

# Contents

<b>Preface</b> . . . . .	i
<b>Abstract</b> . . . . .	ii
<b>Acknowledgements</b> . . . . .	iii
<b>Acronyms</b> . . . . .	iv
<b>Contents</b> . . . . .	vii
<b>1 Introduction</b>	1
1.1 Motivation . . . . .	1
1.2 Literature . . . . .	3
1.3 Thesis overview . . . . .	4
<b>2 The Shallow Water Equations</b>	5
2.1 Notation . . . . .	5
2.2 Derivation of the SWE with conservative variables . . . . .	6
2.3 The SWE in vector form . . . . .	11
2.4 The 1D SWE in integral form . . . . .	11
2.5 The Shallow Water Equations in Spherical Coordinates . . . . .	13
2.6 The Linearized Shallow Water Equations in Spherical Coordinates . . . . .	18
<b>3 The Finite Volume Method</b>	21
3.1 The Finite Volume Method for the 1D Shallow Water Equations . . . . .	22
3.2 The Finite Volume Method for the 2D Shallow Water Equations . . . . .	24
3.3 The Finite Volume Method for the spherical 1D Linearized Shallow Water Equations . . . . .	25
3.4 The Riemann problem . . . . .	27

3.4.1	The Dam-Break problem . . . . .	28
3.4.2	Wave decomposition in the Riemann problem . . . . .	28
3.5	Numerical fluxes . . . . .	30
<b>4</b>	<b>Neural Networks and Fourier Neural Operators</b>	<b>34</b>
4.1	Convolutional Neural Networks . . . . .	34
4.2	Fourier Neural Operators . . . . .	35
<b>5</b>	<b>Data generation</b>	<b>39</b>
5.1	Data generation using the FVM . . . . .	39
5.2	Data Copernicus . . . . .	45
5.3	Mesh generation for the sphere . . . . .	45
<b>6</b>	<b>Numerical results</b>	<b>48</b>
6.1	The 1D Dam Break Problem . . . . .	48
6.2	Toro test cases . . . . .	49
6.3	1D linearized Shallow Water Equations on a sphere . . . . .	56
6.4	2D idealised Circular Dam Break Problem . . . . .	57
6.5	Scalability . . . . .	57
6.6	Animations . . . . .	58
<b>7</b>	<b>Data-driven results</b>	<b>60</b>
7.1	1D Shallow Water Equations with Gaussian initial conditions . . . . .	60
7.2	1D linearized Shallow Water Equations in Spherical Coordinates . . . . .	66
7.3	2D Shallow Water Equations with Gaussian initial conditions . . . . .	70
<b>8</b>	<b>Discussion</b>	<b>77</b>
<b>9</b>	<b>Conclusion</b>	<b>78</b>
9.1	Further work . . . . .	78
	<b>Bibliography</b>	<b>80</b>

<b>A Appendix</b>	<b>81</b>
A.1 Additional figures from 1D SWE spherical CNN and FNO . . . . .	81
A.2 FNO Toro test 1 . . . . .	84
A.3 2D SWE long term prediction . . . . .	85

# Chapter 1

## Introduction

This chapter provides the motivation for the project and explains the background and key objectives of the thesis. It also includes a literature review, where we outline the most important sources relevant to this work. Finally, this chapter offers an overview of the thesis and its structure.

### 1.1 Motivation

Natural disasters such as floods and tsunamis are becoming increasingly frequent, causing devastating impacts on human life and property [1]. Populations in coastal areas, river basins and flood-prone regions are particularly at risk. A recent example is the catastrophic flooding in Valencia, Spain, in late October 2024. On October 29, 2024, Valencia received a year's worth of rain in just eight hours, leading to flash floods that devastated the area, resulting in significant loss of life and property damage [2]. Satellite images highlight the extent of the flooding, comparing the region before the event on October 8, 2024, and after on October 30, 2024. These images are presented in Figure 1.1.

