinesq equations have been used in conjunction with scale wave tank experiments to study deep sea waves which can be difficult to study in nature (Wei and Kirby, 1995). The Boussinesq equation offers a model for simulating near-shore wave evolution without the need for wave-tank experiments.

The way in which water waves change as they approach shorelines is remarkably complex. The Boussinesq family of equations attempt to describe this wave evolution taking into account conservation of mass and momentum laws as well as the varying depth of the water. They model the non-linear transformations that surface waves undergo in shallow water (Nwogu, 1993). Due to the complexity of this problem, many Boussinesq type equations have been formulated that describe these wave phenomena to different degrees of detail. The "Good" Boussinesq equation and the time-fractional Boussinesq equations are two such examples, each with their own limitations (Ismail and Mosally, 2014; Kong and Wang, 2010; Zhang et al., 2018). The Boussinesq

equation, like most nonlinear equations, will have numerical solutions that are sometimes highly unstable. It is therefore important to choose a technique that effectively minimizes computational error.

In this chapter we review the numerical technique of Wei and Kirby for solving the Boussinesq equation (Wei and Kirby, 1995). We start by defining the equations of the potential flow model it uses, then proceed into its predictor corrector time integration scheme. We then derive the matrix equations for solving the horizontal and vertical velocity components in the simulation - a step that is skipped in the literature I investigated. We then address the hot start problem, and describe the relevant boundary conditions for our simulations. The results of the Wei and Kirby technique will then be compared with experimental data in the chapters to follow.

5.2 Wei and Kirby Technique

The second numerical scheme we will explore is that of the Wei and Kirby group. Between 1995 and 1998, Wei and Kirby co-authored a number of papers on their numerical scheme for solving Boussinesq type equations (Kirby et al., 1998; Wei and Kirby, 1995; Wei et al., 1995). Their model uses the Boussinesq equations derived by Nwogu (1993) and a more complex time integration scheme known as the Adams-Bashfourth-Moulton Predictor Corrector method. The spatial domain is discretised using a regular grid of equally spaced cells. In Nwogu's Boussinesq equations, velocities are defined at some distance relative to the still water level. This formulation allows the equations to apply to a larger range of water depths by improving its linear dispersion properties (Nwogu, 1993). Modified versions of this model are still used extensively for Boussinesq simulations today (Martínez-Ferrer et al., 2018; Mehmood et al., 2016; Patel, Kumar and Rajni, 2020). The equations of Nwogu have been used extensively for near-shore ocean wave modelling and have only since been replaced by the more complex models of Chen (2006) and Shi et al. (2012) when the effects of wave breaking and porous ocean beds are necessary.

The Boussinesq equations are as follows:

$$\eta_t+\nabla\cdot(h+\eta)\,\mathbf{u}+\nabla\cdot\left\{\left(\frac{z_\alpha}{2}-\frac{h^2}{6}\right)h\nabla\left(\nabla\cdot\mathbf{u}\right)+\left(z_\alpha+\frac{h}{2}\right)h\nabla\left[\nabla\cdot(h\mathbf{u})\right]\right\}=0, \tag{5.1}$$

$$u_t \;+\; g\nabla\eta \;+\; (\mathbf{u}\cdot\nabla)\,\mathbf{u}+z_\alpha\left\{\frac{z_\alpha}{2}\nabla\left(\nabla\cdot u_t\right) \;+\; \nabla\left[\nabla\cdot(h\mathbf{u}_t)\right]\right\}=0, \tag{5.2}$$

where η is the water surface elevation, $\mathbf{u} = (u, v)$ is the horizontal surface velocity at arbitrary depth z_α, h is water depth, and g is the gravitational acceleration. The subscript t refers to time derivatives of the respective quantity. Here, Equations 5.1 and 5.2 are the conservation of mass and momentum equations for the system, respectively.

In the next section I will go into more detail on how these equations are solved using the potential flow model.

5.3 The Potential Flow Model

As shown in the approach of Wei and Kirby (1995), Boussinesq equations can be modelled in terms of a velocity potential function $\mathbf{U} = (U, V)$, where U and V are the velocity potential components in the x and y directions respectively. Doing this simplifies solving for u_t in Nwogu's conservation of momentum equation (Equation 5.2) as the form the equations take is more suited to the time integration scheme used (the Adams-Bashfourth-Moulton Predictor Corrector method). We define the surface elevation η from Equation 5.1 in terms of a potential, E, and the horizontal components of Equation 5.2, u_t and v_t in terms of some potentials U_t, and V_t as

$$\eta_t = E(\eta, u, v) \tag{5.3}$$
$$U_t = F(\eta, u, v) + [F_1(v)]_t \tag{5.4}$$
$$V_t = G(\eta, u, v) + [G_1(u)]_t, \tag{5.5}$$

where the subscript t refers to their time derivatives. The terms E, F, F_1, G, and G_1 are spatial derivatives of η, u, and v. For the derivation of these quantities see (Wei et al., 1995). These quantities are then defined as

$$\begin{aligned} E(\eta, u, v) = &- [(h+\eta)u]_x - [(h+\eta)v]_y \\ &- \left\{ a_1 h^3 (u_{xx} + v_{xy}) + a_2 h^2 \left[(hu)_{xx} + (hv)_{xy} \right] \right\}_x \\ &- \left\{ a_1 h^3 (v_{yy} + u_{xy}) + a_2 h^2 \left[(hv)_{yy} + (hu)_{xy} \right] \right\}_y, \end{aligned} \tag{5.6}$$

and

$$F(\eta, u, v) = -gn_x - (uu_x + vu_y) \tag{5.7}$$
$$G(\eta, u, v) = -gn_y - (vv_y + uv_x) \tag{5.8}$$
$$F_1(v) = -h\left[b_1 h v_{xy} + b_2 (hv)_{xy}\right] \tag{5.9}$$
$$G_1(u) = -h\left[b_1 h u_{xy} + b_2 (hu)_{xy}\right]. \tag{5.10}$$

The velocity potential functions themselves are defined as

$$U(u) = u + \left[b_1 h u_{xx} + b_2 (hu)_{xx}\right] \tag{5.11}$$
$$V(v) = v + \left[b_1 h v_{yy} + b_2 (hv)_{yy}\right], \tag{5.12}$$

where a_1, a_2, b_1, and b_2 are defined as

$$a_1 = \beta^2/2 - 1/6; a_2 = \beta + 1/2; b_1 = \beta^2/2; b_2 = \beta, \tag{5.13}$$

where $\beta = z_\alpha/h$, and $z_\alpha = 0.531h$ (Nwogu, 1993; Wei and Kirby, 1995).

5.4 Predictor Corrector Methods

We now take a brief aside to discuss the time discretisation scheme used in the potential flow model known as the predictor corrector method. Predictor corrector methods are used for solving differential equations, and they combine an explicit technique with an iterative implicit technique to approximate a solution. The technique consists of two steps. They first make a prediction and then attempt to refine this prediction iteratively to converge on a solution. This is done by feeding the prediction into the corrector equation and comparing this with the original prediction until an error criterion is satisfied. The scheme used in this work is known as the fourth order Adams-Bashforth-Moulton predictor corrector.

Given a function

$$y' = f(x, y), \tag{5.14}$$

we can find the value of y at some time step Δt in the future, y_{i+1} as

$$y_{i+1}^{1} = y_i + \frac{\Delta t}{12}\left(23f_i - 16f_{i-1} + 5f_{i-2}\right), \tag{5.15}$$

where $i = 3, 4, 5, ..., n$ and the superscript 1 refers to the first estimate of y_{i+1}, where i is the current time $t_i = i\Delta t$. This is known as the 3rd order Adams-Bashforth predictor method. From here we can use an implicit multistep/iterative method to better approximate y_{i+1} as

$$y_{i+1}^{k+1} = y_i + \frac{\Delta x}{24}\left(9f_{i+1}^{k} + 19f_i - 5f_{i-1} + f_{i-2}\right), \tag{5.16}$$

where $k = 1, 2.3, ...$ This is known as the 4th order Adams-Moulton corrector method. The superscript on y_{i+1} denotes the current prediction of the value of y_{i+1}.

This numerical method requires values from the previous two time steps and therefore requires priming in order to work. This can take the form of three time steps of initial condition or these values can be determined using a single step numerical scheme that does not depend on so many time intervals. In this work we follow the approach of Long (2006*b*) in Section 5.6 and use lower order Adams-Bashforth methods for the first 3 iterations of the scheme.

The **third order predictor** scheme is as follows:

$$\eta_{jk}^{i+1} = \eta_{jk}^{i} + \frac{\Delta t}{12}\left[23E_{jk}^{i} - 16E_{jk}^{i-1} + 5E_{jk}^{i-2}\right] \tag{5.17}$$

$$\begin{aligned} U_{jk}^{i+1} = U_{jk}^{i} + \frac{\Delta t}{12}&\left[23F_{jk}^{i} - 16F_{jk}^{i-1} + 5F_{jk}^{i-2}\right] \\ &+ 2F_{1_{jk}}^{i} - 3F_{1_{jk}}^{i-1} + F_{1_{jk}}^{i-2} \end{aligned} \tag{5.18}$$

$$\begin{aligned} V_{jk}^{i+1} = V_{jk}^{i} + \frac{\Delta t}{12}&\left[23G_{jk}^{i} - 16G_{jk}^{i-1} + 5G_{jk}^{i-2}\right] \\ &+ 2G_{1_{jk}}^{i} - 3G_{1_{jk}}^{i-1} + G_{1_{jk}}^{i-2}. \end{aligned} \tag{5.19}$$

The extra terms F_1 and G_1 in equations 5.18 and 5.19 come from the time derivatives of F and G from equations 5.4 and 5.5. For the derivation of these equations see the work of Wen Long in his PhD thesis entitled Boussinesq Modelling of Waves, Currents and Sediment Transport, (Long, 2006*b*).

The **fourth order** Adams-Moulton **corrector** scheme is given by

$$\eta_{jk}^{i+1} = \eta_{jk}^{i} + \frac{\Delta t}{24}\left[9E_{jk}^{i+1} + 19E_{jk}^{i} - 5Ejk^{i-1} + E_{jk}^{i-2}\right] \tag{5.20}$$

$$\begin{aligned} U_{jk}^{i+1} = U_{jk}^{i} + \frac{\Delta t}{24}&\left[9F_{jk}^{i+1} + 19F_{jk}^{i} - 5F_{jk}^{i-1} + F_{jk}^{i-2}\right] \\ &+ F_{1_{jk}}^{i+1} - F_{1_{jk}}^{i} \end{aligned} \tag{5.21}$$

$$\begin{aligned} V_{jk}^{i+1} = V_{jk}^{i} + \frac{\Delta t}{24}&\left[9G_{jk}^{i+1} + 19G_{jk}^{i} - 5G_{jk}^{i-1} + G_{jk}^{i-2}\right] \\ &+ G_{1_{jk}}^{i+1} - G_{1_{jk}}^{i}. \end{aligned} \tag{5.22}$$

These equations allow us to accurately approximate the values of η, U, and V at time Δt in the future. However, since equations 5.3, 5.4 and 5.5 are dependent simply on velocity and not the velocity potential, we must update the values of u each time we compute a new iteration of U. These new values of u will then be used in further iterations for computing future states. Transforming from U to u turns out to be a non-trivial operation as the equations relating the two quantities requires the solution of tridiagonal systems of equations. This is the topic of following subsection.

5.5 Solving for u from U

We now face the problem of converting back from the velocity potential to the actual velocity. As we derive a solution it is important to notice the interdependence of the velocity equations at each point on the grid. We begin by discretising Equation 5.23 below, and thereafter organise the resulting set of equations in a way that they can be solved simultaneously. In this derivation we refer to Figures 5.1 and 5.2 to describe the simulation space.

The horizontal velocity potential at each point $(x, y) = (j\Delta x, k\Delta y)$ is defined by the linear relation of U to u, and derivatives of u shown below

$$U = u + h\left[b_1 h u_{xx} + b_2 \left(hu\right)_{xx}\right], \tag{5.23}$$

and

$$V = v + h\left[b_1 h v_{yy} + b_2 \left(hv\right)_{yy}\right]. \tag{5.24}$$

Deriving an equation for $\mathbf{u}$ is the same for u and v so let us do our derivation using u. We begin by using a centred second order difference scheme to find u_{xx} and $(hu)_{xx}$ for each point on the grid. We have

$$u_{xx} = \frac{u_{j+1k} - 2u_{jk} + u_{j-1k}}{\Delta x^2}, \tag{5.25}$$

and

$$(hu)_{xx} = \frac{(h_{j+1k}u_{j+1k}) - 2\,(h_{jk}u_{jk}) + (h_{j-1k}u_{j-1k})}{\Delta x^2}, \tag{5.26}$$

Substituting 5.25 and 5.26 into 5.23 we get

$$U_{jk} = u_{jk} + h_{jk}\left[b_1 h_{jk}\frac{u_{j+1k} - 2u_{jk} + u_{j-1k}}{\Delta x^2} + b_2\frac{(h_{j+1k}u_{j+1k}) - 2\,(h_{jk}u_{jk}) + (h_{j-1k}u_{j-1k})}{\Delta x^2}\right]. \tag{5.27}$$

Grouping terms u_{j-1k}, u_{jk}, and u_{j+1k} we get the following equation

$$U_{jk} = u_{j-1k}\frac{h_{jk}\,(b_1 h_{jk} + b_2 h_{j-1k})}{\Delta x^2} + u_{jk}\left(1 - 2h_{jk}^2\frac{(b_1 + b_2)}{\Delta x^2}\right) + u_{j+1k}\frac{h_{jk}\,(b_1 h_{jk} + b_2 h_{j+1k})}{\Delta x^2}. \tag{5.28}$$

So for a given row, k, we get an equation for three unknowns, u_{j-1k}, u_{jk}, and u_{j+1k}. For each row in the matrix U we can solve this equation by first forming a tridiagonal matrix.

Equation 5.28 relates each cell in U to a set of three corresponding cells in u. Figure 5.2 shows the interdependence of the equations and how each set of unknowns in u overlap across a given row. Since the edge cells are on the boundary of the domain, they reference points outside of the simulation space at $j = 0 - 1$ and $j = J + 1$. To deal with this, we use the approach of Long (2006*a*) made specifically for wall boundaries. For points along the left wall, we replace u_{j-1k} with u_{j+1k}, and u_{J+1k} with u_{J-1k}, where J is the total number of cells in a row. Without this we are unable to compute values along boundaries, and we shrink the simulation space by one column on each side of the grid per iteration until failure.

Simplifying Equation 5.28, we let α_j, β_j, and γ_j be the constants for a given column j:

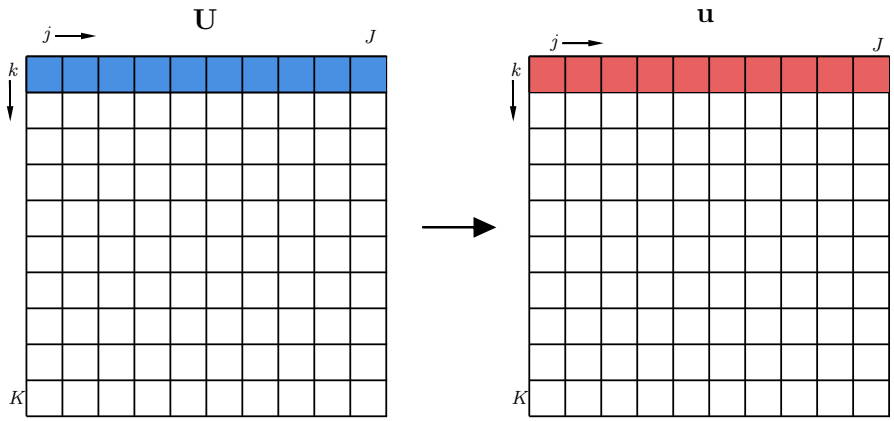


Figure 5.1: Equation 5.28 relates each cell in a row of U to a cell in the corresponding row of u.

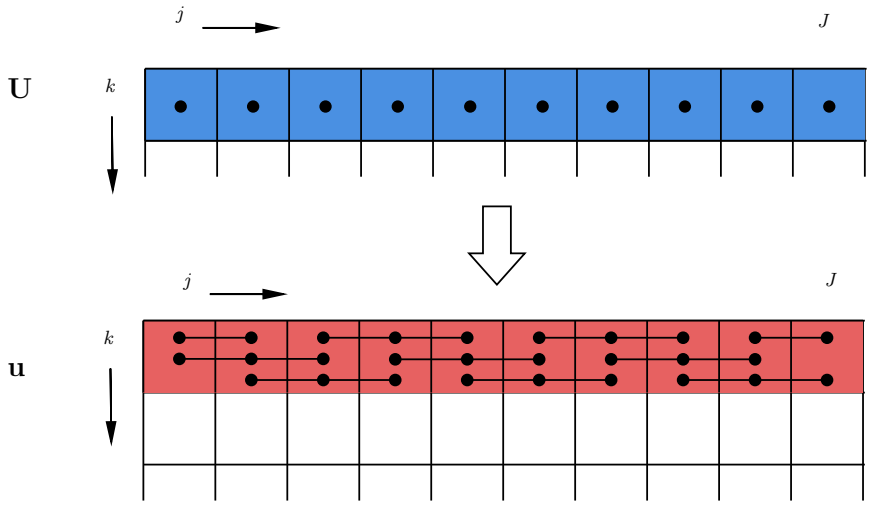


Figure 5.2: Each cell in U is described by three corresponding cells in the same row in u.

$$\alpha_j = \frac{h_{jk}}{\Delta x^2}\left(b_1 h_{jk} + b_2 h_{j-1k}\right) \tag{5.29}$$

$$\beta_j = 1 - 2h_{jk}^2 \frac{(b_1 + b_2)}{\Delta x^2} \tag{5.30}$$

$$\gamma_j = \frac{h_{jk}}{\Delta x^2}\left(b_1 h_{jk} + b_2 h_{j+1k}\right). \tag{5.31}$$

If we make an equation for u_{jk} for every j in a row k, we get a system of equations as follows below

$$\begin{array}{ccccccccccccc}
u_{0,k}\beta_0 & + & u_{1,k}\gamma_0 & + & 0 & + & 0 & + & \dots & + & 0 & = & U_{0,k} \\
u_{0,k}\alpha_1 & + & u_{1,k}\beta_1 & + & u_{2,k}\gamma_1 & + & 0 & + & \dots & + & 0 & = & U_{1,k} \\
0 & + & u_{1,k}\alpha_2 & + & u_{2,k}\beta_2 & + & u_{3,k}\gamma_2 & + & \dots & + & 0 & = & U_{2,k} \\
0 & + & 0 & + & u_{2,k}\alpha_3 & + & u_{3,k}\beta_3 & + & \dots & + & 0 & = & U_{3,k} \\
\vdots & & & & & & & \ddots & & & & & \vdots \\
\vdots & & & & & & & & \ddots & & & & \vdots \\
\vdots & & & & & & & & & \ddots & & & \vdots \\
0 & + & 0 & + & 0 & + & 0 & + & u_{J-1,k}\alpha_J & + & u_{J,k}\beta_J & = & U_{J,k}
\end{array}$$

From this we can create a matrix equation to solve simultaneously for $u_{j,k}$ for all $j \in [0, 1, 2, ..., J-1, J]$. This equation has the form

$$Ax = b, \tag{5.32}$$

where we are solving for the matrix x. The coefficient matrix A is therefore

$$A = \begin{bmatrix}
\beta_0 & \gamma_0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\
\alpha_1 & \beta_1 & \gamma_1 & 0 & 0 & \dots & 0 & 0 & 0 \\
0 & \alpha_2 & \beta_2 & \gamma_2 & 0 & \dots & 0 & 0 & 0 \\
0 & 0 & \alpha_3 & \beta_3 & \gamma_3 & \dots & 0 & 0 & 0 \\
\vdots & & & \ddots & \ddots & \ddots & & & \vdots \\
\vdots & & & & \ddots & \ddots & \ddots & & \vdots \\
\vdots & & & & & \ddots & \ddots & \ddots & \vdots \\
0 & 0 & 0 & 0 & 0 & \dots & \alpha_{J-1} & \beta_{J-1} & \gamma_{J-1} \\
0 & 0 & 0 & 0 & 0 & \dots & 0 & \alpha_J & \beta_J
\end{bmatrix}, \tag{5.33}$$

the matrix for the dependent variable, x, is

$$x = \begin{bmatrix} u_{0,k} \\ u_{1,k} \\ u_{2,k} \\ \vdots \\ \vdots \\ \vdots \\ u_{J-1,k} \\ u_{J,k} \end{bmatrix}, \tag{5.34}$$

and b is defined for each row k in U as

$$b = \begin{bmatrix} U_{0,k} \\ U_{1,k} \\ U_{2,k} \\ \vdots \\ \vdots \\ \vdots \\ U_{J-1,k} \\ U_{J,k} \end{bmatrix}. \tag{5.35}$$

The full equation is given as

$$\begin{bmatrix} \beta_0 & \gamma_0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ \alpha_1 & \beta_1 & \gamma_1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & \alpha_2 & \beta_2 & \gamma_2 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & \alpha_3 & \beta_3 & \gamma_3 & \dots & 0 & 0 & 0 \\ \vdots & & & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & & & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & \alpha_{J-1} & \beta_{J-1} & \gamma_{J-1} \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & \alpha_J & \beta_J \end{bmatrix} \begin{bmatrix} u_{0,k} \\ u_{1,k} \\ u_{2,k} \\ \vdots \\ \vdots \\ \vdots \\ u_{J-1,k} \\ u_{J,k} \end{bmatrix} = \begin{bmatrix} U_{0,k} \\ U_{1,k} \\ U_{2,k} \\ \vdots \\ \vdots \\ \vdots \\ U_{J-1,k} \\ U_{J,k} \end{bmatrix}, \tag{5.36}$$

and can be solved for all the values of u in the current row, k. This is then repeated for all the rows. Each row has an associated $J \times J$ sized matrix, and the entire space is therefore described by a set of these matrices of size $J \times J \times K$. This matrix of coefficients is a function of the constants α, β, and

γ, which are in turn only functions of h, b_1, and b_2, it is constant in time. Because of this, this matrix can be precomputed for use each iteration.