While it is impossible to prevent such disasters from occurring, trustworthy forecasts delivered in sufficiently short time can help in emergency response and disaster management. By having efficient tools to simulate flood scenarios it would be possible to reduce the time needed for decision-making in critical situations, potentially saving lives and minimizing the impact of such disasters. During a catastrophe, fast simulations are particularly valuable, even if some accuracy is sacrificed, as long as the results are accurate enough to make decisions. This is one aspect where fast simulations could be useful.

Another aspect is that when we are not in the middle of a catastrophe, running multiple simulations of different scenarios can provide a better understanding of water dynamics and a stronger data foundation. In this case it may not be needed to use a fast simulation, as we can prioritize accuracy to provide information of potential flood scenarios. For instance, the flooding in Germany during July 2021 was unexpected in several ways. Meteorological forecasts predicted heavy rain in the affected regions, and there were warnings of the potential for flooding [3]. Despite these forecasts, the intensity and scale of the floods surpassed expectations, and climate scientist expressed shock of the extent of the flooding [4]. This is a good example of how a deeper understanding of water dynamics, supported by simulations, could have helped to predict the water flow and its spread.

This motivates the study of the shallow water equations (SWE), a set of hyperbolic partial differential equations that describe the motion of a fluid in a shallow layer of water. These equations are essential for understanding and simulating water dynamics in shallow water regions, such as coastal areas, rivers, and flood-prone regions. A wide range of problems can be modeled by the shallow water equations, such as flooding, tsunamis and dam break

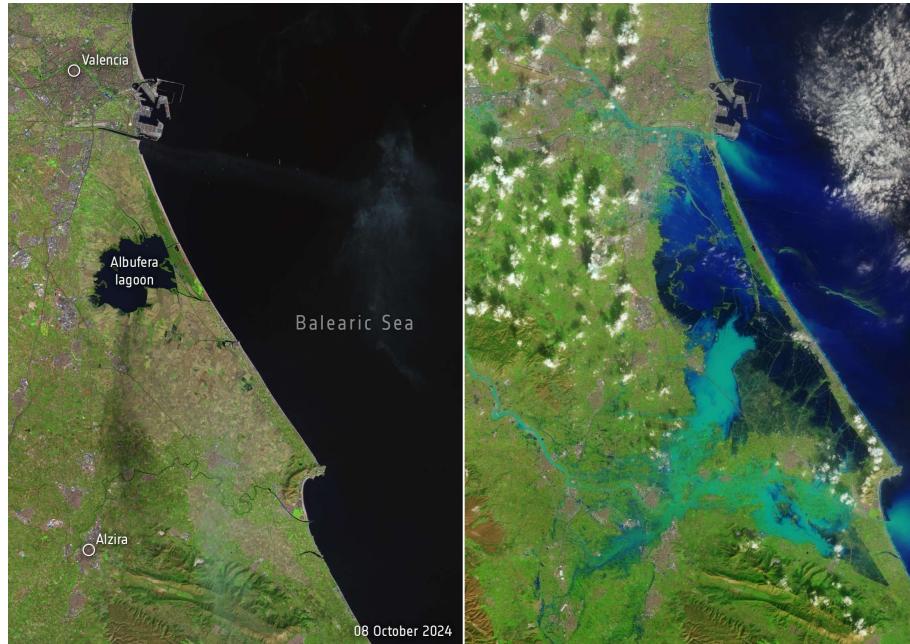


Figure 1.1: Before and after the floods in Valencia, Spain, October 2024.

Source: [https://www.esa.int/ESA\\_Multimedia/Images/2024/10/Valencia\\_flood\\_disaster](https://www.esa.int/ESA_Multimedia/Images/2024/10/Valencia_flood_disaster).

scenarios. By solving the shallow water equations, we can simulate and predict the behavior of water in these regions, providing valuable insights for disaster management and emergency response. In this work, we will derive the SWE with one spatial dimension (1D), two spatial dimensions (2D), and in spherical coordinates, which are particularly useful for modeling water flow on the Earth's surface.

The SWE can be solved using numerical methods, such as the finite volume method (FVM), which is a numerical technique for solving partial differential equations, by discretizing the domain into small control volumes and integrating the equations over these volumes. This method is widely used in computational fluid dynamics to model fluid behavior. In this work, we will implement the FVM in 1D and use it to solve the SWE for several scenarios, including the dam break problem. We will extend the FVM to 2D and solve the idealized circular dam break problem. The FVM solvers will be validated against known test cases, as this validation is critical for future work. For the FVM, it holds that a finer grid resolution improves solution accuracy but comes at the cost of an increased computational run time, resulting in slower simulations. This means that while high-resolution grids can provide more accurate results, they may not be practical for real-time simulations, such as during a flood event, where fast simulations are crucial for decision-making. This limitation motivates the investigation of data-driven approaches to solve the shallow water equations, which may offer a faster and more efficient way to simulate water dynamics while maintaining an acceptable level of accuracy.

This project investigates whether data-driven methods can provide a faster and more efficient alternative to traditional numerical methods for solving the shallow water equations. Specifically, we will explore training a convolutional neural network (CNN) and a Fourier neural operator (FNO) to solve the SWE. The data-driven models will be trained on the data generated from the FVM solvers, and evaluated on performance metrics such as run time, accuracy, long-term prediction capabilities, grid transferability and their response to new initial conditions. These initial conditions could include varying water heights, velocities, and other environmental factors that may change in real-world flood scenarios. We will compare the advantages and limitations of the data-driven approaches against numerical methods, focusing on their potential to improve disaster responder and flood prediction. This comparison will allow us to assess whether data-driven approaches offer a practical solution to efficiently solve the shallow water equations. We expect that data-driven models may be preferred for applications requiring fast

simulations, while numerical methods may be more suitable for scenarios demanding high accuracy. An interesting aspect is to investigate how much more efficient the data-driven methods may be compared to the traditional numerical methods, to determine to which applications they are best suited.

Additionally, we will examine the different models abilities to make long-term predictions, meaning predicting many time steps ahead. Numerical methods, including the FVM, solve the SWE one time step at a time, which can be computationally expensive, particularly for long-term predictions. In contrast, data-driven models also predict one time step ahead at a time, but they may not need to be retrained for each time step. Once trained, these models can make predictions for all time steps, potentially making them faster than numerical methods.

By analysing and comparing these approaches, this project aims to identify the context in which data-driven models can complement or even replace traditional numerical methods for solving the shallow water equations.

## 1.2 Literature

When working in this area it is inevitable to mention the work of E. F. Toro, who has written several books on the topic of Riemann solvers and the finite volume method, specifically for the shallow water equations. In this project, we will use the books *Shock-Capturing Methods for Free-Surface Shallow Flows* from 2001 [5], *Riemann Solvers and Numerical Methods for Fluid Dynamics* from 2009 [6] and the rather new book from 2024 *Computational Algorithms for Shallow Water Equations* [7] as references. The books have been especially useful when deriving the shallow water equations as well as understanding and describing the finite volume method, including the Riemann solvers used in this project. The course *Advanced Numerical Methods for Environmental Models* at the University of Trento, has provided a good foundation for the numerical methods used in this project, both in terms of lecture notes and exercises [8].

Working with the shallow water equations in spherical coordinates, the papers *Well-balanced methods for the shallow water equations in spherical coordinates* by Castro et al. [9] and *Physics-informed neural networks for the shallow-water equations on the sphere* by Bihlo et al. [10] are references for deriving the SWE in spherical coordinates. Additionally, the lecture notes *Shallow water on a sphere* by Raymond from New Mexico Tech [11] and the notes from Geophysical Fluid Dynamics Laboratory [12] have been valuable in this derivation. Furthermore, the papers by Gavete [13] and Galewsky [14] also provide important insights into the spherical shallow water equations. Implementing the FVM to solve the shallow water equations in spherical coordinates is a challenging task, as exact literature on the topic is limited. However, some sources discussing the discontinuous Galerkin scheme [15] for solving the spherical shallow water equations have been useful in this context.