Similarly we can derive a system of equations for V and v for each column, j. In this case, we deal with derivatives of u in the y direction. The associated coefficient matrix for the entire space will be of size $K \times K \times J$. For v the equations become

$$\begin{bmatrix} \epsilon_0 & \zeta_0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ \delta_1 & \epsilon_1 & \zeta_1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & \delta_2 & \epsilon_2 & \zeta_2 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & \delta_3 & \epsilon_3 & \zeta_3 & \dots & 0 & 0 & 0 \\ \vdots & & & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & & & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & \delta_{K-1} & \epsilon_{K-1} & \zeta_{K-1} \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & \delta_K & \epsilon_K \end{bmatrix} \begin{bmatrix} v_{j,0} \\ v_{j,1} \\ v_{j,2} \\ \vdots \\ \vdots \\ \vdots \\ v_{j,K-1} \\ v_{j,K} \end{bmatrix} = \begin{bmatrix} V_{j,0} \\ V_{j,1} \\ V_{j,2} \\ \vdots \\ \vdots \\ \vdots \\ V_{j,K-1} \\ V_{j,K} \end{bmatrix}, \tag{5.37}$$

where instead of α, β, and γ we use corresponding variables for the y direction, δ, ϵ, and ζ:

$$\delta_k = \frac{h_{jk}}{\Delta x^2} \left(b_1 h_{jk} + b_2 h_{jk-1} \right), \tag{5.38}$$

$$\epsilon_k = 1 - 2h_{jk}^2 \frac{(b_1 + b_2)}{\Delta x^2}, \tag{5.39}$$

$$\zeta_k = \frac{h_{jk}}{\Delta x^2} \left(b_1 h_{jk} + b_2 h_{jk+1} \right). \tag{5.40}$$

5.6 The Hot Start Problem

As mentioned in the section above, the predictor corrector scheme for the main algorithm requires information from two previous iterations in time in order to make a prediction about the next state of the system in the next iteration in time. These two initial states are defined by initial condition equations. To compute values for the first two states we use lower order ABM schemes as detailed by Long (2006*a*).

For the first time step we use the 1st order predictor and the 2nd order corrector, where values at i=1 are known

$$\eta_{jk}^{i+1} = \eta_{jk}^{i} + \Delta t E_{jk}^{i} \tag{5.41}$$

$$U_{jk}^{i+1} = U_{jk}^{i} + \Delta t F_{jk}^{i} \tag{5.42}$$

$$V_{jk}^{i+1} = V_{jk}^{i} + \Delta t G_{jk}^{i}, \tag{5.43}$$

and the corrector

$$\eta_{jk}^{i+1} = \eta_{jk}^{i} + \frac{\Delta t}{2}\left[E_{jk}^{i+1} + E_{jk}^{i}\right] \tag{5.44}$$

$$U_{jk}^{i+1} = U_{jk}^{i} + \frac{\Delta t}{2}\left[F_{jk}^{i+1} + F_{jk}^{i}\right] + \left(F_{1_{jk}}^{i+1} - F_{1_{jk}}^{i}\right) \tag{5.45}$$

$$V_{jk}^{i+1} = V_{jk}^{i} + \frac{\Delta t}{2}\left[G_{jk}^{i+1} + G_{jk}^{i}\right] + \left(G_{1_{jk}}^{i+1} - G_{1_{jk}}^{i}\right), \tag{5.46}$$

For the 2nd time step, we use the 2nd order predictor and 3rd order corrector where i=2

$$\eta_{jk}^{i+1} = \eta_{jk}^{i} + \frac{\Delta t}{2}\left[3E_{jk}^{i} - E_{jk}^{i-1}\right] \tag{5.47}$$

$$U_{jk}^{i+1} = U_{jk}^{i} + \frac{\Delta t}{2}\left[3F_{jk}^{i+1} - F_{jk}^{i}\right] + \left(F_{1_{jk}}^{i} - F_{1_{jk}}^{i-1}\right) \tag{5.48}$$

$$V_{jk}^{i+1} = V_{jk}^{i} + \frac{\Delta t}{2}\left[3G_{jk}^{i+1} - G_{jk}^{i}\right] + \left(G_{1_{jk}}^{i} - G_{1_{jk}}^{i-1}\right), \tag{5.49}$$

and the corrector

$$\eta_{jk}^{i+1} = \eta_{jk}^{i} + \frac{\Delta t}{12}\left[5E_{jk}^{i+1} + 8E_{jk}^{i} - E_{jk}^{i-1}\right] \tag{5.50}$$

$$U_{jk}^{i+1} = U_{jk}^{i} + \frac{\Delta t}{12}\left[5F_{jk}^{i+1} + 8F_{jk}^{i} - F_{jk}^{i-1}\right] + \left(F_{1_{jk}}^{i+1} - F_{1_{jk}}^{i}\right) \tag{5.51}$$

$$V_{jk}^{i+1} = V_{jk}^{i} + \frac{\Delta t}{12}\left[5G_{jk}^{i+1} + 8G_{jk}^{i} - G_{jk}^{i-1}\right] + \left(G_{1_{jk}}^{i+1} - G_{1_{jk}}^{i}\right). \tag{5.52}$$

Using Equations 5.41 to 5.52, we can approximate the values of η, U, and V for the first three time intervals. From iteration 3 and onwards we can use Equations 5.17 to 5.22 defined earlier.

5.7 Boundary Conditions

The simulations done by Wei and Kirby (1995) use three types of boundary conditions: reflective boundaries that perfectly reflect waves incident on the boundary; wavemaker boundaries that generate waves moving in to the simulation space; and open/radiating boundaries that absorb all incoming waves.

5.7.1 Reflective Boundaries

Reflective boundaries for the Boussinesq equation are described by three boundary conditions. The first, a Dirichlet condition, states that the horizontal surface velocity component u in the direction of the boundary normals η must be zero at the boundaries as described by

$$\boldsymbol{u} \cdot \boldsymbol{\eta} = 0, \tag{5.53}$$

where η is the normal to the walls boundaries as illustrated in Figure 5.3.

Equation 5.54 below is a Neumann condition that describes the gradient of the water surface. It states that the gradient of η in the direction of the normals must be zero at the boundaries as described by

$$\nabla\eta \cdot \boldsymbol{n} = 0. \tag{5.54}$$

To represent this numerically we set the values at the boundary to equal the adjacent cells in the column or row. Equation 5.55 below, another Neumann condition, states that the gradient of the velocity component perpendicular to the normal (parallel to the boundary) must be zero along the normal:

$$\frac{\partial \boldsymbol{u}_T}{\partial \boldsymbol{n}} = 0. \tag{5.55}$$

This represents the no-shear condition for fluid near the boundary, i.e., fluid travelling in two adjacent cells parallel to a boundary have the same velocity.

From this we see that the x velocity components must be zero on the left and right and the y velocity components must be zeros on the top and bottom boundaries as described by Equations 5.56, 5.58, 5.60, and 5.62 as described by

$$\frac{\partial \eta}{\partial x}(j, K) = 0 \tag{5.56}$$

$$\frac{\partial v}{\partial x}(j, K) = 0, \tag{5.57}$$

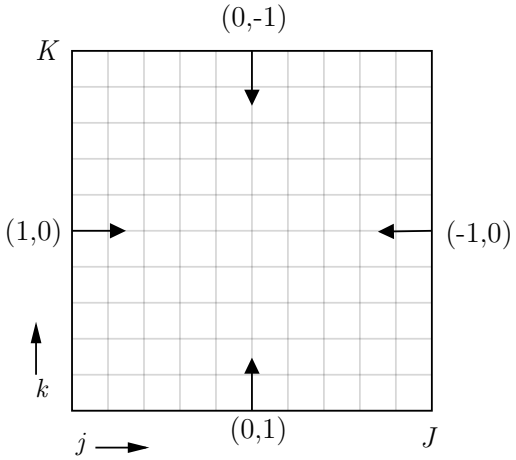


Figure 5.3: The normals to the boundary walls in the simulation space at $(j,k) = (J,k), (j,0), (j,K), (0,k)$.

for the right boundary,

$$\frac{\partial \eta}{\partial x}(j,0) = 0 \tag{5.58}$$

$$\frac{\partial v}{\partial x}(j,0) = 0, \tag{5.59}$$

for the left boundary,

$$\frac{\partial \eta}{\partial y}(J,k) = 0 \tag{5.60}$$

$$\frac{\partial u}{\partial y}(J,k) = 0, \tag{5.61}$$

for the top boundary,

$$\frac{\partial \eta}{\partial y}(0,k) = 0 \tag{5.62}$$

$$\frac{\partial u}{\partial y}(0,k) = 0, \tag{5.63}$$

and for the bottom boundary.

5.7.2 Wavemaker boundaries

When injecting waves into the simulation, we use what is known as wavemaker boundaries. To do this we simply set the surface elevation at the boundary to be some function of time. From this, the associated horizontal velocity components are also computed for each corresponding time instance by the following functions:

$$u_t = \frac{\omega}{kh_0[1-(\alpha+1/3)(kh_0)^2]}\eta_t cos\theta, \tag{5.64}$$

$$v_t = \frac{\omega}{kh_0[1-(\alpha+1/3)(kh_0)^2]}\eta_t sin\theta, \tag{5.65}$$

where k is wave number, h_0 is resting water surface depth, and θ is angle of wave propagation relative to the x axis.

Another point to note is that the boundary of a simulation space often consists of more than one border point. In order for changes to be propagated across a set of cells, boundary conditions must be specified over two or more adjacent boundary points. For this reason, the boundary condition is specified as

$$n_{(d-i)} = Asin\left(2\pi f\left(t+idt\right)\right), \tag{5.66}$$

where d is the boundary depth, i is the cell index from the x axis, and t is the current time for the simulation step. With all of the theory now established, the algorithm can be implemented using Algorithm 2.

5.8 Algorithm

Algorithm 2: Pseudocode for solving the 2D Boussinesq equation

foreach *iteration* i *in iterations* n **do**
  `// Prediction steps;`
  if *first iteration* **then**
    Compute E, F, G, F_1, and G_1 at i;
    Make 1st order Adams-Bashfourth prediction for η, U, and V at $i+1$;
  end
  else if *second iteration* **then**
    Make 2nd order Adams-Bashfourth prediction for η, U, and V at $i+1$;
  end
  else
    Make 3rd order Adams-Bashfourth prediction for η, U, and V at $i+1$;
  end
  Compute u from U at $i+1$ and u coefficient matrix;
  Compute v from V at $i+1$ and v coefficient matrix;
  Apply boundary conditions;
  `// Corrector steps;`
  while *Error in* η *or* $u <$ *some tolerance* **do**
    if *first iteration* **then**
      Apply 2nd order Adams-Moulton correction to η, U, and V at $i+1$;
    end
    else if *second iteration* **then**
      Apply 3rd order Adams-Moulton correction to η, U, and V at $i+1$;
    end
    else
      Apply 4th order Adams-Moulton correction to η, U, and V at $i+1$;
    end
    Compute u from U at $i+1$ and u coefficient matrix;
    Compute v from V at $i+1$ and v coefficient matrix;
    Apply boundary conditions;
    Compute E, F, G, F_1, and G_1 at $i+1$;
    Store current estimates for η, u, and v for $i+1$;
    Compute error for $i+1$ using estimates;
    if *number of corrector iterations* $>$ *iteration limit* **then**
      Fail due to non-convergeance;
    end
    Apply filtering to η, u, and v;
  end
end

5.9 Summary

In this chapter we derived the equations used for solving Boussinesq equation numerically using the potential flow model described by Wei and Kirby (1995). This will be used in Chapter 7 to attempt to solve the Boussinesq equation numerically. Until then we take a brief aside in the next chapter to analyse Boussinesq wave data captured from a physical wave tank experiment that was done by Mukaro et al. (2013).

Chapter 6

Experimental wave analysis

6.1 Introduction

We will now use the theory developed in this chapter to simulate waves in a wavetank. The simulation will be for the following case. Figure 6.1 shows a schematic of a real wave tank that was used by Mukaro et al. (2013) to create Boussinesq type beach waves by oscillating a paddle at one end and letting the resulting waves approach a sloped embankment on the right. The results of this simulation will be discussed in this chapter and Chapter 7, but before we go into the simulation results it will be worthwhile to examine results from the experimental case of Mukaro et al. (2013). The experimental setting will be used to explain the techniques for analysing Boussinesq waves and these same techniques will be applied in Chapter 7 to the simulated data.

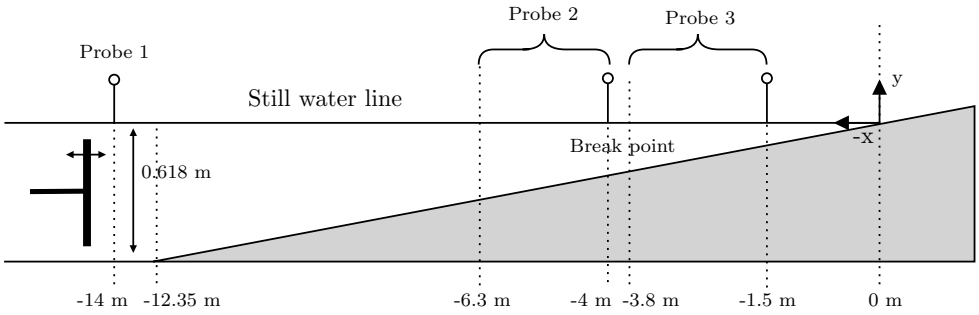


Figure 6.1: Experimental setup showing the water tank with a constant-slope floor profile beginning at -12.35 m from the shoreline, a horizontally oscillating paddle on the left of the diagram and an equilibrium water depth of 0.618 m at its deepest point where the floor is flat. The positions of probes 2 and 3 were changed in successive runs of the experiment, shown in Table 6.1

In this chapter I will introduce a physical experiment that was done by Dr Mukaro and Dr Govender to generate Boussinesq type beach waves in a wave tank with an oscillating paddle at one end (Mukaro et al., 2013). I will then present the results of my own independent analysis of their data. I will explain the physical apparatus used as well as the method by which the data was collected from the experiment. Along with my analysis, I will also detail each algorithm that was used to conduct my analysis. This analysis will act as a reference for my own numerical simulation and the aim is that the simulation should be able to predict similar results to what was seen in the real experiment. I will do my own independent analysis of Dr Mukaro's data including wave height analysis, phase velocity calculations and, further, I will also undertake a frequency analysis of the data, which was not done by Mukaro et al. (2013).

6.2 Experimental setup

The physical experiment consists of a rectangular wave tank with a movable paddle at one end. The paddle oscillates horizontally with a frequency of 0.4 Hz producing waves that propagate down the length of the tank. The floor of the tank has a constant upward slope preceded by a small flat region. The slope protrudes above the water surface at one end creating a shoreline as shown in Figure 6.1.

This apparatus is able to generate Boussinesq type waves that begin as sinusoidal waves on one end of the tank and form breaking waves on the shore at the other end of the tank. Along the length of the tank are three probes that constantly sample the height of the water surface about its equilibrium position. Over multiple runs these probes are used to capture the time series of the surface elevation for a 2 minute period at various positions along the tank. The experiments were started with probes initially located as indicated in Figure 6.1 at $x = -14$ m, $x = -4$ m, and $x = -1.5$ m. A time series was capture at these positions, then probes 2 and 3 were moved 10 cm toward the paddle while keeping probe 1 fixed. Then a new set of time series was captured. By repeating this over multiple runs the time series of the surface elevation was captured every 10 cm between -1.5 m and -6.3 m. In each of these runs the time series of the wave was sampled every 20 ms. Since each experiment run lasted for 2 minutes, only subsections of the data are shown in all figures shown below.

Table 6.1 below shows the probe positions for all runs of the experiment. Probe 1 was kept at the same position for each run, while Probe 2 and Probe 3 were moved further away from the shoreline with each run. The experiments

are numbered 1 through 24, followed by a 25th documented experiment that is a rerun of experiment 13 except with Probe 3 being measured at 2.7 m from the shoreline instead of 2.8 m.

Table 6.1: Experiment probe positions (m)

Experiment	**Probe 1**	**Probe 2**	**Probe 3**
1	14	4.0	1.5
2	14	4.1	1.6
3	14	4.2	1.7
4	14	4.3	1.8
5	14	4.4	1.9
6	14	4.5	2.0
7	14	4.6	2.1
8	14	4.7	2.2
9	14	4.8	2.3
10	14	4.9	2.4
11	14	5.0	2.5
12	14	5.1	2.6
13	14	5.2	2.8
14	14	5.3	2.9
15	14	5.4	3.0
16	14	5.5	3.1
17	14	5.6	3.2
18	14	5.7	3.3
19	14	5.8	3.4
20	14	5.9	3.5
21	14	6.0	3.6
22	14	6.1	3.7
23	14	6.2	3.8
24	14	6.3	3.8
30	14	5.2	2.7

In the rest of the chapter we will refer to the time series at a particular horizontal position along the tank relative to the still water mark on the beach, which we take as $x = 0$ m. The equilibrium position of the water surface is the still water level of 0.618 m.

6.3 Data inspection and cleaning

The output from Dr Govenders experiments were plain text files generated by the data acquisition units used for the experiments. In order to import all of the data into MATLAB I needed to do some preprocessing on the original text files to transpose them into a more modern tab delimited format. This was done using a simple Python script that looped through all of the data and applied the required formatting rules to all rows of data across all of the files. Once this was done I began creating a comprehensive object oriented data model in MATLAB to represent the data and metadata of each experiment. This was then used in the following sections for further analysing the data. In the next section I detail a simple phase adjustment algorithm that I wrote to synchronise the phases of the signals from the three probes in order to more clearly present the signal data. In Figure 6.2 we see a 10 second subplot of the original data captured by Dr Govender's experimental apparatus. The figure shows the height of the water at three different probes positioned in the tank over time. Each probe captures unique characteristics about the waves as they travel along the tank. Since waves travel from the left of the tank towards the right, the signals measured on the probes are flipped as the front of each wave is recorded first. This is why the steep side of the waves appears on the left in the figure. Here you can see the shapes of three waves - one for each probe. On Probe 3 (yellow) you can already see waves with the distinctive nonlinear shape with a steep slope on one side followed by a tapering slope on the other.

MATLAB was used for all of the following analysis in this chapter with a strict emphasis on not using external packages. This ensured I had full control over and understanding of all of the software I wrote. This code can be found on my GitHub account by searching for my name or going to the following URL: *https://github.com/JordanScarrott/boussinesq-waves*.

6.3.1 Phase adjustment

Upon inspection of the initial data it was apparent that the signals from each probe were not in phase. Naturally the position of the probes in the experimental setup meant that waves did not arrive at each one in the same phase. In order to present more usable data I wrote a phase correcting algorithm to shift all the signals into the same phase so that, for plotting and comparison purposes, they all begin on a zero crossing position with a positive gradient. The algorithm finds the first zero crossing of each signal that has a positive gradient and uses this to shift all signals into the same phase. The pseudocode for the simple phase adjustment algorithm is shown

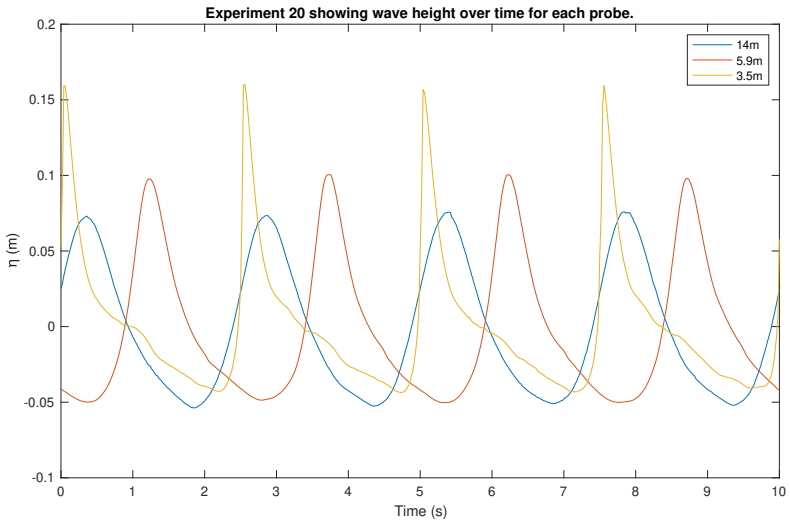


Figure 6.2: Initial data from experiment 20 showing out of phase signals for time series captured at $x = -14$ m, -5.9 m, and -3.5 m, where η is the instantaneous wave height.

in Algorithm 3.

Algorithm 3: Pseudocode for the phase adjustment algorithm

```
foreach signal in experiment do
    Take the sign of each data point;
    Compute the forward difference of this signed data;
    Find the first element where the sign changes from −1 to 1;
    Crop out all data before this point;
end
Crop ends of all signals to the same length;
```

This algorithm is well suited to MATLAB and makes use of its efficient vectorizing operations. When applying this algorithm to the initial data we get the following more usable result shown in Figure 6.3. This wave phase synchronisation algorithm was applied to all 25 experiment datasets. This work benefited greatly from the the object oriented data model that was used and as a result, applying this algorithm (or any other in this chapter) to any number of the datasets became trivial.

Further analysis of this data will be discussed in the following section.

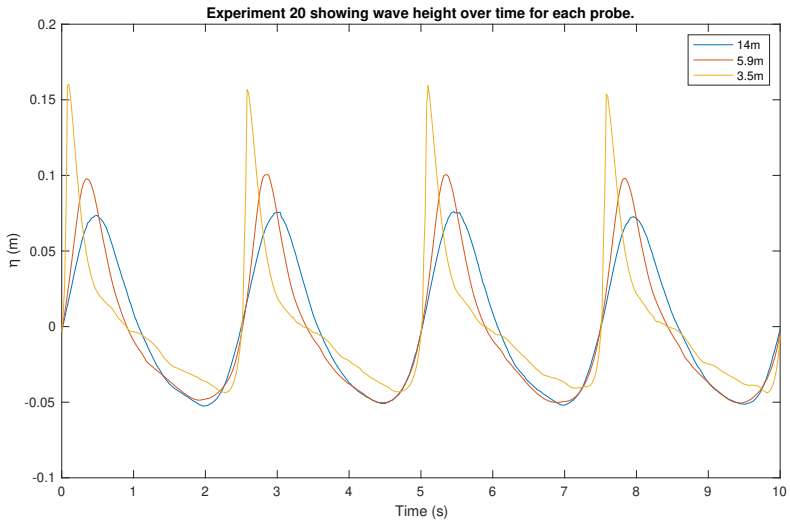


Figure 6.3: Processed data from Experiment 20 with phase-synchronised probe signals.

6.4 Analysis of Experimental Results

In this section I will present my own independent analysis of the probe data from different runs of Dr Govender's experiments. We will discuss the characteristics of the waves at three regions along the tank: near the paddle where the waves are generated, a region half way up the slope, and the region where the waves begin to break. I will refer to these regions as the near-paddle region, the near-breaking region, and the breaking region respectively. An analysis of the frequency spectra in these regions will follow in Section 6.6.

6.4.1 Wave height analysis

In this section, an independent analysis of the average wave height[1], H, is presented at every probe position along the wave tank. The results from this analysis are shown in Figures 6.4 and 6.7 and referenced throughout this section.

Algorithm 4 below shows the pseudocode for computing the average wave height of each signal. Given a signal, we determine the locations of all the

[1]The wave height is defined as the distance from the bottom of the trough of the wave to the top of the crest.

positive gradient zero crossings and divide the signal into subregions each containing a single wave period. The goal is then to compute the difference between the maximum and minimum values in each subregion and average them to get the average wave height for the signal. Due to the fact that there are often oscillations around the $y = 0$ m line, I wrote a minimum distance zero crossing detection algorithm to remove unwanted zero crossings near the expected zero crossing location for each period. For a given signal, I compute the Fourier transform and determine the dominant, non-zero frequency component of the wave (the fundamental frequency of the wave). This allows us to know how often to expect a zero crossing to occur[2] . We can then limit the number of zero crossings that occur within some area, $\delta_{(f)}$, around the expected boundary crossing to one, where δ is a function of the dominant frequency, f, given by

$$\delta(f) = \frac{(1-\gamma)}{f}, \tag{6.1}$$

where the tolerance γ is between 0 and 1 and δ is measured in seconds. This tolerance allows us to accommodate slight variations in the exact location of the zero crossing.

Algorithm 4: Pseudocode for computing the average wave height of a signal

```
Take the sign of each data point;
Compute the forward difference of this signed data;
Find all the array indices where the sign changes from −1 to 1;
Compute dominant non-zero frequency component from its FFT;
Compute the period of the wave;
foreach zeroCrossing z in zeroCrossings do
    if any two zero crossings are within δs of each other then
        Remove one of them.
    end
end
foreach wave period in periods do
    Compute wave height as difference between max and min values;
end
Average all of the wave heights;
Compute all of the standard deviations;
```

[2]Although we know from observations of Figure 6.2 that the wave period is around 2.5 s or 0.4 Hz, using the FFT[3]allows us to treat each signal individually.

[3]A high performance algorithm for computing the Discrete Fourier Transform of a signal, thus giving it the name Fast Fourier Transform

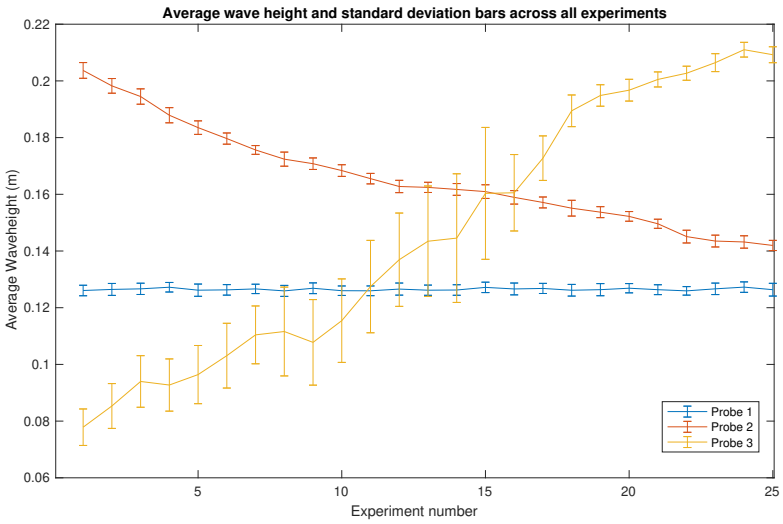


Figure 6.4: Average wave height and standard deviations from all probes across all experiments.

This algorithm ensures that every wave height calculation is done using exactly a single wave period within some error range of $\pm\delta/2$. For this analysis, δ was set to 0.1. This algorithm was then used to compute the average wave height and standard deviation for every position along the length of the tank. Plotting the average wave height for each experiment gives us Figure 6.4 and plotting the average wave height across the length of the tank gives us Figure 6.7. These results will be discussed in the following sub sections.

6.4.2 Near paddle characteristics

We know that the paddle oscillates sinusoidally with a constant frequency of 0.4 Hz. This oscillation creates a sinusoidal surface level oscillation that propagates along the length of the tank. This generated wave is the wave we expect to see at $x = -14$ m. However, when examining the blue signal at this point nearest to the paddle in Figure 6.3 it is clear that there is a subtle asymmetry to the wave and appears to slope to the left. It has narrower peaks and wider troughs than a perfect sine wave. This asymmetry is reflected in the frequency spectrum for this wave in Figure 6.16 in a later section (Section 6.6) where we do a more in-depth analysis of the spectra. Even though the signal is measured as close to the wave source as possible, it has

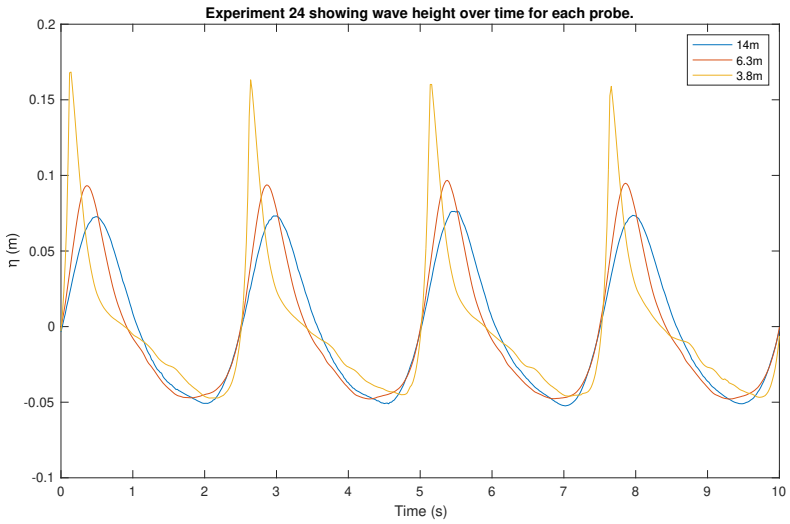


Figure 6.5: Time series of the water surface from experiment 24 for probe positions at $x = -14$ m, -6.3 m and -3.8 m. The series at -6.3 m corresponds to the mid-slope region

still undergone some nonlinear wave distortion. We know that the nonlinear features of Boussinesq waves are primarily caused by the relationship between the wave height and the water depth. In this case the water is shallow enough even at this deepest point for there to be an interaction between the wave and the floor of the tank, causing the nonlinear effects seen in the figure. If, for example, the wave source was out in the deep sea region where the ratio of the wave height to the water depth was much smaller, this effect would not be so prominent.

6.4.3 Mid-slope region

In Figure 6.5, we see that the waves have grown in height as they move from the paddle to a point near the centre of the wave tank, $x = -6.3$ m. We can see that the troughs of this signal are wider than at $x = -14$ m. At this point, no wave breaking has yet occurred as can be seen from the smooth wave profile. From Figure 6.7 we can see that the standard deviation of the wave height is relatively low at this position. From this we know that the wave height in this region is predictable and has a low variability.

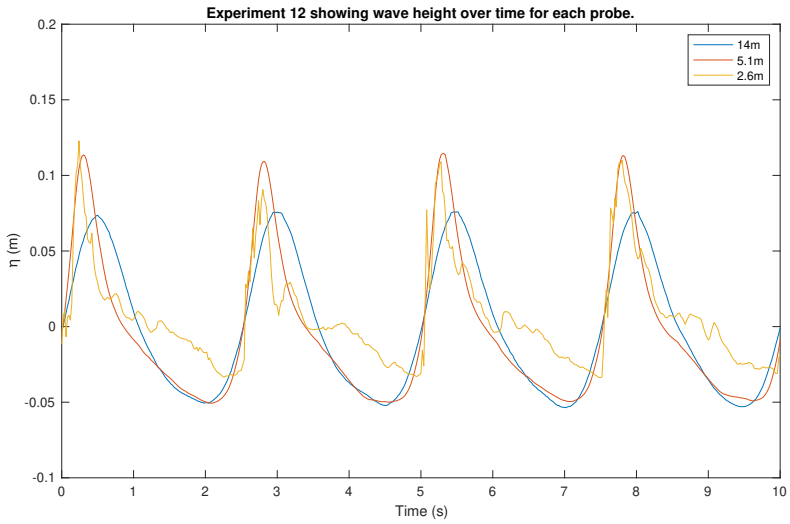


Figure 6.6: The time series from experiment 12 showing breaking waves on Probe 3 which is located at $x = -2.6$ m.

6.4.4 Breaking region

Below you can also see results in Figure 6.6 showing the waveform at $x = -2.6$ m. This time series shows a good example of breaking waves. The signal is far more chaotic than at $x = -6.3$ m and has lost most of its smooth beach-wave shape. This is a clear indication of wave breaking. Another characteristic of the breaking region is clear from Figure 6.7. That is the sudden increase in the standard deviation of the average wave height. In this region, the wave height is less predictable and varies more across each wave period. Another significant feature is that more of the wave appears to be above the zero line. This means that the water in this region has gained potential energy during this process.

6.4.5 Final wave height results

The reader will remember that 0 m represents the shoreline on the right of Figure 6.1 and distance is measured in metres from the shoreline. Figure 6.7 illustrates the average wave heights of waves moving in the same way, from the source on the left to the shoreline on the right. We can see that the average wave height increases predictably from $x = -6.3$ m to $x = -4$ m. At $x = -3.8$ m we see that the average wave height reaches a maximum

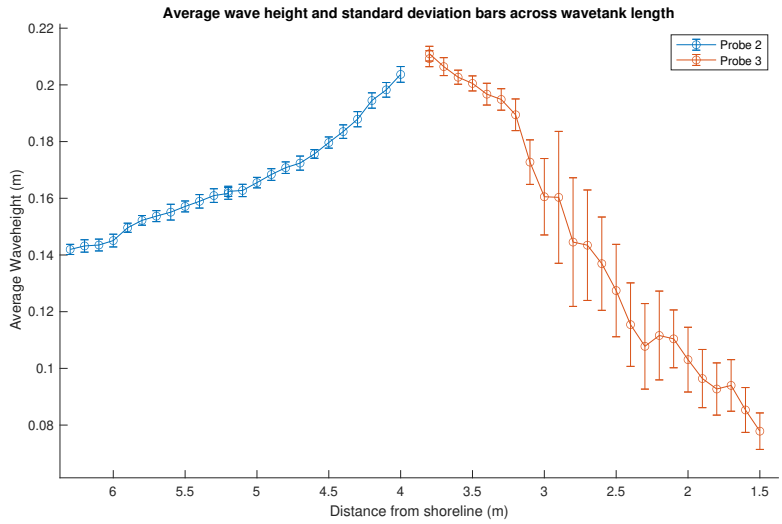


Figure 6.7: Average wave heights and standard deviations of all probes over the length of the tank.

and begins to decrease at a relatively predictable rate before the standard deviation shows any significant increases. It is at $x = -3.2$ m that we see the standard deviation of the waves suddenly increase and we see a corresponding increase in the rate of decay of the wave height. From here the average wave height falls off dramatically. As can be seen from the figure, the probes were well situated in these experiments in order to record data about the transition from the mid-slope region to the breaking region. It is clear from Figure 6.7 that the waves begin to break somewhere between $x = -3.8$ m and $x = -4.1$ m.

6.5 Phase velocity

In the beginning of this chapter we mentioned that the signals on the three probes were shifted out of phase. While we synchronised the signals in order to better display the waveforms, we can actually make use of this phase information to determine the phase velocity of the waves. In this section I will analyse the phase information of the waveforms and use this to compute the phase velocity of the waves as they progress up the slope. We will first develop the core concepts then show the final phase velocity algorithm at the end of this section.

We would like to work out the phase velocity of the waves at each point along the slope. If we had access to the entire waveform at every point in time, we could simply use a peak tracking algorithm to work out the velocity at each instant for each wave peak in the tank. However what we have is only a series of cross sections of that data. Each probe represents a cross section through time of the water surface at a specific position. While data was recorded at different positions along the tank, this means that the data from each experiment was recorded at a different time and there is no guarantee that the signals from one experiment will begin in the same phase as those from another experiment. In order to work out the phase velocity it helps to clarify the question we are asking. That is *By how much has the phase of a wave measured at some position changed since the time it was generated at the source?*. We happen to know the source signal for every experiment. We can therefore simply compute the phase difference between any signal and the source signal from the same experiment. Doing this for each signal in each experiment will give us a set of offsets, and working out the difference between these offsets will allow us to calculate the average velocity over a small range. We will now go through each step of this process, investigating the signals involved at each stage in the process.

Figure 6.8 shows a five second sub plot of the signals for $x = -3.6$ m, $x = -6$ m, and $x = -14$ m. We will first manually work out the phase difference between two sets of peaks then show how we use a cross correlation to do the same thing more accurately. This allows us to highlight a few key differences between these two ideas.

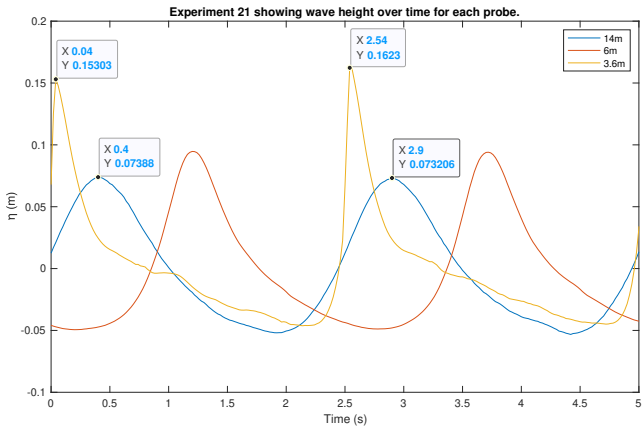


Figure 6.8: Time series corresponding to probes at $x = -14$ m, $x = -6$ m, and $x = -3.6$ m.

In Figure 6.8 we can see three signals with the peak values of two of the signals shown. To work out the phase difference between these two waves we could simply find the average difference between the peaks of the waves. This approach gives an average phase offset of 0.36 s between the signals at $x = -14$ m and $x = -3.6$ m. We will now compute the phase offset using the cross correlation. In order to compute the cross correlation of the two signals, each of length N samples, we first pad the ends with N extra zero samples to increase their length to $2N$ samples. This ensures that our cross correlation avoids making unwanted circular correlations. We then take the $2N$ point Fast Fourier Transform (FFT) of both signals and compute the frequency domain correlation function according to the following equation:

$$C_{(f)} = f_1 f_2^*, \tag{6.2}$$

where f_1 is the frequency spectrum of the signal at $x = -3.6$ m and and f_2^* is the complex conjugate of the frequency spectrum at $x = -14$ m.

Taking the real component of the inverse FFT (iFFT) of Equation 6.2 we get the correlation function which is shown in Figure 6.9 below. From this correlation function we can find the maximum correlation to be at the 31st index. Since we know each sample of the input signals corresponds to 20 ms we deduce that the phase offset between the two signals is 20 ms $\times$ 31 samples = 0.62 s. This result differs greatly from our manual calculation of 0.36 s above and is outside of a reasonable error range of about $\pm$20 ms. In this case the cross correlation we are computing is between two signals that are not purely of one frequency. Specifically, the signal at $x = -3.6$ m is made up of many harmonic frequencies and since the cross correlation is most heavily weighted by the fundamental frequency of the signal, it is more accurate to know that we are finding the difference mainly between the fundamental frequency components of the waves being compared since the peaks of the dominant frequency component of the signal are not necessarily in the same place as the peaks of the original wave. This is especially true for waves with more harmonic components. It is for this reason that we pass our signals through what is know as a perfect reconstruction filter upsampler before we do our cross correlations. The details of this filter are discussed in Section 6.5.1 below. This filter allows us to isolate a single frequency component of each wave as shown in Figure 6.10 where only the fundamental frequency of each wave is present. It can be seen from the figure that the peaks of these waves are closer than those of peaks of the original signal. In this context it is now a satisfactory approach to simply compare the peak positions. This difference can be clearly seen in Figure 6.12, where the fundamental frequency component of the time series

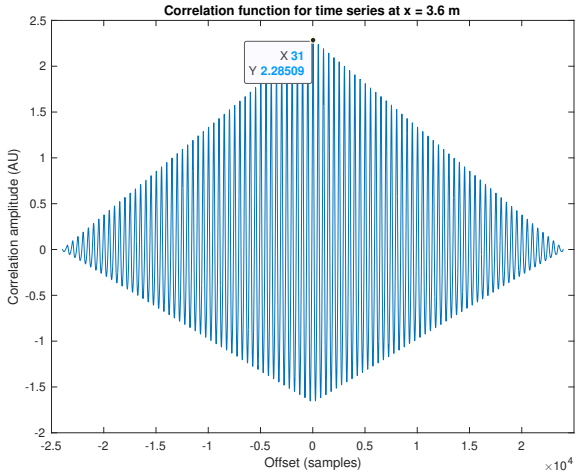


Figure 6.9: Cross correlation of time series at $x = -14$ m and $x = 3.6$ m with $4\times$ upsampling.

at $x = 3.8$ m is compared against the original unfiltered signal. As well as filtering the signals down to their fundamental frequency component of 0.4 Hz, the perfect reconstruction filter upsampler has also upscaled the signals in the figure to 4 times their original resolution (increasing their resolution from 20 ms to 5 ms). For reasons that will be discussed in Section 6.6, we have also used a Hann window function as well as zero padding in our cross correlation to reduce spectral leakage. Figure 6.11 shows the improved cross correlation function. Applying the Hann window function is a step that was not included in the analysis by Dr Govender and Dr Mukaro according to their paper (Mukaro et al., 2013). When computing the phase offset for Figure 6.10 by looking at the difference between the peaks (corresponding to the waves $x = -14$ m and $x = -3.6$ m) we get an average phase offset of 103 ms and when computing the cross correlation of the same filtered and upscaled signals we get cross correlation of 5 ms $\times$ 22 samples = 110 ms. This is of course a much more accurate result.

6.5.1 Perfect reconstruction filter upsampler

We will now discuss the perfect reconstruction filter upsampler technique that was used to preprocess signals before computing our cross correlation functions. It is a basic frequency domain filtering and resampling technique that allows us to reconstruct the original signal using only a specific subset of

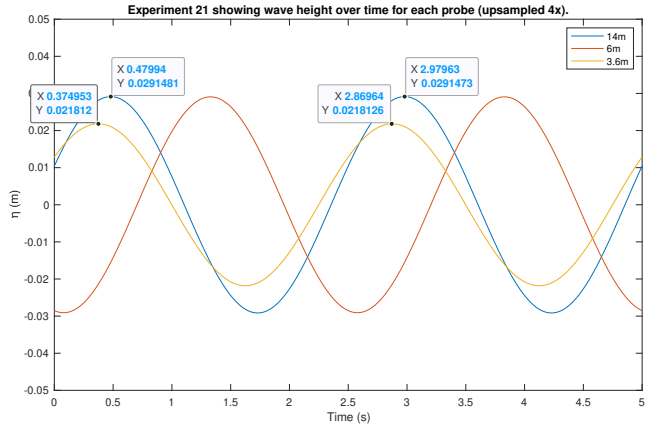


Figure 6.10: Filtered and 4× upsampled time series at $x = -14$ m, $x = -6$ m, and $x = -3.6$ m (showing fundamental frequency of the waves). Note that the amplitude values on the vertical axis are scaled by a factor of 4 compared to the original signal. This is a consequence of using MATLAB forward and backwards FFTs together with interpolation. This is accounted for in Algorithm 5.

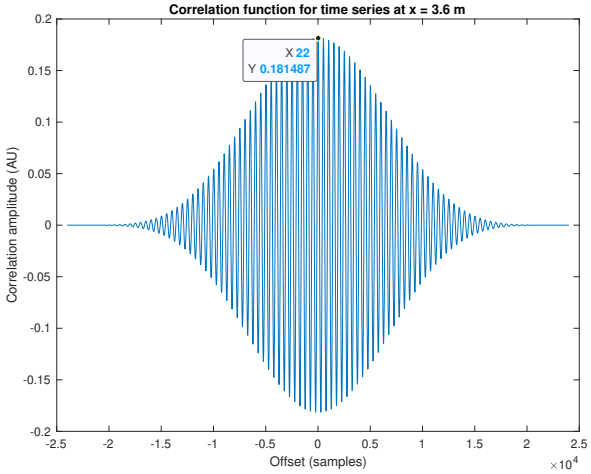


Figure 6.11: Cross correlation of fundamental frequency components of $x = -3.6$ m and $x = -14$ m with 4× upsampling and Hann windowing applied. The windowing can be see in the tapering of the correlation function.

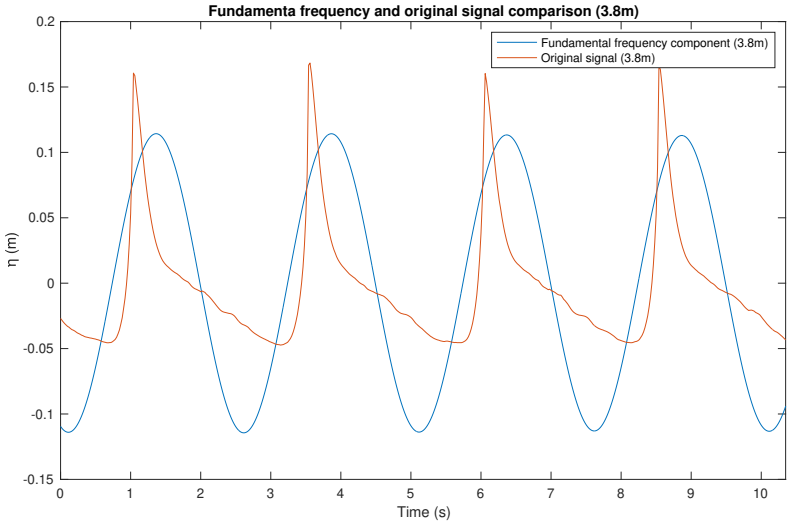


Figure 6.12: Comparison of original signal and its fundamental frequency component.

frequencies while increasing the sample resolution of the signal. It works by first taking some N-point FFT of the input signal, then in our case applying both a low pass and high pass filter to the frequency spectrum on either side of the fundamental frequency component of the signal. In our case this means simply setting every frequency bin to zero either side of the 0.4 Hz peak. Since we are using this to determine a cross correlation between waves that all have the same frequency, it allows us to effectively track the phase offset of waves even as the rest of their frequency spectrum changes over time. Once this is done, we then move on to resampling the signal. Since we are in the frequency domain already, we can simply split the full spectrum in the centre into two parts and insert a number of zeros proportional to the upsampling factor such $M/N = k$, where M is the number of output samples, N is the number of input samples, and k is the upsampling factor. Then we compute the M point inverse FFT to reconstruct the signal using only the fundamental frequency component of the signal at an upsampled sample rate of k. This process is analogous to time domain sinc function interpolation.