The field of Fourier neural operators (FNO) is relatively new, and as a result, there is limited literature on the topic. However, the paper *Fourier Neural Operator for Parametric Partial Differential Equations* [16], written by several authors, is a key reference. The paper suggests that FNOs have shown great potential in being an efficient and resolution independent operator in the emerging field of scientific machine learning to solve partial differential equations. A key reason of their success is their ability to generalize to unseen data. For more on the topic of scientific machine learning, consider the tech report [17]. In the last years the company Nvidia has done some very interesting work on the topic of FNO, and they have published several blog posts on the topic. One of the posts consider the use of spherical Fourier neural operators (SFNO) to generate weather forecasts around the globe [18]. Another paper regarding SFNOs is [19], which generalizes FNOs on the sphere. Convolutional neural networks (CNNs) are a well-researched area, with numerous sources available on the topic. For this project, the paper *An Introduction to Convolutional Neural Networks* [20] and the blog post *A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way* [21] have been particularly helpful in explaining the theoretical foundations of the methods.

### 1.3 Thesis overview

The rest of the thesis is structured as follows. In chapter 2, we derive the shallow water equations (SWE) in 1D, 2D, and spherical coordinates. In chapter 3, we present the finite volume method (FVM) used to solve the shallow water equations, including the Riemann problem and the numerical fluxes essential for the FVM. These chapters form the theoretical and methodological foundation for the numerical methods employed in this project, leading into the discussion of data-driven methods.

In chapter 4, we introduce the concepts of convolutional neural networks (CNNs) and Fourier neural operators (FNOs), explaining how these methods can be applied to solve the shallow water equations. In chapter 5, we describe the process of generating the data required for the data-driven methods, as we produce all the data used in this project ourselves.

Chapter 6 focuses on the numerical results for the 1D and 2D SWE using the FVM. To validate these results, we compare the FVM outputs with test cases from the literature. Validation is crucial since the data-driven models are trained on the FVM data. In chapter 7, we present the results for the shallow water equations using the data-driven models. We analyze the outcomes, discuss the performance of the data-driven models, and compare them to the numerical results.

Finally, in chapter 8, we discuss the findings, and in chapter 9, we conclude the thesis and propose directions for future work.

## Chapter 2

# The Shallow Water Equations

In this chapter, we explore the theory behind the shallow water equations (SWE). We begin by introducing the relevant notation used throughout the report. Next, we derive the SWE in conservative form and present them in both vector and integral form for both the 1D- and 2D-case in cartesian coordinates. We extend the derivation to spherical coordinates, where we derive the nonlinear SWE and afterwards linearize the equations.

## 2.1 Notation

Before deriving the shallow water equations (SWE), we will introduce the notation that will be used throughout this report. In both the 1D case and the 2D case of the SWE, we use cartesian coordinates  $(x, y, z)$  with time denoted by  $t$ . Given that linear algebra is a fundamental tool used in this report, we first establish the relevant notation. Lowercase bold letters represent vectors, while uppercase bold letters represent matrices. For instance,  $\mathbf{a}$  is a vector of size  $r \times 1$ , where  $r \in \mathbb{N}$ , and  $\mathbf{A}$  is a matrix of size  $m \times n$  with  $m, n \in \mathbb{N}$ . The identity matrix, denoted by  $\mathbf{I}$ , is a square matrix with ones along the diagonal and zeros elsewhere. For example, the  $3 \times 3$  identity matrix is given by:

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

In this project, differential calculus plays a significant role. We denote partial derivatives using the following notation:

$$f_x = \frac{\partial f}{\partial x}, \quad f_y = \frac{\partial f}{\partial y}, \quad f_z = \frac{\partial f}{\partial z}. \quad (2.1.1)$$