The number of zeros, Z, needed to pad the centre of the full spectrum FFT in order to upscale an N-point input signal by an upsample factor of k

is given by Equation 6.3:

$$Z = N\,(k-1)\,. \tag{6.3}$$

The pseudocode for this perfect reconstruction upsampler is shown in Algorithm 5. While this algorithm does not include a windowing step for addressing spectral leakage, I found that in this case it did not affect the final phase velocity calculations[4].

Algorithm 5: Pseudocode for perfect reconstruction filter

```
foreach signal S in experiment do
    Compute the FFT of the signal;
    Zero all frequency components outside of a desired range;
    Pad right side of FFT with zeros proportional to the upsampling
      factor;
    Scale magnitude of remaining frequency components based on
      upsample factor[a];
    Compute the iFFT;
end
```

[a] When filtering out frequencies from the spectrum we are also removing energy from the system. To compensate for this it can be convenient to amplify the resulting wave. This does not affect the final correlation results.

6.5.2 Final phase velocity calculations

So far in this chapter we have investigated how to work out the phase offset of each signal in our dataset relative to its source signal. We have discussed the problem of finding a reference point for the phase offset of each signal, the algorithm for computing the cross correlation of two signals. We investigated the discrepancy between the phase difference between the peaks of two signals and a more accurate definition of the phase difference between two signals. We investigated how a perfect reconstruction filter can help solve this problem and at the same time be used to upsample our signals for increased accuracy. Since we can now compute the phase difference between a signal and its reference signal, we can simply do this for each signal in our dataset, then use the difference between these offsets to compute the phase velocity at each point.

[4] An appropriate windowing function for this is the Hamming function since it tapers off to a non-zero value on both sides. This means it can be applied before doing the FFT to decrease spectral leakage, and when the reconstruction step is done using the iFFT, one can divide by the Hamming function again to reproduce the original non-windowed signal without any tapering on the ends.

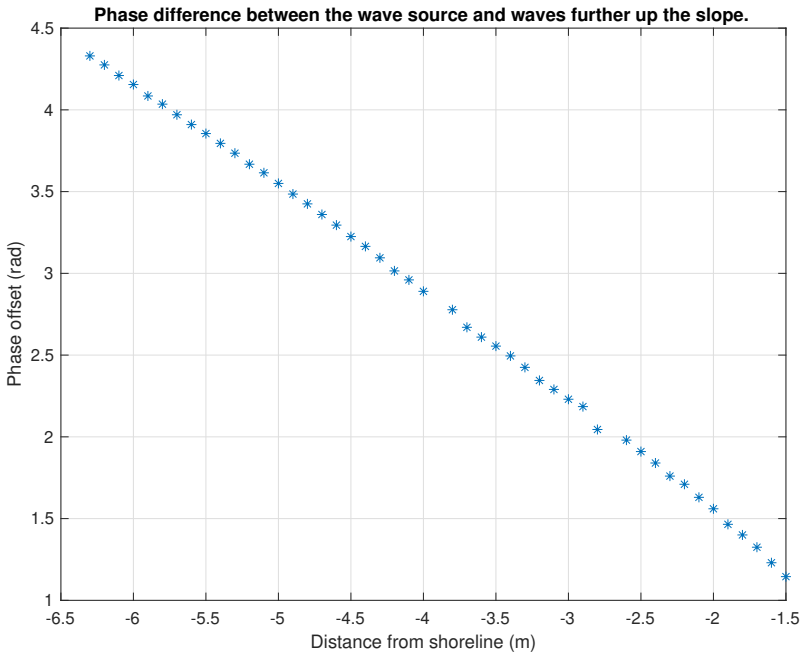


Figure 6.13: Phase differences

Figure 6.13 shows the phase offset for each signal captured along the length of the tank. The offsets for points where data was captured by more than one experiment have been averaged. The phase offset computed between two signals can only be within $\pm\pi$ rad even if the offset between the functions is larger in reality. The offsets in the figure have therefore been accounted for by adding a 2π rad shift to portions of the data. According to the figure, the phase offset of the waves decreases as the waves approach the shore. This may appear contradictory to the results of Mukaro et al. (2013), as the offsets they show increase as the waves approach the shore instead of decrease. However, this is actually a matter of interpreting the time shifts calculated by the cross correlation. The cross correlation measures the time between two nearest peaks in the two signals, and it may not be the two correct peaks that are compared. We can correct this by simply subtracting the measured time shifts between signals from one period.

We can now compute the phase velocity using Equation 6.4 below, where f is the fundamental frequency of the wave, Δx is the distance between the positions of two phase differences, and $\Delta\Phi$ is the change in phase difference

between the two points. This is given by

$$c = \frac{2\pi f \Delta x}{\Delta \Phi}. \tag{6.4}$$

When calculating Δx and $\Delta\Phi$, the centred difference was used. Once the phase velocity was computed at each point, a five-point moving mean was taken of this resulting data. This algorithm is shown in Algorithm 6. The final result is shown in Figure 6.14 along with the two theoretical curves, one derived from linear theory ($\sqrt{gh}$), and the other using the roller model concept ($1.3\sqrt{gh}$) of Schäffer, Madsen and Deigaard (1993). As can be seen from the figure, the experimental velocity matches that of linear theory very closely up until just before 4.2 m from the shoreline. At this point, the velocity begins to increase until it reaches a maximum of about 1.81 m/s at 2.9 m from the shoreline. During this increase it crosses the line predicted by the roller model at 3.4 m. After reaching a maximum, the velocity decreases again, following the roller model's curve. The sudden increase in wave velocity between $x = -4.2$ m and $x = -2.9$ m occurs in the same region that we know the waves begin to break. This suggests that the increase in velocity is due to the breaking of these waves in this transition region. This knowledge also separates the graph into three distinct regions: pre-breaking, breaking, and post-breaking regions. In the pre-breaking region, the waves seem to follow the curve derived from linear theory. After this the waves break and they then appear to follow the curve derived by the roller model. This seems logical as the roller model is intended to describe waves in the post breaking region. Another significant attribute of the breaking and post-breaking regions is the apparent increase in variation of the velocity.

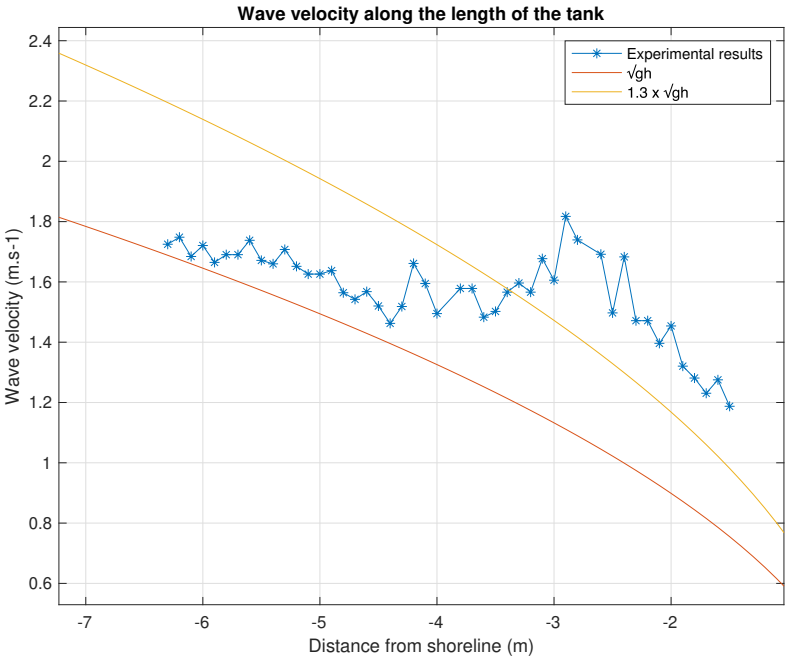


Figure 6.14: Phase velocity

Algorithm 6: Pseudocode for determining the phase velocity

```
foreach signal in experiment do
    - Filter and upsample signal using perfect reconstruction
      upsampler from Algorithm 5;
    - Apply Hann window function to signal;
    - Pad right sides of inputs to prevent circular correlations;
    - Compute the FFT of the padded data;
    - Compute correlation;
    - Compute the iFFT to get the correlation function;
    - Return index of max correlation;
    - Compute the time offset as dt / resampleFactor;
    - Adjust points that are off by +-pi rad;
    - Average duplicate offsets;
    - Compute the centered difference of the offsets;
    - Compute the 5 point moving mean;
end
```

6.5.3 Future optimisations

A natural optimisation is to combine the perfect reconstruction code with the cross correlation code and compute both quantities while still in the frequency domain when doing the perfect reconstruction filter. This minimises the number of times you need to transform the signals between the frequency and time domains. However, this was not done in my code for a number of reasons. Firstly, keeping these two operations separate allows them to be run independent of one another. This made my code far more readable and made analysis of the different stages of data processing far more accessible. This follows a simple separation of concerns and a standard functional programming principle. Secondly, the scale of the data involved meant that the value of this optimisation was simply not valid and even premature. With only about 75 signals to analyse, all processing was done in a matter of a few seconds. However, this optimisation is good to note if one wishes to make a highly optimised version of this type of algorithm.

In the next section I will show my analysis of the frequency spectra of the waves as they travel up the slope.

6.6 Frequency analysis

In this section we will investigate the spectral properties of the experimental beach wave data. We will investigate three main regions along the length of

the tank. The wave source, the near-breaking region and the breaking region. We will first look at the frequency spectrum of the source region to establish a reference for the rest of the results, then proceed to analyse the remaining regions. For each frequency spectrum, I have run a minimum-distance peak detection algorithm and used this data to identify the location of the harmonic peaks in the data. This information was then used to compute an exponential curve of best fit for the peaks. Note that the zeroth index in the FFT data was excluded from all peak detection and curve fitting analysis [5].

In Figure 6.16 we see the frequency spectrum for waves at $x = -14$ m (the wave source). The reader will remember that the wave generator was set to generate waves with a frequency of 0.4 Hz. This can be seen in the diagram with the largest peak at 0.4 Hz on the far left of the diagram. However, this is not the only frequency component that appears in the spectrum. As well as the fundamental frequency, we also see other prominent harmonic peaks followed by a number of small peaks that appear to have the same regular spacing. It is clear that the waves measured at the source are not purely sinusoidal, as seen previously, and this is proven by the fact that there is clearly more than one frequency component forming a significant portion of the spectrum. As discussed previously, this can be due to the nonlinear interaction of the water surface with the floor of the tank at the paddle due to the relatively shallow depth of the tank. As for the relationship between the magnitude of the different peaks in Figure 6.16, we can see that they fit an exponentially decaying curve very closely. From the fitted curve in the figure we can see there is very minimal difference between the exponential curve and the peaks of the frequency spectrum. The fitted curve equation is shown to be the following for waves at the source:

$$y = 7.2079e^{(-4.8768f)}, \tag{6.5}$$

where y is the normalised spectral amplitude and f is the frequency.

In Figure 6.18 we see the frequency spectrum for waves at $x = -6.0$ m. This is one of the points furthest from the shore besides the wave source for which we have data. At this point along the length of the tank, the waves are still in the non-breaking region. Here we can see the peak at 0.8 Hz has grown from a magnitude of about 0.25 to a magnitude of 0.425. The same pattern can be seen for other low frequency harmonic peaks in the spectrum.

[5] Non zero values at the 0 Hz index of a frequency spectrum indicate a constant offset to the signal. While this information is useful and is shown in the spectra, it is excluded from curve fitting. These curves represent the decay of energy from the fundamental component of the wave into non-zero frequency components of the waves.

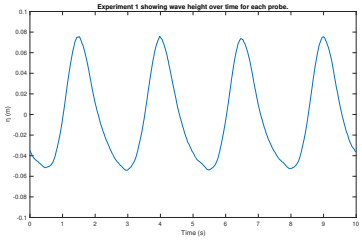


Figure 6.15: The time series at $x = -14$ m.

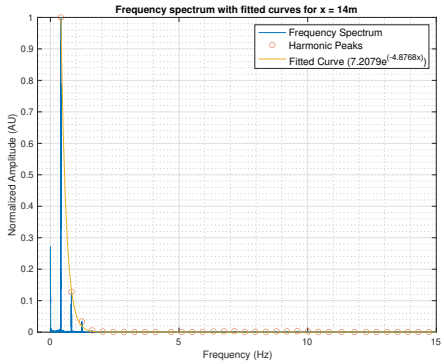


Figure 6.16: Frequency spectrum for $x = -14$ m with peak detections and curve fitted to these peaks.

It can be seen that energy is now distributed in a wider range of low frequency harmonic peaks than in Figure 6.16. This is evident in the changes to the equation of best fit given by Equation 6.6 for $x = -6$ m. Note that both the magnitude of the exponential and magnitude of the decay rate have both decreased. Notice that the peak at 0 Hz has also increased slightly. The peaks appear to still fit an exponential curve very closely according to the equation of best fit:

$$y = 2.3534e^{(-2.1383f)}. \tag{6.6}$$

We will now examine the frequency spectrum of waves that form part of the near-breaking and breaking regions, where the waves have reached their peak height and begin to collapse under their own weight. Figure 6.20 shows the frequency spectrum and fitted curve for $x = -3.8$ m from the shoreline. From Figure 6.7 we know that at this point the waves have already reached their peak wave height and are about to enter the most chaotic region known as the breaking region. As can be seen from the figure, the frequency spectrum at this point has become very interesting. The magnitude of the peaks in this figure oscillate along the frequency axis, creating six frequency bands. The frequency pattern that we see is due to the fact that in the breaking region the wave has a more saw-tooth shape. The spectrum of a saw-tooth signal has shape which is proportional to a sinc function squared. While the distribution of peak magnitudes follows a more complicated pattern compared to those seen previously, it does generally follow an exponentially decaying curve with the following equation:

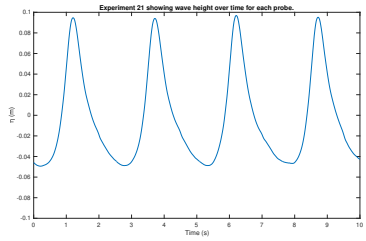


Figure 6.17: The time series at $x = -6$ m.

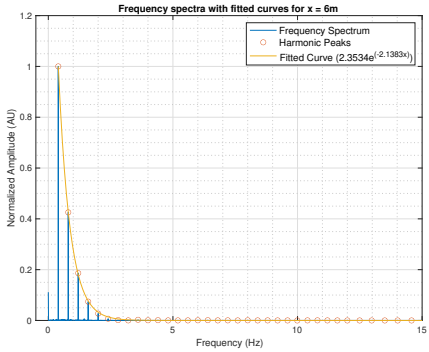


Figure 6.18: Frequency spectrum for $x = -6$ m with peak detections and a curve fitted to these peaks.

$$y = 1.6845e^{(-1.315f)}. \tag{6.7}$$

6.6.1 The relationship between Boussinesq waves and sawtooth waves

While the spectrum shown does correspond correctly to a sawtooth wave it is possible that some of its shape is influenced by spectral leakage. Spectral leakage tends to occur when the fundamental frequency of the input signal to the FFT does not fall exactly on a bin in the FFT result. In other words leakage occurs when the input signal does not contain an integral number of cycles in N samples, where N is the size of the FFT to be computed. Readers who are experienced with working with Fourier transforms may notice that the spectrum of the wave in the breaking region is similar to a sawtooth wave in the frequency domain. Equation 6.8 describes a reverse sawtooth wave made up of the sum of K sinusoids, each of frequency kf and amplitude $\frac{2a}{k\pi}$:

$$y\left(t\right) = \frac{2a}{\pi}\sum_{k=1}^{K}\left(-1\right)^{k}\frac{sin\left(2\pi kft\right)}{k}. \tag{6.8}$$

This similarity is due to the fact that as the Boussinesq waves become more deformed by nonlinear effects, the trailing side of each peak tends to drop off slowly while the rising edge of each wave tends to become increasingly steep, forming what begins to look like sawtooth waves. To demonstrate this point, a sawtooth wave and its Fourier transform are shown in Figures 6.21,

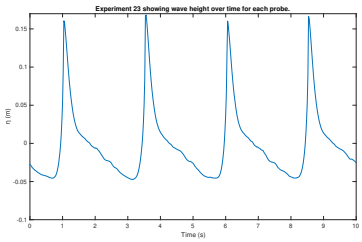


Figure 6.19: The time series at x = −3.8 m

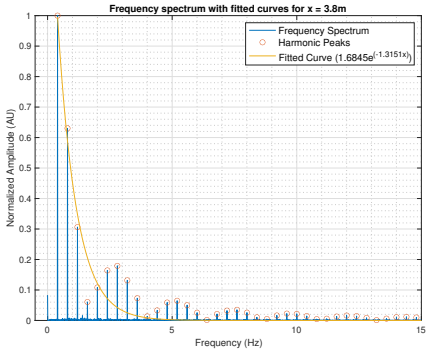


Figure 6.20: The frequency spectrum for the time series at $x = -3.8$ m with peak detections and a curve fitted to those peaks.

6.22, and 6.23 below. Interestingly, when constructing a sawtooth wave with the same parameters as the Boussinesq waves ($f = 0.4$ Hz, $dt = 0.02$ s, $t = 120$ s) we do not see the same frequency banding effect. Figure 6.22 shows a relatively smooth exponential decay of frequencies. However, if we vary the frequency of the sawtooth wave by only 0.22% from 0.4 Hz to 0.40089 Hz, we see the same banding effects present for our Boussinesq waves. For a sawtooth wave with a frequency of 0.40089 Hz, we get a frequency spectrum with many of the same properties to what we see at $x = -3.8$ m in Figure 6.20. Varying the frequency of the sawtooth wave I have noticed that we get smooth exponential decay of the spectral peaks with harmonic frequencies (that is frequencies that are multiples of 0.4 Hz). Figure 6.22 shows the FFT results of the saw-tooth wave using a signal frequency of 0.4 Hz, sample time of $dt = 20$ ms and an FFT length of $N = 6000$. While Figure 6.23 shows the same result but using a signal frequency of 0.40089 Hz, sample time of $dt = 20$ ms and a FFT of length $N = 6000$. Since both spectra is that of a saw-tooth waveform we know from theoretical analysis that we expect the spectra to have a sinc function squared profile. Thus the plot in Figure 6.22 appears to have a smearing of the spectra which is a sign of leakage.

Figures 6.24, 6.25, 6.26 and 6.27 show more examples of the very different spectra obtained at different points along the tank when not addressing spectral leakage. With this in mind, we can apply a few different techniques to decrease the amount of spectral leakage in our FFT results.

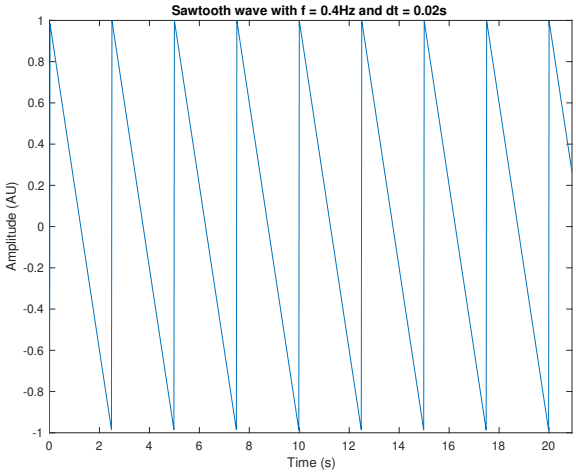


Figure 6.21: Sawtooth wave with $f = 0.4$Hz

6.6.2 Addressing spectral leakage

One technique for decreasing the amount of spectral leakage when working with FFTs is to apply a windowing function to the input signals for the FFT. There are various well studied window functions available including the Hann, Hamming, Flat Top, simple triangular, Blackman, and Riemann window functions to choose from, to name a few (Jai Krishna Gautam and Saxena, 1995). While they all have slightly different spectral properties, they are all used to taper the ends of the input signal to the FFT in order to reduce the amount of spectral leakage and amplify important features of the spectrum. I have chosen to use the Hann window function as it is well suited to waves that are made up of a sum of harmonic frequencies (Harris, 1978). While newer techniques have been developed that use hybrid windowing techniques that better capture the spectral properties of signals with sharp spectral peaks, I have chosen to use the classic Hann window for simplicity (Kallel, Hu and Kanoun, 2022).

Another technique for addressing spectral leakage is to pad the end of the input signal to the FFT with zeros. Normally the number of zeros is chosen such that the new length is a power of 2 since many FFT algorithms rely heavily on this for optimisations. Since spectral leakage is caused by doing the FFT of a signal with a non-integer number of wave lengths, we also ran our minimum distance zero crossing detection algorithm from Algorithm 4 on the input signals and trimmed both ends of the signals so that they

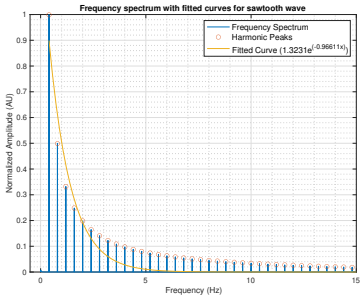


Figure 6.22: Frequency spectrum of sawtooth wave with $f = 0.4$Hz

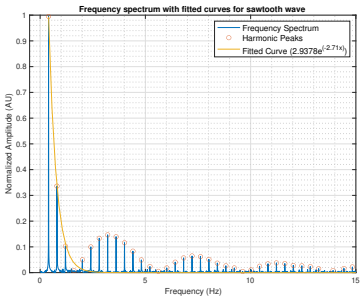


Figure 6.23: Frequency spectrum of sawtooth wave with $f = 0.40089$Hz

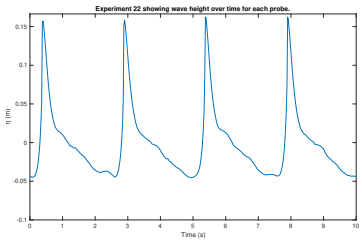


Figure 6.24: Time series of the wave at $x = -3.7$ m

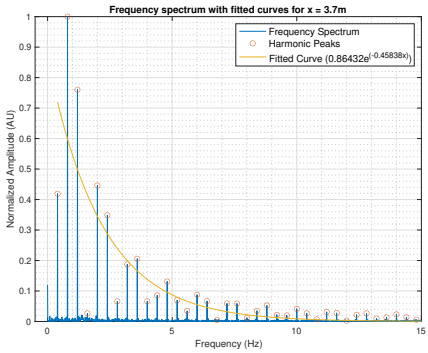


Figure 6.25: Frequency spectra of the time series at $x = -3.7$ m

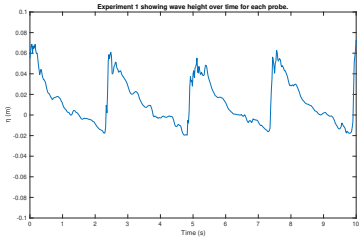


Figure 6.26: Time series of the wave at $x = -1.5$ m

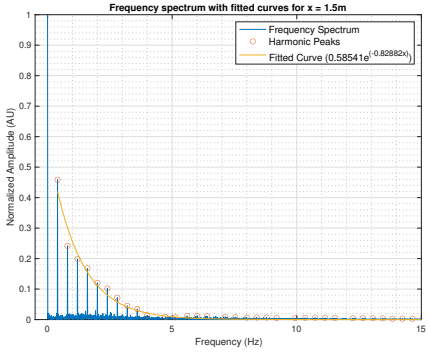


Figure 6.27: Frequency spectra of the time series at $x = -1.5$ m

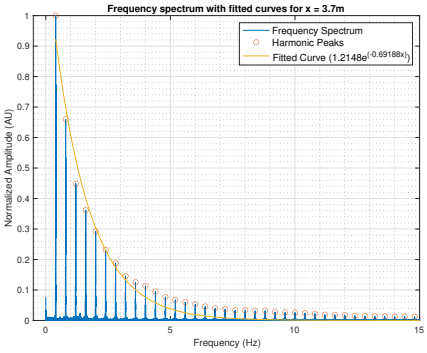


Figure 6.28: Frequency spectra of the waves at $x = -3.7$ m with the above mentioned techniques for reducing spectral leakage applied.

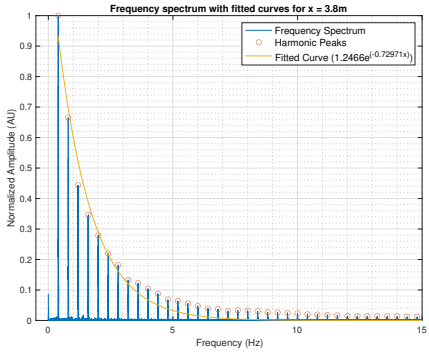


Figure 6.29: Frequency spectra of the waves at $x = -3.8$ m with the above mentioned techniques for reducing spectral leakage applied.

started and ended on the rising edge of the waves. We then combined all of these techniques by first cropping our input signals to an integer number of wavelengths, then multiplying this signal of sample length N by a Hann window of the same length N. We then pad this windowed signal with zeros, increasing its length to $4N$ samples and use this as the input to our FFT. With these improvements, we get the following results shown in Figures 6.28 and 6.29.

6.6.3 Analysis of the a and b coefficients

With the improved results from the previous section we can much more accurately fit a curve of best fit to the harmonic peaks of the frequency spectra. We can then do this for each frequency spectrum along the length of the tank. This fitted curve is represented by the simple exponential equation

$$y = a^{-bx}, \tag{6.9}$$

where a is the exponential amplitude, b is the decay rate of the harmonic frequencies, and x is the frequency axis. Plotting the constants, a and b from Equation 6.9 over the length of the tank gives us Figures 6.30 and 6.31. The curves in these figures have been smoothed using a 5 point moving mean. From Figure 6.31, we can see that at $x = -6.3$ m the decay rate of the harmonic frequencies was about -2.18. As we move along the length of the tank the decay rate increases linearly to its maximum of about -0.7

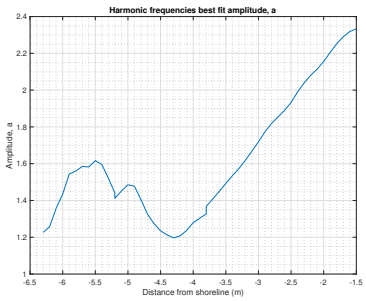


Figure 6.30: Best fit amplitude, a

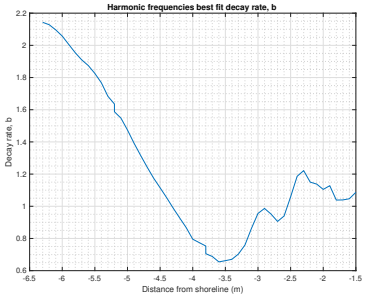


Figure 6.31: Best fit harmonic decay rate, b

at $x = -3.6$ m. This means that at this point the waves have their widest spread of harmonic frequency components. This appears to happen around the breaking point of the waves where the waves have just reached their maximum height. Because of the wide spread of harmonics this is also the point at which the original wave most resembles a sawtooth wave. After this point the decay rate decreases to around -1.05 and begins to oscillate. The curve for a in Figure 6.30 seems to follow the same but inverse pattern, decreasing at a constant rate from around 2.35 to a local minimum of 1.2 at 3.6 m from the shoreline. It then follows the same oscillation pattern as it increases again to about 1.6 then decreasing again to a new minimum of about 1.15.

If we plot the harmonic decay rate and average wave height along the length of the tank, we get Figure 6.32. This further illustrates the relationship between the average wave height and the range of harmonic frequencies contained in the waves. It is shown that the average wave height peaks around the same position as when the waves have the widest range of harmonic components and that this occurs near the breaking point of the waves. Figure 6.33 shows the rate of change of these two quantities over the length of the tank. This shows that a change in the average wave height at a specific position along the slope is associated with a corresponding change in the number of harmonic components in the wave.

6.6.4 Harmonic analysis

As mentioned previously, the 0 Hz peak in the frequency spectrum represents any offset that the signals have from the x axis. Plotting the unnormalised amplitude of this peak we get Figure 6.34. For reference, the average am-

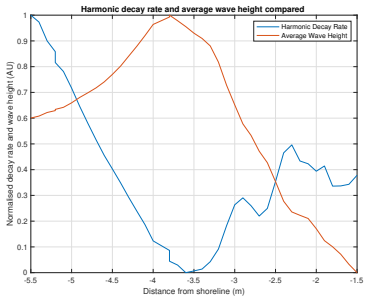


Figure 6.32: Plot of the normalised decay coefficient, b, and normalised wave height across the simulated tank. The wave height was normalised so that the peak wave height at the break point was unity, while the decay rate was normalised so that the peak decay rate at the source was unity.

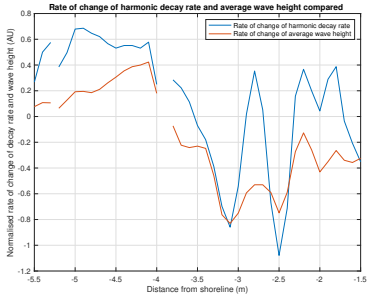


Figure 6.33: Change in the harmonic decay rate and wave height comparison.

plitude of the 0 Hz peak at $x = -14$ is 4.75[6]. In the figure we can see that the offset of the wave increases from this value to about 15 m just before the breaking point. At this point the 0 Hz peak drops to almost zero before increasing rapidly to a maximum of 41 at $x = -1.5$. Figure 6.35 shows how the amplitude of the fundamental frequency component of the wave changes over the length of the tank. Figures 6.36, 6.37, 6.38, and 6.39 show the unnormailsed harmonic peak amplitudes of four peaks along the length of the tank.

6.7 Summary

In this chapter a detailed analysis of the experimental data captured by Dr Govender and his PhD student (now Dr Mukaro) was undertaken. These include examining the shape of the wave at various positions along the tank, changes in wave height and phase speed as the waves move up the sloping beach. The wave height was found to increase steadily as the wave moves up the beach, until they became unstable and breaking followed thereafter.

[6]The spectra computed using the FFT in MATLAB needs to be scaled by a factor proportional to N. The spectra presented in this thesis have not been scaled. Therefore the units of the spectral components are unscaled and therefore represent relative numbers.

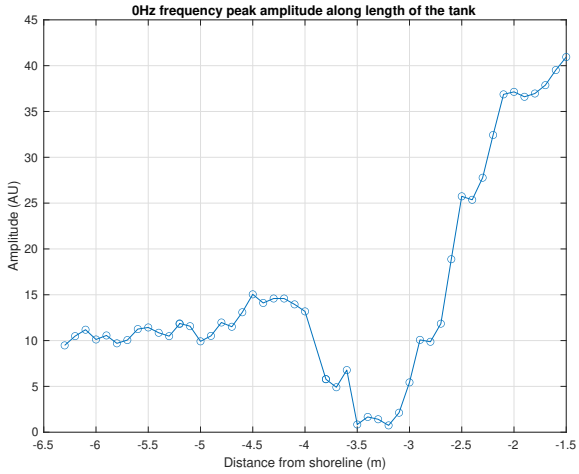


Figure 6.34: The unadjusted/uncorrected amplitude of the 0 Hz peak from the frequency spectra versus position.

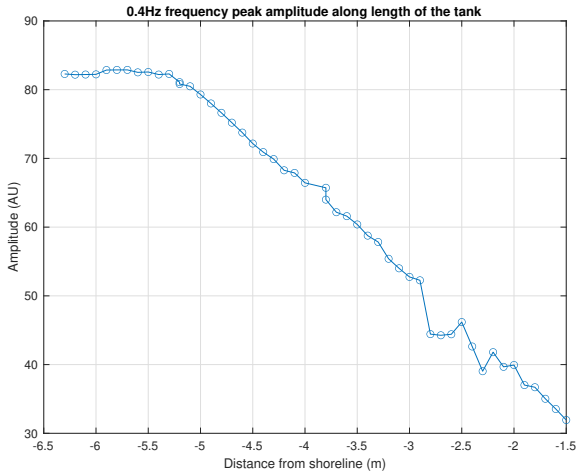


Figure 6.35: The unnormalised amplitude of the 0.4 Hz peak from the frequency spectra versus position.

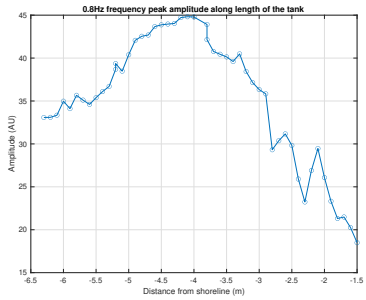


Figure 6.36: The amplitude of the 0.8 Hz peak across the length of the tank.

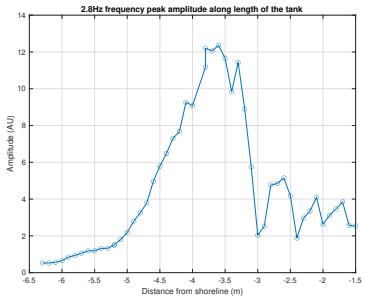


Figure 6.37: The amplitude of the 2.8 Hz peak across the length of the tank.

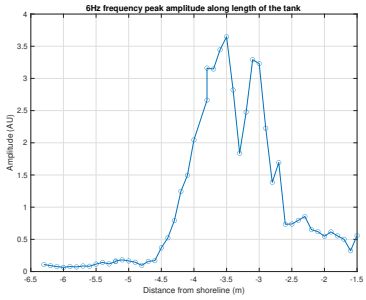


Figure 6.38: The amplitude of the 6 Hz peak across the length of the tank.

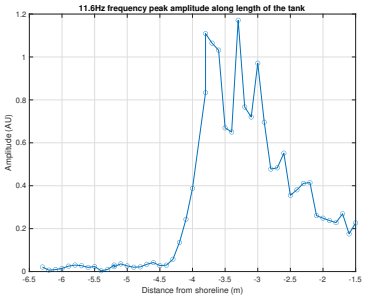


Figure 6.39: The amplitude of the 11.6 Hz peak across the length of the tank.

A detailed analysis of the spectral changes was also examined. In the next chapter we will analyse the data from my numerical simulations and compare with the experimental analysis from this chapter.

Chapter 7

Boussinesq simulation analysis

7.1 Introduction

In this chapter I will introduce the results of my simulations that were developed using the techniques described in Chapter 5. In the previous chapter we analysed experimental results from a real wave tank experiment and a similar analysis will be conducted using this simulated data. While I spent a great deal of time and effort attempting to write my own Boussinesq equation solver, it was unfortunately too unstable to be used for analysis and comparison with real world data. We will briefly discuss my MATLAB simulations (first in 2D then followed by a 1D simulation), discuss their limitations (where they become unstable), then proceed with the simulation of using the Fortran based numerical solver called FUNWAVE-TVD developed by Wei and Kirby (1995, 1998) as a replacement. These FUNWAVE-TVD results will then be compared with those of the previous chapter. As many of the algorithms for analysing the data were discussed in Chapter 6, this chapter will deal mainly with the presentation and discussion of the simulation results.

7.2 MATLAB simulation results

In Chapter 5 we investigated the theory behind the fully nonlinear Boussinesq model developed by the Wei and Kirby group. By studying the papers written by Wei and Kirby between 1995 and 1998 shown in Wei and Kirby (1995); Wei et al. (1995), as well as the PhD thesis of Long (2006*a*), I derived the equations necessary to implement this model in MATLAB. This was implemented without the use of any external MATLAB packages. The results of my simulations are shown in the next subsection starting with some validation tests.

7.2.1 Validation testing

Once my simulation was written, I began recreating some of the results from the 1995 paper by Wei and Kirby (1995). I started with the evolution of an initial Gaussian distribution within a rectangular basin, then moved on to the evolution of a solitary wavefront over a flat bottom.

The initial Gaussian distribution is described by Equation 7.1:

$$n_0(x, y) = A_0 e^{-2\left((x-3.75)^2+(y-3.75)^2\right)}, \tag{7.1}$$

where A_0 is the amplitude, and the centre of the peak of the wave is set to be in the centre of the 7.5 m by 7.5 m enclosure. The boundary conditions are given by the standard reflective boundary conditions defined in Chapter 5. The initial wave amplitude is $A_0 = 0.045$ m and the water depth at rest is 0.45 m. The discretisation variables are as follows: $dx = dy = 0.075$ m, and $dt = 0.05$ s. It is also worth remembering that a low pass filter is applied every 50 iterations. All four boundaries of the simulation used reflective boundary conditions.

In Figures 7.1, and 7.2, I show the stable results of my simulation at $t = 1$ s in a 3D view as well as a 1D cross section. When viewing the contour plot of these results in Figure 7.3, we can see they match up with the simulations by Wei and Kirby (1995).

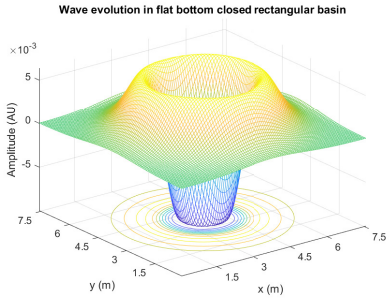


Figure 7.1: A 3D view of the surface elevation, of an initial Gaussian distribution that was allowed to evolved for 1 second.

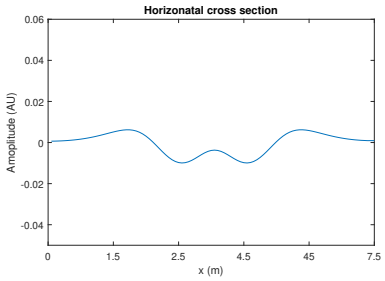


Figure 7.2: The 1D cross section at $y = 3.75$ m through the 2D space at $t = 1$ s.

As can be seen from the figures, we can see that the central peak of the Gaussian waveform has dropped downwards then sprung back upwards during the 1 second time frame. However, I found that if I increased the simulation time and let the waves hit the boundaries my simulation would

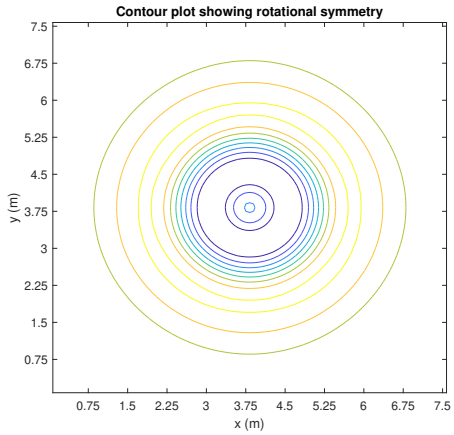


Figure 7.3: A colour contour plot of the 3D picture shown in Figure 7.1. These results match those of Wei and Kirby (Wei and Kirby, 1995).

become unstable. This is shown in Figures 7.4 and 7.5. High frequency oscillations can be seen at all corner boundaries, particularly the north-east and south-east boundaries[1]. Although I took great care to implement the simulation as correctly, it is possible that there was a programming error in the calculation of the corner boundary conditions.

With these results, I moved on to simulating waves moving up a constant slope beach in order to recreate the results of Mukaro et al. (2013). This required having a sloped floor profile, as well as using a rectangular simulation space and more complex boundary conditions. This is where I hit the first major issues with my simulations. To avoid the boundary condition issues from the previous simulation I increased the size of the simulation domain by moving the north and east boundaries further away from the wave source. This ensured that as few waves as possible reached these unstable boundaries.

7.2.2 Sloped floor profile validation

The beach waves simulation has a number of properties that make it a more complex simulation than the Gaussian evolution example of the previous example. This includes the introduction of a wave maker boundary condition on the west boundary (the vertical boundary on the left in Figure 7.3), the addition of a sloped floor profile, as well as the existence of a breaking region in the simulation domain. Using my 2D MATLAB simulation I started by

[1]The top and bottom of the page is considered to be north and south, respectively.

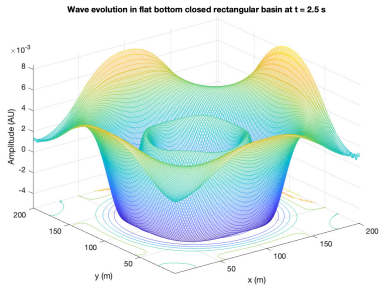


Figure 7.4: A 3D view of the Gaussian initial distribution at $t = 2.5$ s.

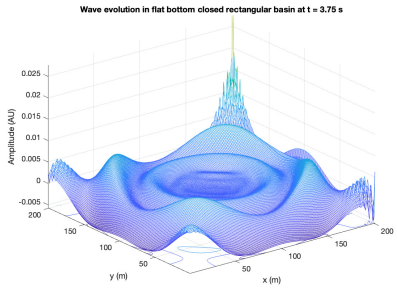


Figure 7.5: A 3D view of the Gaussian initial distribution at $t = 3.75$ s.

creating a plane hyperbolic secant wave initial condition. The wave then decays into two smaller waves, one moving towards the west boundary (before reflecting from this boundary) and one towards the shoreline in the east. This is shown in Figures 7.6.

While this simple simulation is relatively stable, unfortunately the addition of a wavemaker boundary caused the simulations to become very unstable and therefore unusable. While my MATLAB simulations were stable for small time periods, I found that they became unstable at the north and east boundaries of the simulation space for more complex scenes. Additionally, we were consistently seeing that the waveforms were not evolving as in the experimental setup. For this analysis, it unfortunately made these simulations unusable for analysis and comparison purposes.

I tried many things to address this including rewriting my entire simulation in a more object oriented manner in order to make debugging the simulation more manageable. I also investigated and implemented more advanced filtering techniques such as the truly two dimensional Shapiro filters of Falissard (Falissard, 2013). Unfortunately this did not improve the stability of my simulations. After this investigation I decided to rather write a one dimensional version of the simulation to simplify the problem further. Some results from this 1D simulation are shown below in Figures 7.7, 7.8, 7.9, and 7.10 below in order to illustrate the quality of the results from a typical simulation. In this 1D simulation I found that I could get a working wavemaker boundary to be more stable than in my 2D simulations. However, for my wavemaker boundary I found that I could only specify an initial wave velocity that equated to $\frac{1}{4}\sqrt{gh}$ instead of the the desired $\sqrt{gh}$. Faster moving waves would simply cause the simulation to blow up. The results

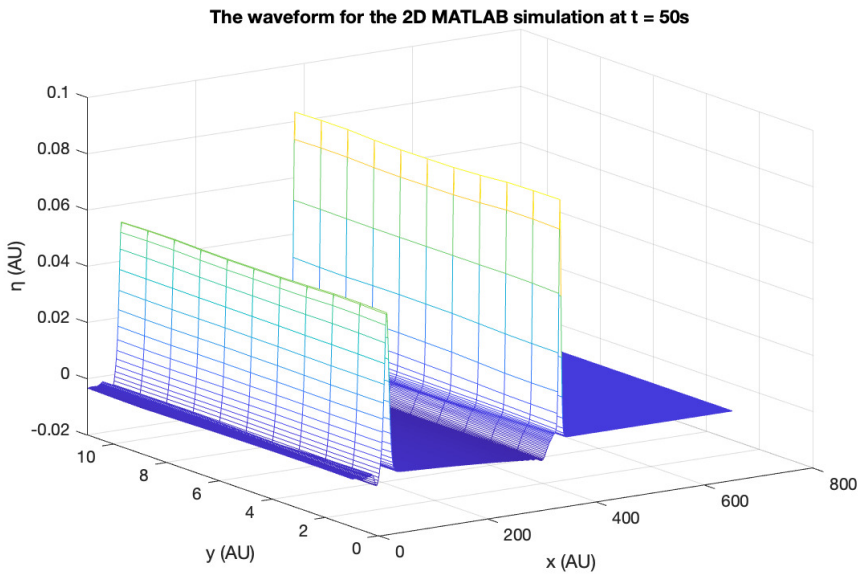


Figure 7.6: The 2D *sech* plane wave that has decayed into two smaller waves. At this point, the westward wave has reflected off the west wall, and the eastern wave has increased in height and changed shape slightly.

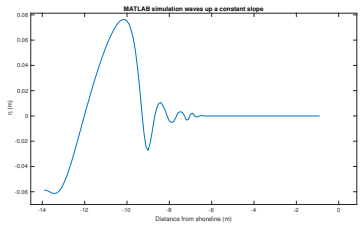


Figure 7.7: The waveform over the slope generated by my 1D MATLAB simulations at $t = 1$ s.

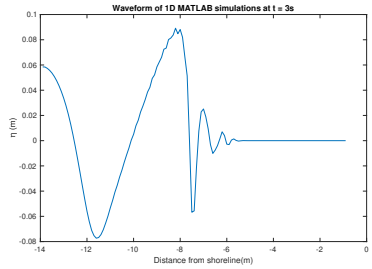


Figure 7.8: The waveform over the slope generated by my 1D MATLAB simulations at $t = 3$ s.

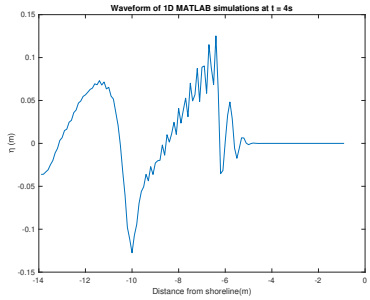


Figure 7.9: The waveform over the slope generated by my 1D MATLAB simulations at $t = 4$ s.

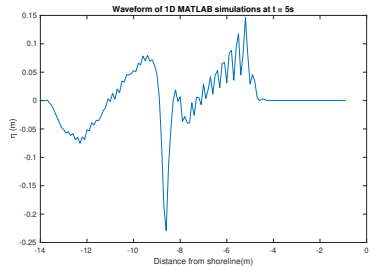


Figure 7.10: The waveform over the slope generated by my 1D MATLAB simulations at $t = 5$ s.

below are as close as I could get to recreating the physical experiments by Mukaro et al. (2013) in 1D.

From these simulations it appears that the waves could propagate all the way to just before the expected breaking region without exploding. Interestingly, these waves appear to pick up high frequency oscillations fairly close to the FUNWAVE-TVD results breakpoint as will be seen later in this chapter. However, the shape of the waves appears to not exactly match what is expected. Although we see the leading edge of the waves steepen and grow in height, and the trailing edge flattens out, the peak is not sharp enough to resemble the waves in the experimental data. It is also obvious that my MATLAB simulations were not stable in the wave breaking region. In this case the addition of a sponge layer over the entire breaking region may have been worth investigation in order to get some usable results across the pre-breaking region.

However, due to the instability of my MATLAB simulation and the time commitment that would be involved in debugging it and making it useful enough for comparisons with experimental data, I decided to put my own numerical simulation to rest. As its replacement I decided to start using a brilliant, well known Boussinesq solver called FUNWAVE-TVD that will allow us to more successfully simulated the experimental setup from the previous chapter. The FUNWAVE-TVD program is based on the discretisation scheme discussed in Chapter 5. This will be the central focus of the following section.

7.3 FUNWAVE-TVD results

In this section we will discuss what FUNWAVE-TVD is and how I configured it to replicate the physical experiment by Mukaro et al. (2013). We will then analyse the simulation results in detail.

FUNWAVE-TVD is a software package written in FORTRAN that is used for modelling ocean wave phenomena such as deep sea waves, tsunamis, ship wakes, and beach wave shoaling and even wave breaking and is used extensively in coastal engineering for harbour design. The original version of FUNWAVE used a fully nonlinear Boussinesq equation solver developed by Wei and Kirby (1995). This is the same model used in my MATLAB simulations. However, it is now called FUNWAVE-TVD because it uses a Total Variational Diminishing version of the fully nonlinear Boussinesq wave model developed by Shi et al. (2012).

7.3.1 Configuration

FUNWAVE-TVD configuration is done using variables in a *.txt* config file. The following configuration was used to replicate the experimental setup of Mukaro et al. (2013). The meaning of the variables in the configurations files are explained in the configuration script using comments below, as well as in the discussion in the next paragraph.

```
! INPUT FILE FOR FUNWAVE_TVD
! --------------------DEPTH-------------------------------------
DEPTH_TYPE = SLOPE
DEPTH_FLAT = 0.618 % water depth in the flat section near the paddle
SLP = 0.05   % slope of the beach
Xslp = 10.5    % distance from the left boundary to the start of the slope
```

```
! ------------------DIMENSION-----------------------------
! global grid dimension
Mglob = 229 % number of x discretisation points
Nglob = 3    % number of y discretisation points

!-----TIME-----
TOTAL_TIME = 120.0
PLOT_INTV = 0.02

! -----------------GRID----------------------------------
DX = 0.1
DY = 1.0

!-----WAVEMAKER-----
WAVEMAKER = WK_REG
DEP_WK = 0.618 % wave maker depth
Xc_WK = 7.0 % distance from left x boundary to the wave maker
Yc_WK = 0.0
Tperiod = 2.5   % generated wave period
AMP_WK = 0.06  % generated wave amplitude
Delta_WK = 3.0 % degree of wave nonlinearity

! ---------------- PERIODIC BOUNDARY CONDITION ---------
! South-North periodic boundary condition
PERIODIC = F

!-----SPONGE LAYER-----
DIFFUSION_SPONGE = F
FRICTION_SPONGE = T
DIRECT_SPONGE = T
Csp = 0.0
CDsponge = 1.0
Sponge_west_width = 5.0      ! this line
Sponge_east_width = 1.4
Sponge_south_width = 0.0
Sponge_north_width = 0.0


! ---------------PHYSICS-------------------------------
! parameters to control type of equations
!---------------Friction-----------------------------
```

```
Cd = 0.0

! ----------------NUMERICS-----------------------------
CFL = 0.5
FroudeCap = 3.0

! --------------WET-DRY--------------------------------
! MinDepth for wetting-drying
MinDepth=0.01

! -------------- BREAKING ----------------------------
VISCOSITY_BREAKING = T
Cbrk1 = 0.65
Cbrk2 = 0.35

! -----------------OUTPUT-------------------------------
ETA = T  % output the waveform at each iteration to a .txt file
```

On the website for FUNWAVE-TVD, a base configuration file is provided with all necessary configuration needed for running a basic simulation. What is shown above are the configuration parameters relevant to the surface wave simulation. For more information regarding the configuration of FUNWAVE-TVD as well as the base configuration file, please see the definition of parameters page on the FUNWAVE-TVD website at the following link: *https://fengyanshi.github.io/build/html/index.html*.

Up until now we have defined all measurements on the x axis relative to the shoreline. However, FUNWAVE-TVD expects values to be defined from the left boundary where the left boundary is at $x = 0$ m. This configuration file specifies that we would like to simulate a wavetank with a flat bottom at a depth of 0.618 m (DEPTH_FLAT) for the first 5.5 m (Xslp) from the left boundary, followed by a constant slope with a value of 0.05 (SLP) or 1:20. The value for Δx (DX) was chosen to be 0.1 m and Δy (DY) was set at 1.0 m as it is recommended that DY is much larger than DX for the 1D surface waves simulation. Mglob is then used to define 229 points along the x axis and Nglob defines the required 3 points on the y axis. The simulation is then configured to save a file with the entire water surface waveform every 0.02 s (PLOT_INTV) for a time period of 120 s (TOTAL_TIME). The wavemaker was then set to be at 2.0 m from the left boundary at a water depth of 0.618 m generating waves with a period of 2.5 s (Tperiod) and an amplitude of 0.06 m (AMP_WK) or wave height of 0.12 m. Delta_WK is a constant

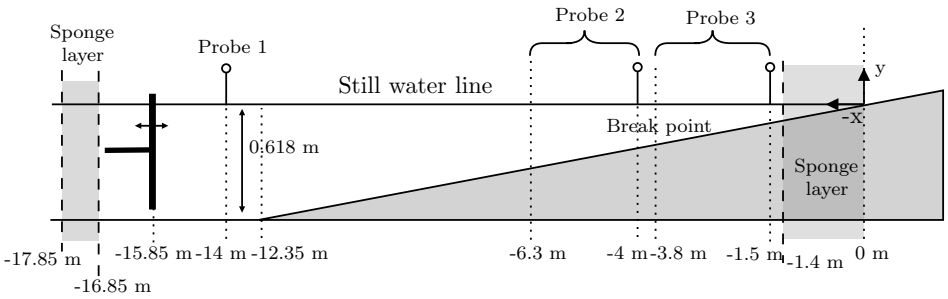


Figure 7.11: FUNWAVE-TVD simulated water tank layout described by the FUNWAVE-TVD configuration file. This setup is exactly the same as the experimental setup shown in the previous chapter in Figure 6.1 except that the simulated setup has a sponge layer on the left side for absorbing unwanted waves that move to the left from the wave source. A sponge layer is also added from $x = -1.4$ m to shoreline to reduce the intensity of shoreline reflections.

that determines the nonlinearity of the waves that is found by trial and error. Finally we have a sponge layer on the left boundary from $x = 0$ m to $x = 1$ m defined by CDsponge and Sponge_west_width, as well as a sponge at the shoreline positioned at $x = 1.4$ m from the shoreline just after our last probe that is at $x = -1.5$ m (see in Figure 7.11). This sponge layer was added after finding that the simulated waves were reflecting too strongly from the shoreline and travelling back towards the source which interfered with our analysis later in the chapter. The remaining parameters are simply the default physics based parameters for the surface wave example provided with the software package.

In the following section we will discuss the processing of the simulation data as well as its analysis.

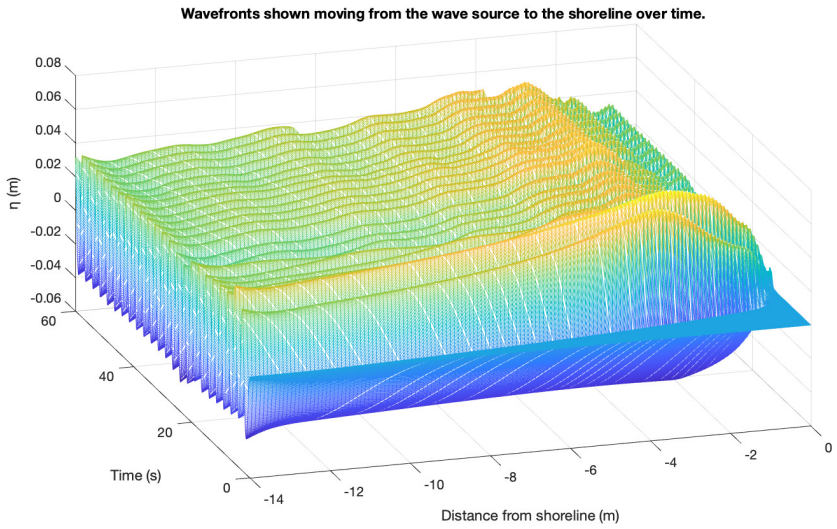


Figure 7.12: The full simulated waveform showing waves moving from the oscillating wave maker source at $x = -14$ m up the slope to the shoreline between the times of $t = 0$ s and $t = 60$ s.

7.3.2 Simulation results

When run with the configuration file from the previous section, FUNWAVE-TVD generates 6000 text files each containing 229 data points with a spacing of 0.1 m along the tank. Slices of this data are then taken through time at the exact same locations that were used in the real experimental setup, where each slice represents a probe placed in the wavetank measuring the wave height at a specific location for the duration of the experiment. After this stage of processing we end up with 25 text files in the same format as the experimental setup, where each file contains four columns (one time column, and three probe data columns). This approach allowed me to feed my simulation results into the exact same MATLAB code as for the experimental analysis.

FUNWAVE-TVD was successful at generating stable results and the full waveform between $t = 0$ s and $t = 60$ s is shown below in Figure 7.12. In Figure 7.13 below, a top down view of the same wavefronts is shown. In this figure it can be seen that the velocity of the waves decreases as they approach the shoreline. This is shown by the fact that the curve traced by the waves slopes slightly upwards. The phase velocity of these waves will be analysed in the same manner as the previous chapter in Section 7.4.

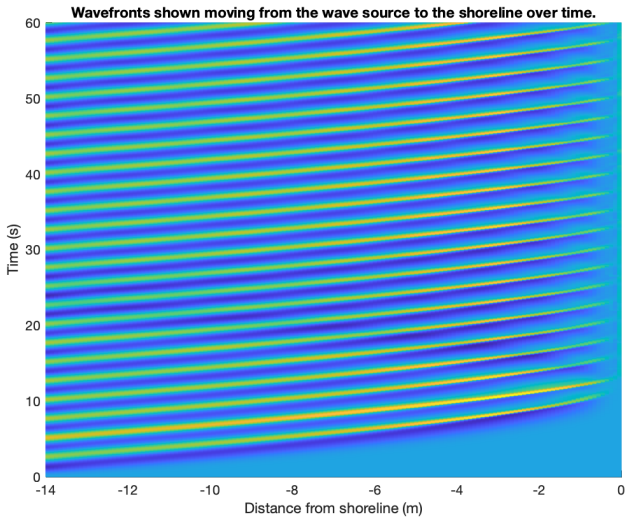


Figure 7.13: Top down view of the simulated wavefronts approaching the shoreline between the times of $t = 0$ s and $t = 60$ s.

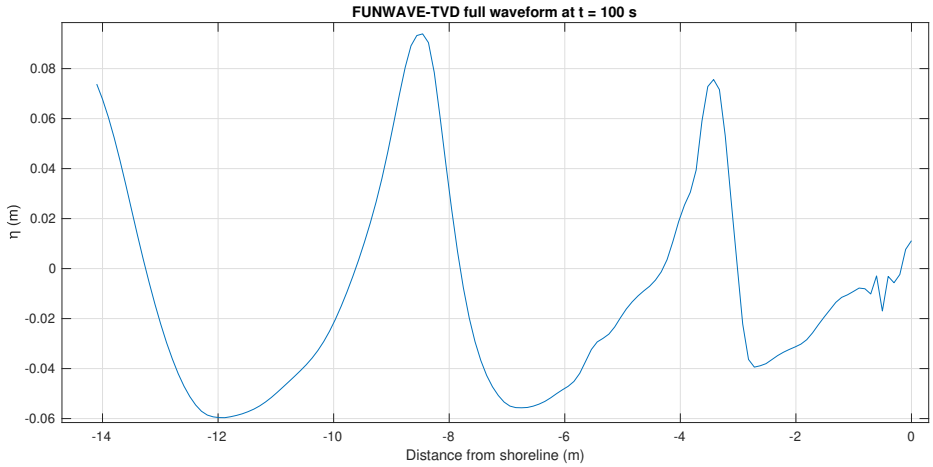


Figure 7.14: A snapshot of the simulated waveform at $t = 100$ s.

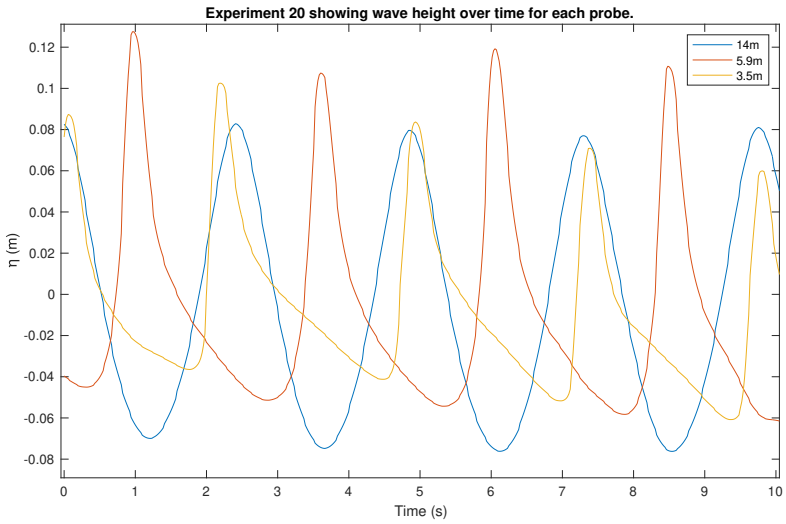


Figure 7.15: Initial data from simulated experiment 20 showing out of phase signals for time series captured at $x = -14$ m, $x = -5.9$ m, and -3.5 m. Where η is the instantaneous wave height.

Figure 7.14 shows a snapshot of the water surface height across the length of the simulated tank. In this figure the wave source is at $x = -16$ m and Probe 1 is at $x = -14$ m. Figure 7.15 shows the simulated results of the time series of surface elevation for $x = 14$ m, 5.9 m and 3.5 m, corresponding to experiment 20, followed by Figure 7.16 which shows the same signals after being passed through the phase adjustment algorithm (Algorithm 3) from the previous chapter. These signals are comparable to the physical experiment results in Figures 6.3. However one can see in Figure 7.16 that the peak positions after phase synchronisation across the three probes appear to be in relatively the same position. In the previous chapter we saw in Figure 6.3 that peaks of waves closer to the shoreline were generally offset to the left of the source wave peaks. This is likely due to the distribution of harmonic peaks in the simulated waves.

From Figure 7.15, we can see that the source time series in blue at $x = -14$ m appear to match those of the physical experiment, having a slightly asymmetrical sine wave shape, and a slightly pointier peak than trough. The red time series at $x = -5.9$ m show an increase in the peak height, as well as a further widening of the trailing edges of the waves and a steepening of the leading edges of the waves. The yellow curve at $x = -3.5$ m shows the

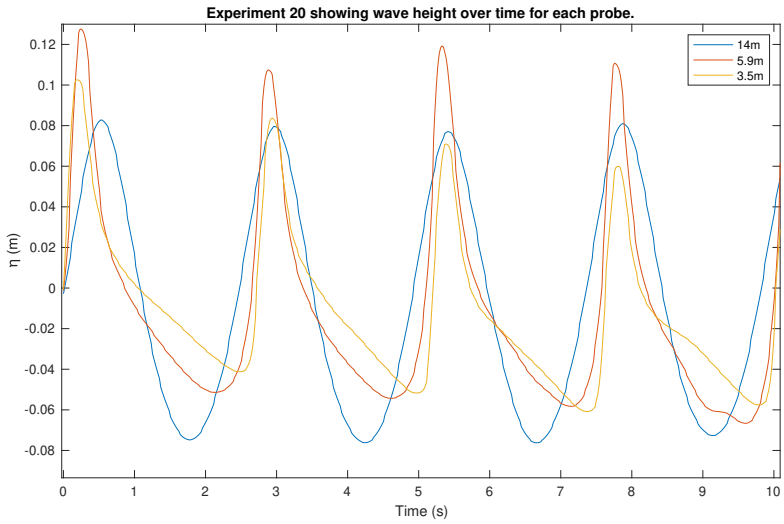


Figure 7.16: Phase-synchronised time series of experiment 20 showing $x = -14$ m, $x = -5.9$ m, and $x = -3.5$ m.

same effect and also stabilises later in the time series. As can be seen, the mean height of the wave in the yellow trace waves is greater than the other time series in the figure and has a lower peak height than the wave in the red trace. This is because it is measured in middle of the breaking region where the wave height decreases. In Figure 6.3 in the previous chapter we saw a much more chaotic signal at $x = -3.5$ m than we see here. While the model does support wave breaking, this signal would be highly dependent on the resolution of the simulation and even so, the simulation might not model the same intricacies of water splashing that is seen in real life, therefore producing a smoother output with nonetheless a good representation of the wave profile. Figures 7.17, 7.19, 7.20, and 7.21 show four simulated experiments in order to better convey the shapes of the simulated waves at different points along the simulated tank.

In Section 7.3.3 we will discuss the results of the wave height analysis of this simulated data.

7.3.3 Wave height analysis

As in Chapter 6, an analysis of the wave heights of the simulated time series data was conducted. The results are shown in Figure 7.18. Since we are simulating the waves in these experiments, we can sample from anywhere along

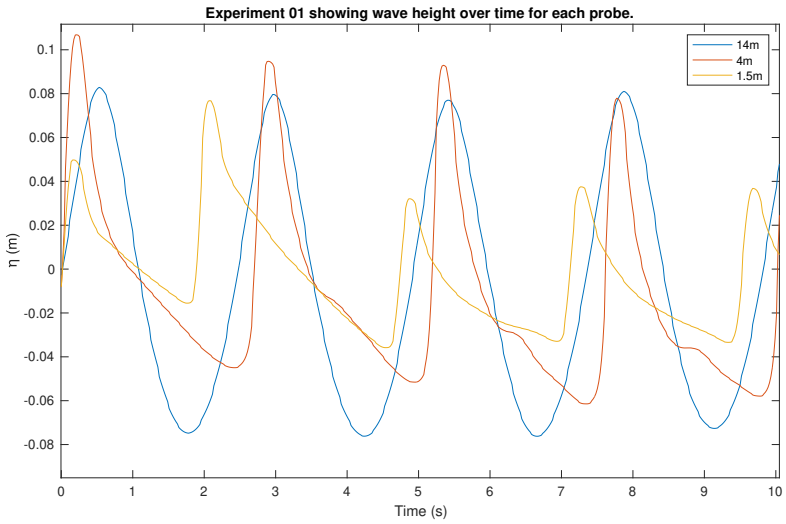


Figure 7.17: Phase-synchronised time series of experiment 20 showing $x = -14$ m, $x = -4$ m, and $x = -1.5$ m.

the simulated wavetank. So as well as sampling at all the same points as in the experimental setup we also sample all the way up to just before the wave generator. The simulated domain now goes from $x = -12.9$ m to $x = -1.5$ m in intervals of 0.1 m. We can see in Figure 7.18 that the average wave height across the tank increases almost linearly until it reaches a maximum average height at $x = -5.5$ m of 0.175 m. When compared to the experimental results we note the following: we can see that the peak average wave height is reached earlier on the slope than in the experimental results. We can also see that the maximum that was reached in the simulation is also less than the experiments. After this point the average wave height then decreases rapidly down to a minimum of 0.0775 m at $x = -1.5$ m. This is comparable to the minimum of 0.08 m in the experimental analysis. During this sudden decrease we can also see two upward oscillations at $x = -5.1$ m and $x = 3.8$ m. We can also see that the standard deviation in the average wave height also increases near and after the break point while the average wave height remains relatively constant in the entire pre-breaking region. This is consistent with the experimental results in the breaking region. However, the increase in standard deviation just before the breaking region was something that was not present in the experimental results.

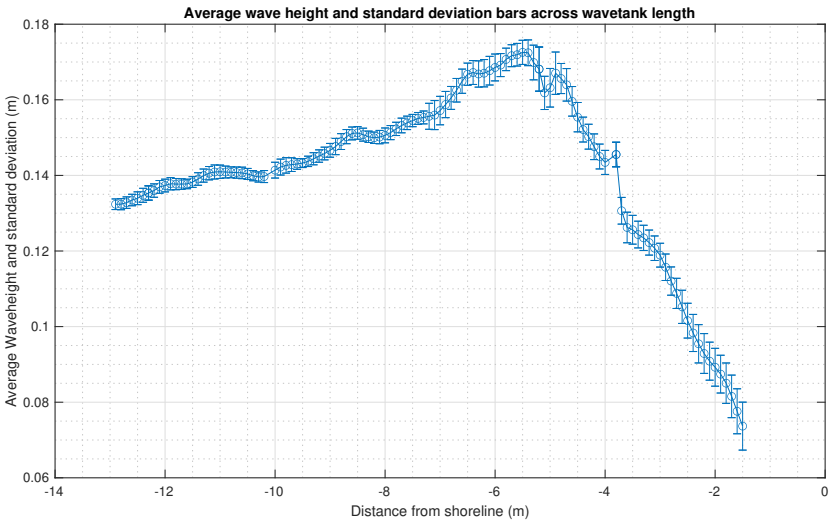


Figure 7.18: Average wave height and standard deviations of all time series across all simulated experiments.

7.4 Phase velocity

In this section we will analyse the phase velocity of the simulated waves in the same manner as in the previous chapter. In Figure 7.22 we can see the phase velocity across the entire length of the tank plotted along with two theoretical curves for the beach wave velocity in red and yellow. We can notice a few key things about the simulated wave velocity. Firstly, we notice that the phase velocity in the pre-breaking region mostly falls between the two theoretical curves. Secondly, the velocity at the break point of $x = -5$ m suddenly increases. This is the same phenomenon seen in the experimental analysis. The velocity after this point oscillates up and down until the break point. In this region the velocity no longer follows the theoretical curves. We can compare these observations to Figure 7.23 which shows the phase velocity of simulated waves when the sponge layer between $x = -1.4$ m and the shoreline is removed. The reader will remember from the start of this chapter that the shoreline sponge layer was added to reduce the amount of interference of waves reflected from the shoreline. We can see how these reflected waves affect the phase velocity in the pre-breaking region. In this region we see a large, steady oscillation in the phase velocity of the waves. However if we look at the breaking region, we see a steep increase in the wave velocity at the break point. After this point the phase velocity continues to

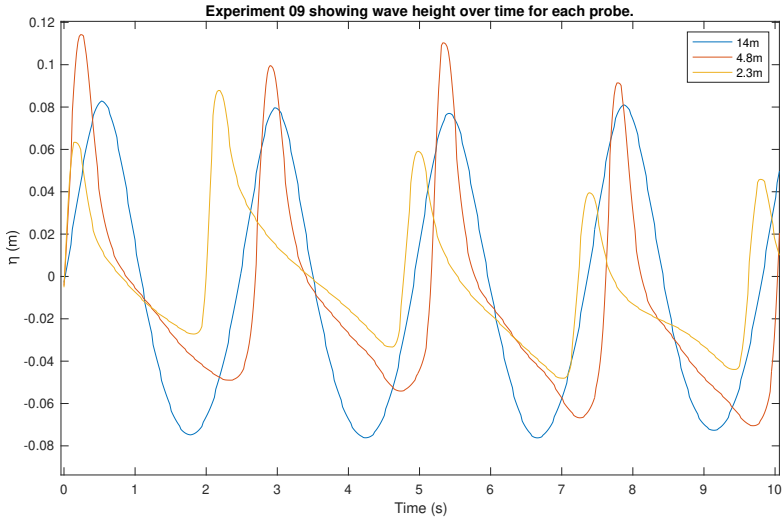


Figure 7.19: Phase-synchronised time series of experiment 20 showing $x = 14$ m, $x = -4.8$ m, and $x = -2.3$ m.

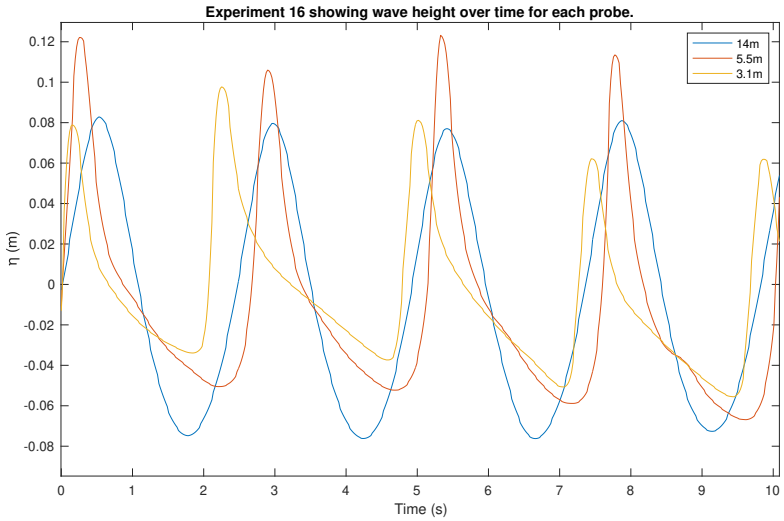


Figure 7.20: Phase-synchronised time series of experiment 20 showing $x = -14$ m, $x = -5.5$ m, and $x = -3.1$ m.

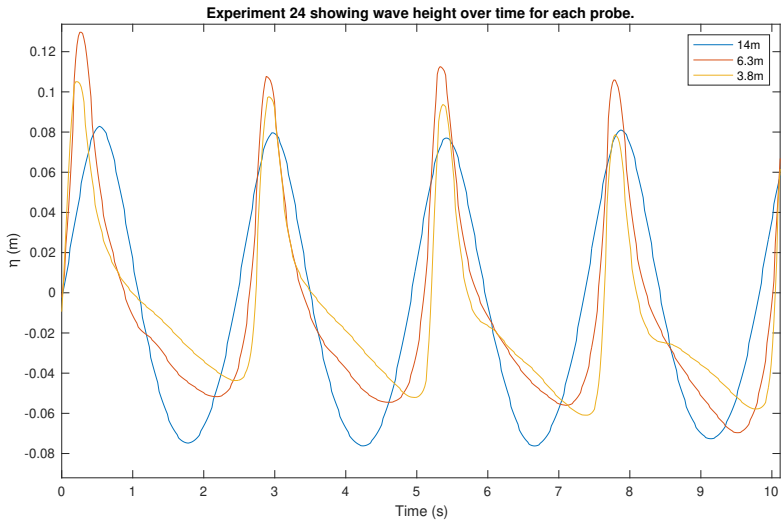


Figure 7.21: Phase-synchronised time series of experiment 20 showing $x = -14$ m, $x = 6.3$ m, and $x = -3.8$ m.

oscillate but does seem to follow the theoretical curves slightly better than when a shoreline sponge layer is used.

In Figures 7.22 and 7.23 the phase velocity of the waves is higher than the experimental results. While the simulation results that were obtained with a shoreline sponge layer approximately fit between the two theoretical curves, and show a sudden increase in wave velocity at the break point, which is similar to the experimental results, the phase velocity after the break point does not appear to match experiment. Without a shoreline sponge layer, the phase velocity in the breaking region appears to follow a more similar shape to the experimental results. However in both cases, the magnitude of the phase velocity is larger than that of the experiment, with peak velocities of 2.2 m/s and 2.5 m/s just after the break point in Figures 7.22 and 7.23 respectively, whereas it reached a value of 1.81 m/s at this point in the experimental results.

7.5 Frequency Analysis

In this section we analyse the frequency spectra of the time series generated by FUNWAVE-TVD. As in the previous chapter, we will examine the spectra at the wave source, near-breaking region, and the breaking region. The

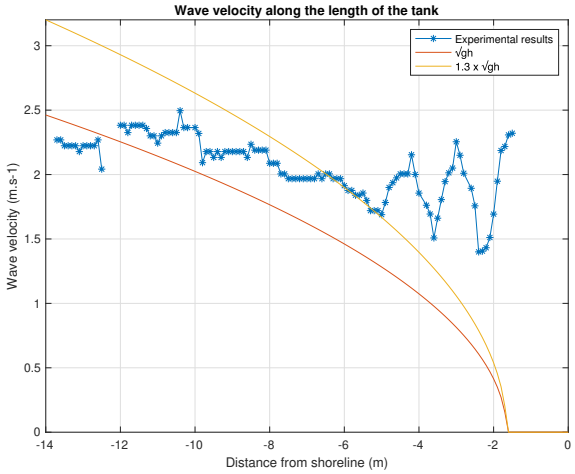


Figure 7.22: The phase velocity of the simulated waves across the length of the tank with a sponge layer between $x = -1.4$ m and the shoreline.

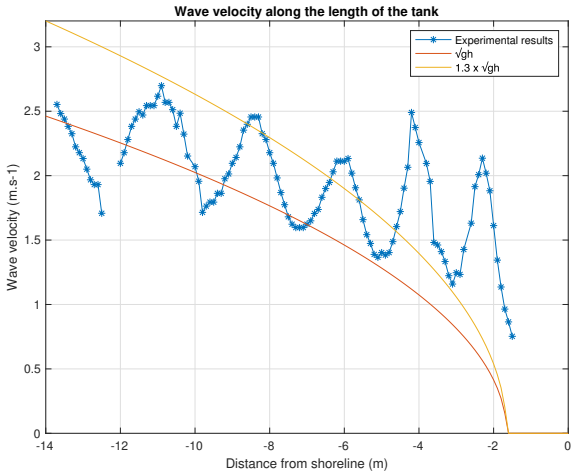


Figure 7.23: The phase velocity of the simulated waves across the length of the tank with no sponge layer near the shoreline.

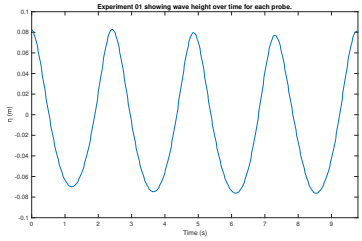


Figure 7.24: The simulated time series at $x = -14$ m.

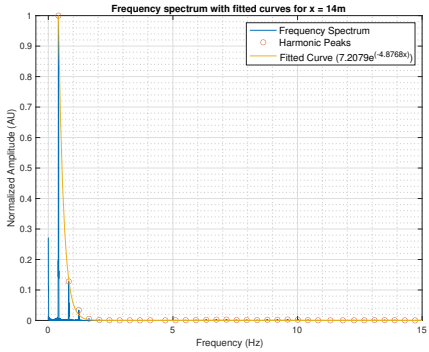


Figure 7.25: The frequency spectrum for $x = -14$ m with peak detections and curve fitted to these peaks.

figures shown all examine the exact same points in the wave tank as in the previous chapter ($x = -14$ m, $x = -6$ m, $x = -3.8$ m, $x = -3.7$ m, and $x = -1.5$ m). The same minimum distance peak detection algorithm from the experimental analysis was run on this simulated data. Exponential curves with equation $y = ae^{-bf}$ have also been fitted in the same manner where y is the fitted curve amplitude, f is the frequency and a and b are constants.

In Figure 7.25 we see the frequency spectrum for waves from Figure 7.24 at $x = -14$ m (the wave source). The reader will remember that the time period of the signal was configured as 2.5 s (i.e. a frequency of 0.4 Hz). This can be seen in the diagram with the largest peak at 0.4 Hz on the far left of the diagram. In the previous section we commented on the fact that waves at the wave source ($x = -14$ m) are not perfectly sinusoidal, with their peaks appearing pointier than their troughs as shown in Figure 7.24. This is reflected in the accompanying frequency spectra in Figure 7.25 by the existence of the harmonic peaks in the spectra. We can also see a 0 Hz peak indicating a slight offset to the signal. The curve fitted to the peaks is described by Equation 7.2:

$$y = 7.2079e^{(-4.8768f)}, \tag{7.2}$$

where y is the normalised spectral amplitude and f is the frequency. The coefficients of this fitted exponential curve very closely match those of the physical experimental results in Equation 6.5 from the previous chapter ($y = 4.0694e^{(-3.50544f)}$).

In Figure 7.27 we see the frequency spectrum for waves from Figure 7.26

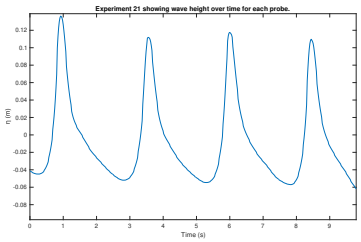


Figure 7.26: The simulated time series at $x = -6$ m.

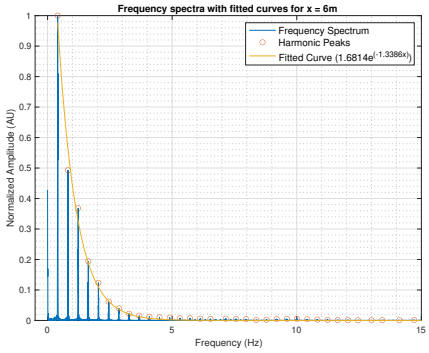


Figure 7.27: The frequency spectrum for $x = -6$ m with peak detections and curve fitted to these peaks.

at $x = -6.0$ m. This is one of the points furthest from the shore besides the wave source for which we have data. At this point along the length of the tank, the waves are still in the near-breaking region. Here we can see the peak at 0.8 Hz has grown from a magnitude of about 0.25 to a magnitude of 0.425. The same pattern can be seen for other low frequency harmonic peaks in the spectrum. It can be seen that energy is now distributed in a wider range of low frequency harmonic peaks than in Figure 7.25. This is evident in the changes to the equation of best fit given by Equation 7.3 for $x = -6$ m given by

$$y = 1.6814e^{(-1.3386f)}. \tag{7.3}$$

Note that both the magnitude of the exponential and magnitude of the decay rate have both decreased. Notice that the peak at 0 Hz has also increased slightly. The peaks appear to still fit an exponential curve very closely.

From Figure 7.29 we get the following equation of best fit shown in Equation 7.4 given by

$$y = 1.4872e^{(-0.9341f)}. \tag{7.4}$$

From Equations 7.2, 7.3, and 7.4, we can see that the amplitude coefficient of these equations appears to decrease as we measure closer to the shoreline. We notice that the decay rate constant b is also decreasing indicating a widening in the range of harmonic components present in the waves. More precisely, for a given change in the decay rate constant b in our best fit equations, the constant a describes by how much the intensity of the FFT peaks in the

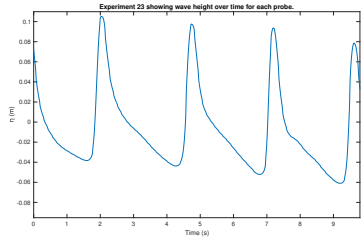


Figure 7.28: The simulated time series at $x = -3.8$ m.

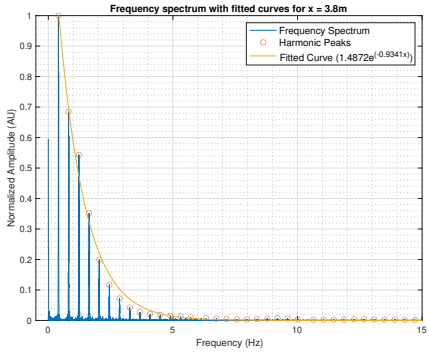


Figure 7.29: The frequency spectrum for $x = -3.8$ m with peak detections and curve fitted to these peaks.

spectrum will change. We notice that both a and b appear to be decreasing in magnitude as waves approach the shoreline. This indicates that the range of harmonic frequencies that are appearing in the waves as they progress along the slope is increasing and at the same time, the intensity of these peaks is decreasing. This relation between the two constants is related to the fact that the total energy in the system should remain relatively constant. This trend will be further investigated in Section 7.5.2 on Harmonic analysis.

Figures 7.29, 7.31, and 7.33, show the spectra for the waves in Figures 7.28, 7.30, and 7.32 respectively described by their well defined exponentially decaying harmonic peaks. It can be seen that a wider spectral envelope is present for waves at $x = -3.7$ m than at $x = -1.5$ m from the shoreline.

In Section 7.5.1 we analyse how the relative amplitude of the harmonic peaks changes over the course of the tank.

7.5.1 Analysis of the a and b coefficients

Fitting an exponential curve to each frequency spectrum along the length of the tank allows us to track how the energy contained in the wave is distributed amongst its different harmonic frequencies. This fitted curve is represented by the simple exponential equation

$$y = a^{-bf}, \tag{7.5}$$

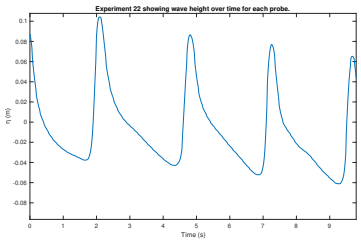


Figure 7.30: The simulated time series at $x = -3.7$ m.

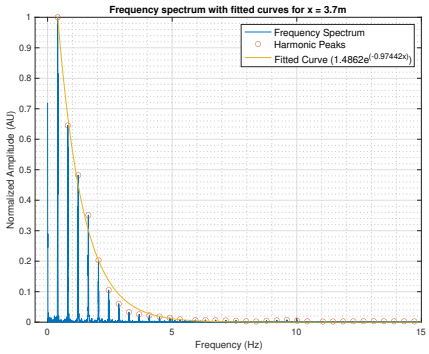


Figure 7.31: The frequency spectrum for $x = -3.7$ m with peak detections and curve fitted to these peaks.

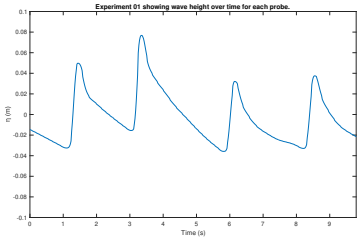


Figure 7.32: The simulated time series at $x = -1.5$ m.

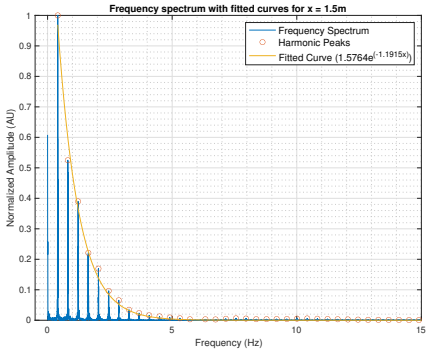


Figure 7.33: The frequency spectrum for $x = -1.5$ m with peak detections and curve fitted to these peaks.

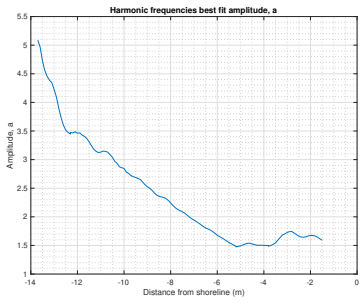


Figure 7.34: Best fit amplitude, a

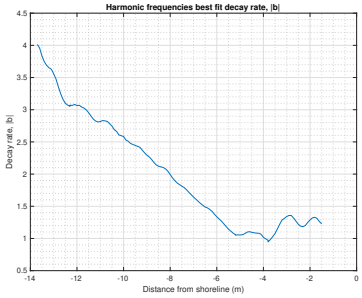


Figure 7.35: Best fit harmonic decay rate, b

where a is the exponential amplitude, b is the decay rate of the harmonic frequencies, and f is the frequency axis.

Plotting the constants, a and b from Equation 7.5 over the length of the tank gives us Figures 7.34 and 7.35 below. The curves in these figures have been smoothed using a 5 point moving mean. In Figure 7.34 we see that the best fit exponential curve amplitude a, starts at 5.1 at $x = -13.8$ m. It then decreases down to a minimum value of 1.5 just after the break point at $x = -5.2$ m. The value of a then begins to oscillate and increase slightly for the remainder of the slope. This curve appears to be directly related to the following curve in Figure 7.35 of the value of b, the decay rate. This curve starts at a value of 4.1 at $x = -13.8$ m and follows the same pattern of reaching a minimum value around 1 just after the break point before increasing and oscillating upwards. As previously mentioned, the decrease of the decay rate b corresponds to a widening of the exponential curve and an increase in the intensity and number of harmonic components present in the waves. This decrease in b is accompanied by a decrease in a which has the opposite effect. These two opposing curve adjustments correspond to keeping the total energy in the spectra relatively constant. The point $x = -5$ m is also the point at which the original wave most resembles a sawtooth wave, because at this point the waves have the widest spread of harmonics. It is in this region that the waves have the highest average wave height as we saw in Figure 7.18.

If we plot the harmonic decay rate and average wave height along the length of the tank, we get Figure 7.38 below. This further illustrates the relationship between the average wave height and the range of harmonic frequencies contained in the waves. It is shown that the average wave height peaks around the same position as when the waves have the widest range

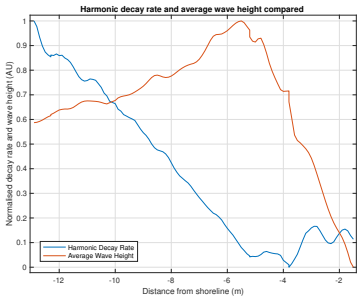


Figure 7.36: Plot of the normalised decay coefficient, b, and normalised wave height across the simulated tank. The wave height was normalised so that the peak wave height at the break point was unity, while the decay rate was normalised so that the peak decay rate at the source was unity.

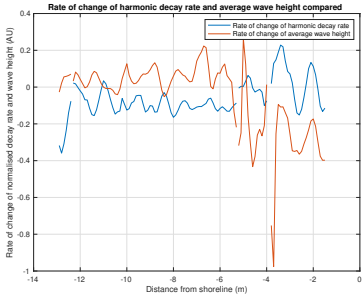


Figure 7.37: Change in the harmonic decay rate and wave height comparison.

of harmonic components and that this occurs near the breaking point of the waves. We see that at the point of maximum average wave height, we have the some of the highest spreads of harmonic components in the waves. After the break point, the decay rate b remains high, but due to the breaking of the waves, the same energy contained in the waves is not used to make the wave peaks taller. In the experimental analysis this corresponds to an increase in the amount of noise on the spectra as well as an increase in the amplitude of the 0 Hz peak. In the next section we will further investigate if the same phenomenon is present in the simulation results. Figure 7.37 shows the rate of change of the decay rate and average wave height over the length of the tank. This shows that a change in the average wave height at a specific position along the slope is correlated to the average wave height. Specifically in the breaking region we can see that the changes in b are accompanied by corresponding changes in the average wave height.

7.5.2 Harmonic analysis

In this subsection I will examine the behaviour of the various spectral components beginning with the 0 Hz component.

As mentioned previously, the 0 Hz peak in the frequency spectrum rep-

resents any offset that the signals have from the the zero point on the y axis. Plotting the unnormalised amplitude of this peak we get Figure 7.38. We can see that the amplitude of the 0 Hz peak increases up to a maximum at the break point of $x = -5$ m then drops off rapidly before spiking one more time then dropping to a minimum at $x = -1.5$ m. What we would have expected to see here is a drop in the 0 Hz peak just before the break point, called the set down of the mean water level, followed by an increase in the 0 Hz peak after wave breaking known as the set up of the wave. That is the expected behaviour of real breaking waves and is exactly what was seen in the experimental analysis. However, in Figure 7.41 we see that this same behaviour does not occur when using a shoreline sponge later. Although we do see a decrease in the 0 Hz peak at the break point in Figure 7.38, followed by an upward spike, we do not see pattern of increasing mean water level after the break point. It appears that in the breaking region, that the mean water level decreases with the average wave height. The reader will remember from the beginning of this chapter that we added a sponge layer between $x = -1.4$m and the shoreline. This was done to reduce the intensity of reflections from the shoreline that were interfering with the rest of our analysis. This gives us accurate results for all our other analysis except for the amplitude of the 0 Hz peak amplitude. After some testing we found that if we removed the sponge layer at the shoreline we would get the correct 0 Hz peak amplitude results shown in Figure 7.39. This shows the peak wave height decreasing at the break point followed by a sudden increase. However, the effect of the shoreline reflections can be seen by looking at the oscillations in the pre-breaking region of the figure.

We will now move on to analysing the fundamental frequency component of the waves as well as some other harmonic frequencies. Figure 7.40 shows how the amplitude of the fundamental frequency component of the wave changes over the length of the tank. This figure appears to match the experimental results with a decrease in the amplitude of the fundamental frequency component of the wave as energy is consistently lost to the other harmonics of the waves. What we expect to see when looking at other harmonic components of the waves therefore is to see increases in their amplitude as they gain energy that was lost by the 0.4 Hz peak. Figures 7.41, 7.42, 7.43, and 7.44 below show the unnormalised harmonic peak amplitudes of four harmonic peaks along the length of the tank. What we can notice is that during the breaking region higher harmonic frequencies gain more energy and the lower harmonics lose energy. The 0.8 Hz peak gains energy consistently during the formation of the wave, then loses most of it during wave breaking. The higher harmonics gain a smaller amount of energy during wave formation then increase in energy during wave breaking. This is the same trend that is

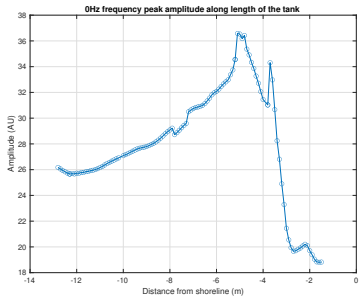


Figure 7.38: The 0 Hz peak amplitude when using a sponge layer between $x = -1.4$ m and the shoreline. We see that the mean water level does drop off at the break point, but then only increases momentarily after this point before dropping again.

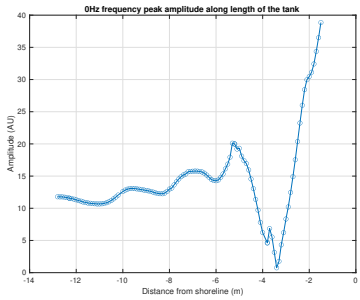


Figure 7.39: The 0 Hz peak amplitude when not using a sponge layer near the shoreline. This appears to more closely match experimental results as we see a drop in the mean water level at the break point followed by a sudden increase after the break point.

seen in the experimental results in the previous chapter.

7.6 Summary

In this chapter we analysed the results of numerical simulations of the Boussinesq equation and discussed how they compare to the experimental results of the previous chapter. The results and limitations of my own MATLAB simulation were shown and the decision was made to instead use FUNWAVE-TVD for my simulations. This software was introduced and configured before an analysis of its results was conducted. We saw that the average wave heights matched those from the previous chapter on experimental data analysis. We found that the phase velocity of the simulated waves also matched that of the experimental data. We found that the distribution of harmonic components was comparable to experimental data except for the 0 Hz peak that did not present the same decrease at the break point and subsequent increase after the break point that was seen in the experimental data. However we did see that when there is no sponge layer near the shoreline, that the 0 Hz peak behaved as in the experimental results.

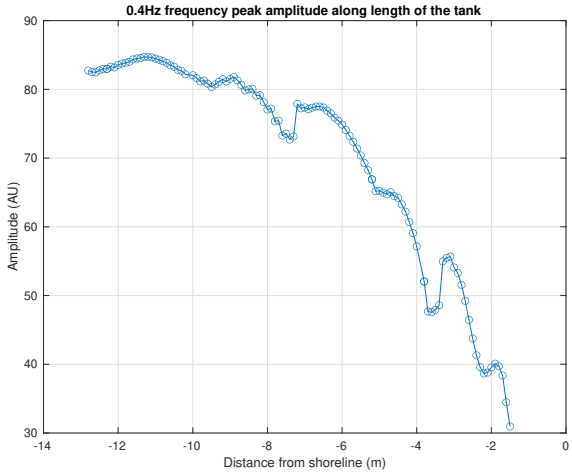


Figure 7.40: The unnormalised amplitude of the 0.4 Hz peak from the frequency spectra versus position.

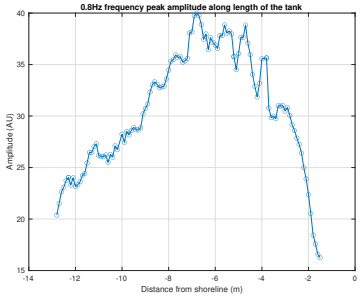


Figure 7.41: The amplitude of the 0.8 Hz peak across the length of the simulated tank.

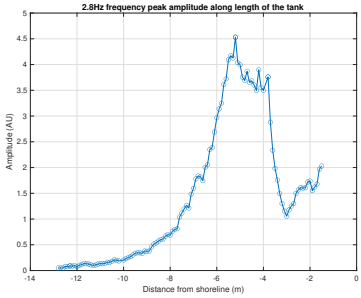


Figure 7.42: The amplitude of the 2.8 Hz peak across the length of the simulated tank.

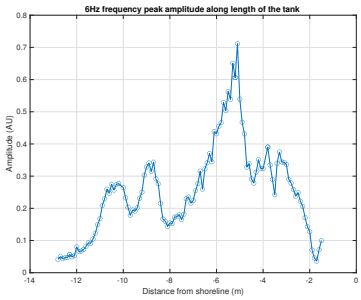


Figure 7.43: The amplitude of the 6 Hz peak across the length of the simulated tank.

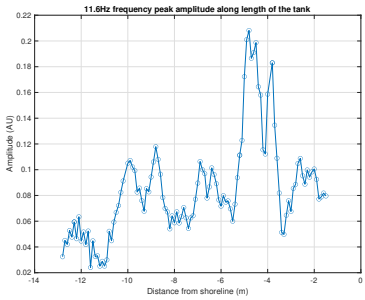


Figure 7.44: The amplitude of the 11.6 Hz peak across the length of the simulated tank.

Chapter 8

Summary and Conclusion

In this thesis I have examined the numerical solution of the Boussinesq equation for water waves propagating in a long tank and driven by an oscillating paddle at one end. In preparation for this main objective I have examined various numerical schemes for discretising partial differential equations. This was then applied to the familiar one dimensional wave equation and the Korteweg-de Vries equation. Thereafter I have examined the discretisation of the Boussinesq equation and its simulation. Further I have also independently analysed the data from a real experiment for comparison with the simulation. Below an examination of the main findings of these analyses is provided.

In order for numerical models to be valid, they must be comparable to real world experimental equivalents. Furthermore, all aspects of the real world experimental data must be exhaustive and comprehensively understood. This is particularly true for complex phenomena like Boussinesq beach waves. Since being certain of a numerical schemes validity is so crucial, this thesis analysed both the temporal and spectral data from a real world wave tank experiment and compared it with numerical simulations using FUNWAVE-TVD. Using the numerical model we were able to calculate the average wave heights, phase velocities, and spectral properties at points along the tank that were outside of the range of the experimental data.

We found that while the simulations appeared to follow the same pattern of wave heights over the lengths of the tank, the break point of the waves in our simulation was at $x = -5$ m instead of the $x = -4$ m in the experiments. The peak average wave height reached was 17.5 cm in the simulation while the analysis of experimental data found a value of 21.5 cm.

We conducted two simulations, one with a sponge layer between our last probe point at $x = -1.4$ m and the shoreline and another with no shoreline sponge layer. The simulation without the sponge layer had large wave reflec-

tions from the shoreline that presented as oscillations in our phase velocity calculations. We obtained cleaner results with the sponge layer, but at the cost of slightly different phase velocity results and mean water level results in the breaking region of the simulation than without a sponge layer. The sponge and no-sponge simulations phase velocities both spiked at the break point as is seen in the experimental data, but reached higher peak velocities of 2.2 m/s and 2.5 m/s respectively compared the the experimental results of 1.81 m/s. In the pre-breaking region the phase velocities appear to follow between the linear and non-linear theoretical curves of $\sqrt{gh}$ and $1.3\sqrt{gh}$ respectively.

By analysing the height of spectral peaks of waves at different points along the slope we showed how the range of harmonic frequency components contained in the waves increases as the waves progress up the slope, up until the break point where this range of harmonics decreases and oscillates. We showed how this distribution of harmonics matched experiment closely and was related to the wave height along the slope. We found that the simulation without the shoreline sponge layer had a mean water level (described by the 0 Hz peak of the spectra) more comparable to experimental data than that of the simulation with the shoreline sponge layer. However this had the drawback of shoreline reflections affecting spectra in the pre breaking region.

In future work more investigation could be done into reducing shoreline reflections in the simulations without the need for a sponge layer. There are numerous optimisations to the phase velocity code that can be done including merging the perfect reconstruction upsampler and cross correlation steps into a single function to reduce the number of FFT / iFFT steps needed. A thorough investigation can be done into determining the set of simulation conditions that allow the simulated waves to break at $x = -4$ m from the shoreline instead of $x = -5$ m and whether or not these changes are meaningful for predicting beach wave breaking regions. A thorough analysis of the error between the numerical and experimental data can be conducted. This will be easier to conduct if the simulated waves have the same breaking point as the experimental data.

Bibliography

Adytia, D. and Groesen, E. (2010), 'Variational boussinesq model for simulation of coastal waves and tsunamis', **1**, 122–128.

Aitchison, J. S. et al. (1991), 'Experimental observation of spatial soliton interactions', *Opt. Lett.* **16**(1), 15–17.
URL: *https://opg.optica.org/ol/abstract.cfm?URI=ol-16-1-15*

Ali, R., Saha, A. and Chatterjee, P. (2017), 'Analytical electron acoustic solitary wave solution for the forced kdv equation in superthermal plasmas', *Physics of Plasmas* **24**, 122106.

Ando, R., Thurey, N. and Wojtan, C. (2013), 'Highly adaptive liquid simulations on tetrahedral meshes', *ACM Trans. Graph. (Proc. SIGGRAPH 2013)* .

Antoine, X., Bao, W. and Besse, C. (2013), 'Computational methods for the dynamics of the nonlinear schrödinger/gross–pitaevskii equations', *Computer Physics Communications* **184**(12), 2621–2633.
URL: *https://www.sciencedirect.com/science/article/pii/S0010465513002403*

Askar, A. (1982), 'A generalization of the korteweg-de vries equation for anharmonic lattices and vectorial solitons', *International Journal of Engineering Science* **20**, 169–179.

Axler, S., Bourdon, P. and Wade, R. (2013), *Harmonic Function Theory*, Vol. 137, Springer Science & Business Media.

Bhatia, H., Norgard, G., Pascucci, V. and Bremer, P.-T. (2013), 'The helmholtz-hodge decomposition—a survey', *IEEE Transactions on Visualization and Computer Graphics* **19**(8), 1386–1404.

Blazek, J. (2001), *Computational Fluid Dynamics: Principles and Applications*, first edn.

Brauer, K. (2000), 'The korteweg-de vries equation: history, exact solutions, and graphical representation', *University of Osnabrück/, Germany* .

Bridges, T. J. and Reich, S. (2006), 'Numerical methods for hamiltonian pdes', *Journal of Physics A: Mathematical and General* **39**(19), 5287–5320.

Carretero-González, R. et al. (2017), 'A korteweg–de vries description of dark solitons in polariton superfluids', *Physics Letters A* **381**(45), 3805–3811.
URL: *https://www.sciencedirect.com/science/article/pii/S0375960117309490*

Catmull, E. and Clark, J. (1978), 'Clark, j.: Recursively generated b-spline surfaces on arbitrary topological meshes. computer-aided design 10(6), 350-355', *Computer-Aided Design* **10**, 350–355.

Chen, Q. (2006), 'Fully nonlinear boussinesq-type equations for waves and currents over porous beds', *Journal of Engineering Mechanics-asce - J ENG MECH-ASCE* **132**.

Chen, Y. et al. (2020), 'Reduction and analytic solutions of a variable-coefficient korteweg de vries equation in a fluid, crystal or plasma'.

Chong, T. (1978), 'A variable mesh finite difference method for solving a class of parabolic differential equations in one space variable', *Siam Journal on Numerical Analysis - SIAM J NUMER ANAL* **15**.

Deng, D. and Wu, Q. (2021), 'Analysis of the linearly energy- and mass-preserving finite difference methods for the coupled schrödinger-boussinesq equations', *Applied Numerical Mathematics* **170**.

Elgarayhi, A. et al. (2013), 'Propagation of nonlinear pressure waves in blood', *ISRN Computational Biology* **2013**.

Falissard, F. (2013), 'Genuinely multi-dimensional explicit and implicit generalized shapiro filters for weather forecasting, computational fluid dynamics and aeroacoustics', *Journal of Computational Physics* **253**, 344–367.

Feng, B.-F. and Mitsui, T. (1998), 'A finite difference method for the korteweg-de vries and the kadomtsev-petviashvili equations', *Journal of Computational and Applied Mathematics* **90**(1), 95–116.
URL: *https://www.sciencedirect.com/science/article/pii/S0377042798000065*

Harris, F. J. (1978), 'On the use of windows for harmonic analysis with the discrete fourier transform', *Proceedings of the IEEE* **66**(1), 51–83.

Holden, D. et al. (2019), Subspace neural physics: Fast data-driven interactive simulation, *in* 'Proceedings of the 18th annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation', pp. 1–12.

Ismail, M. S. and Mosally, F. (2014), 'A fourth order finite difference method for the good boussinesq equation', *Abstract and Applied Analysis* **2014**.

Jai Krishna Gautam, A. K. and Saxena, R. (1995), 'Windows: A tool in signal processing', *IETE Technical Review* **12**(3), 217–226.

Kakutani, S. (1944), 'Two-dimensional brownian motion and harmonic functions', *Proceedings of the Imperial Academy* **20**(10), 706–714.

Kallel, A. Y., Hu, Z. and Kanoun, O. (2022), 'Comparative study of ac signal analysis methods for impedance spectroscopy implementation in embedded systems', *Applied Sciences* **12**(2).
URL: *https://www.mdpi.com/2076-3417/12/2/591*

Kim, H.-Y. and Kim, H.-G. (2021), 'A novel adaptive mesh refinement scheme for the simulation of phase-field fracture using trimmed hexahedral meshes', *International Journal for Numerical Methods in Engineering* **122**(6), 1493–1512.
URL: *https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.6587*

Kirby, J. et al. (1998), 'Funwave 1.0: Fully nonlinear boussinesq wave model - documentation and user's manual', *Research report NO. CACR-98-06* .

Kong, L. and Wang, L. (2010), 'Numerical studies on boussinesq-type equations via a split-step fourier method', *Int. J. Comput. Math.* **87**, 1768–1784.

Kreyszig, E. (2011), *Advanced Engineering Mathematics*, tenth edn, John Wiley and Sons, Inc.

Li, X., Wei, X. and Zhang, Y. (2019), 'Hybrid non-uniform recursive subdivision with improved convergence rates', *Computer Methods in Applied Mechanics and Engineering* **352**.

Lind, S., Rogers, B. and Stansby, P. (2020), 'Review of smoothed particle hydrodynamics: towards converged lagrangian flow modelling', *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* **476**, 20190801.

Lind, S. and Stansby, P. (2016), 'High-order eulerian incompressible smoothed particle hydrodynamics with transition to lagrangian free-surface motion', *Journal of Computational Physics* **326**.

Liu, M. and Liu, G. (2010), 'Smoothed particle hydrodynamics (sph): an overview and recent developments', *Archives of Computational Methods in Engineering* **17**, 25–76.

Logan, J. D. (1987), *Applied Mathematics - A Contemporary Approach*, John Wiley and Sons, Inc.

Long, W. (2006*a*), Boussinesq Modeling of Waves, Currents and Sediment Transport, PhD thesis.

Long, W. (2006*b*), Boussinesq Modeling of Waves, Currents and Sediment Transport, PhD thesis.

Martínez-Ferrer, P. J. et al. (2018), 'Improved numerical wave generation for modelling ocean and coastal engineering problems', *Ocean Engineering* **152**, 257–272.
URL: *https://www.sciencedirect.com/science/article/pii/S0029801818300520*

Maruzewski, P., Le Touzé, D., Oger, G. and Avellan, F. (2010), 'Sph high-performance computing simulations of rigid solids impacting the free-surface of water', *Journal of Hydraulic Research* **48**.

Mehmood, A. et al. (2016), Numerical simulation of nonlinear water waves based on fully nonlinear potential flow theory in openfoam®-extend, *in* 'The 26th International Ocean and Polar Engineering Conference', OnePetro.

Mocz, P. and Succi, S. (2015), 'Numerical solution of the non-linear schrodinger equation using smoothed-particle hydrodynamics', *Physical Review E* **91**.

Mukaro, R., Govender, K. and Mccreadie, H. (2013), 'Wave height and wave velocity measurements in the vicinity of the break point in laboratory plunging waves', *Journal of Fluids Engineering* **135**.

Muller, M. (1956), 'Some continuous monte carlo methods for dirichlet problem', *The Annals of Mathematical Statistics* **27**.

Nwogu, O. (1993), 'An alternative form of the boussinesq equations for nearshore wave propagation', *Journal of Waterway Port Coastal and Ocean Engineering* **119**.

Patel, P., Kumar, P. and Rajni (2020), The numerical solution of boussinesq equation for shallow water waves, Vol. 2214, p. 020019.

Peregrine, D. (1967), 'Long waves on beach', *Journal of Fluid Mechanics* **27**, 815 – 827.

Pfaff, T. et al. (2021), Learning mesh-based simulation with graph networks, *in* 'International Conference on Learning Representations'.

Sanchez-Gonzalez et al. (2020), Learning to simulate complex physics with graph networks, *in* 'International Conference on Machine Learning', PMLR, pp. 8459–8468.

Sawhney, R. and Crane, K. (2020), 'Monte carlo geometry processing: A grid-free approach to pde-based methods on volumetric domains', *ACM Trans. Graph.* **39**(4).

Sawhney, R., Miller, B., Gkioulekas, I. and Crane, K. (2023), Walk on stars: A grid-free monte carlo method for pdes with neumann boundary conditions, Technical report.
URL: *http://arxiv.org/abs/2302.11815*

Scalerandi, M. (1997), 'A stable finite-difference scheme for the boussinesq equation'.

Schember, H. (1982), A New Model for Three-Dimensional Nonlinear Dispersive Long Waves, PhD thesis.
URL: *https://resolver.caltech.edu/CaltechETD:etd-09232005-153011*

Schäffer, H. A., Madsen, P. A. and Deigaard, R. (1993), 'A boussinesq model for waves breaking in shallow water', *Coastal Engineering* **20**(3), 185–202.
URL: *https://www.sciencedirect.com/science/article/pii/037838399390001O*

Sederberg, T., Zheng, J., Sewell, D. and Sabin, M. (1999), 'Non-uniform recursive subdivision surfaces', *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1998* .

Sengupta, T. and Bhaumik, S. (2019), *DNS of Wall-Bounded Turbulent Flows*.

Shi, F. et al. (2012), 'A high-order adaptive time-stepping tvd solver for boussinesq modeling of breaking waves and coastal inundation', *Ocean Modelling* **43-44**, 36–51.

Skogestad, J. O. and Kalisch, H. (2009), 'A boundary value problem for the kdv equation: Comparison of finite-difference and chebyshev methods', *Mathematics and Computers in Simulation* **80**(1), 151–163. Nonlinear

Waves: Computation and Theory VII.
URL: *https://www.sciencedirect.com/science/article/pii/S0378475409001888*

Sun, Y. et al. (2023), 'Analytical study of three-soliton interactions with different phases in nonlinear optics', *Nonlinear Dynamics* **111**(19), 18391–18400.
URL: *https://doi.org/10.1007/s11071-023-08786-z*

Taniuti, T. (1974), 'Reductive Perturbation Method and Far Fields of Wave Equations', *Progress of Theoretical Physics Supplement* **55**, 1–35.
URL: *https://doi.org/10.1143/PTPS.55.1*

Wang, H.-P., Yu-Shun, W. and Hu, Y.-Y. (2008), 'An explicit scheme for the kdv equation', *Chinese Physics Letters* **25**(7), 2335.

Wei, G. and Kirby, J. (1995), 'Time-dependent numerical code for extended boussinesq equations', *Journal of Waterway Port Coastal and Ocean Engineering-asce - J WATERW PORT COAST OC-ASCE* **121**.

Wei, G. et al. (1995), 'A fully nonlinear boussinesq model for surface waves. i: Highly nonlinear unsteady waves', *Journal of Fluid Mechanics* **294**, 71 – 92.

Xiao, Y. et al. (2020), 'An adaptive staggered-tilted grid for incompressible flow simulation', *ACM Trans. Graph.* **39**(6).
URL: *https://doi.org/10.1145/3414685.3417837*

Yan, J., Zheng, L., Lu, F. and Zhang, Q. (2022), 'Efficient energy-preserving methods for the schrödinger-boussinesq equation', *Mathematical Methods in the Applied Sciences* pp. n/a–n/a.

Zabusky, N. J. and Kruskal, M. D. (1965), 'Interaction of "solitons" in a collisionless plasma and the recurrence of initial states', *Physical review letters* **15**(6), 240.

Zeng, R., Wu, Z., Deng, S., Zhu, J. and Chi, X. (2021), 'Adaptive smoothing length method based on weighted average of neighboring particle density for sph fluid simulation', *Virtual Reality and Intelligent Hardware* **3**(2), 129–141. Special issue on simulation and interaction of fluid and solid dynamics.
URL: *https://www.sciencedirect.com/science/article/pii/S2096579621000115*

Zhang et al. (2018), 'Spectral method for solving the time fractional boussinesq equation', *Applied Mathematics Letters* **85**, 164–170.