The gradient operator, denoted by  $\nabla$ , gives the gradient of a scalar function  $f(x, y, z)$  as a vector:

$$\nabla f = [f_x \quad f_y \quad f_z].$$

Given two vectors  $\mathbf{a} = [a_1 \quad a_2 \quad a_3]^\top$  and  $\mathbf{b} = [b_1 \quad b_2 \quad b_3]^\top$ , the dot product of  $\mathbf{a}$  and  $\mathbf{b}$  is given by:

$$\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + a_3 b_3.$$

The dot product can also be written as a matrix product:

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^\top \mathbf{b}.$$

The divergence operator, represented as  $\nabla \cdot$ , gives the divergence of a vector  $\mathbf{a}$  as:

$$\nabla \cdot \mathbf{a} = \frac{\partial a_1}{\partial x} + \frac{\partial a_2}{\partial y} + \frac{\partial a_3}{\partial z} = a_{1x} + a_{2y} + a_{3z},$$

using the notation for partial derivatives introduced in (2.1.1). The tensor product of two vectors  $\mathbf{a}$  and  $\mathbf{b}$ , denoted by  $\mathbf{a} \otimes \mathbf{b}$ , is a matrix where each element is the product of the elements of  $\mathbf{a}$  and  $\mathbf{b}$ , i.e.,

$$\mathbf{a} \otimes \mathbf{b} = \begin{bmatrix} a_1 b_1 & a_1 b_2 & a_1 b_3 \\ a_2 b_1 & a_2 b_2 & a_2 b_3 \\ a_3 b_1 & a_3 b_2 & a_3 b_3 \end{bmatrix}.$$

Establishing this relevant notation, we can now derive the shallow water equations.

## 2.2 Derivation of the SWE with conservative variables

The shallow water equations are derived from the conservation laws for mass and momentum, and are fundamental in fluid dynamics. The derivation follows four steps: First we consider the conservation laws for mass and momentum, and then we consider the boundary conditions for a free surface problem. Afterwards we make some necessary assumptions and finally we use the boundary conditions to integrate the conservation laws over depth. The derivation follows the methods outlined in [5] and [22].

### Conservation laws

The conservation laws for mass and momentum state that the total mass and momentum are conserved over time. The conservation laws for mass and momentum can be expressed generally as follows [5, eq.'s (2.1) and (2.2)]:

$$\rho_t + \nabla \cdot (\rho \mathbf{v}) = 0, \quad (2.2.1a)$$

$$(\rho \mathbf{v})_t + \nabla \cdot (\rho \mathbf{v} \otimes \mathbf{v} + p \mathbf{I} - \mathbf{T}) = \rho \mathbf{g}, \quad (2.2.1b)$$

where  $\rho$  is the fluid density,  $\mathbf{v} = [u \ v \ w]^\top$  is the fluid velocity in the  $x, y$  and  $z$ -direction respectively,  $p$  is the pressure,  $\mathbf{I}$  is the identity matrix, and the vector  $\mathbf{g} = [g_1 \ g_2 \ g_3]^\top$  represents body forces including gravity. In these equations, the density  $\rho$  and the pressure  $p$  are dependent of  $x, y, z$  and  $t$ , but later we will introduce some assumptions that simplify the equations. The matrix  $\mathbf{T}$  is the viscous stress tensor, given by

$$\mathbf{T} = \begin{bmatrix} \tau_{xx} & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \tau_{yy} & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \tau_{zz} \end{bmatrix},$$

which accounts for the viscous forces in the fluid. However, in this project the viscous stress tensor  $\mathbf{T}$  is neglected, since we assume the function  $\tau(x, y, z)$  is constant. The matrix  $\mathbf{v} \otimes \mathbf{v}$  represents the tensor product of the velocity vector  $\mathbf{v}$  with itself, i.e.,

$$\mathbf{v} \otimes \mathbf{v} = \begin{bmatrix} u^2 & uv & uw \\ vu & v^2 & vw \\ wu & wv & w^2 \end{bmatrix}.$$

Note that  $\mathbf{v} \otimes \mathbf{v} = \mathbf{v}\mathbf{v}^\top$ . Putting this together, we can rewrite the momentum equation (2.2.1b) as

$$(\rho \mathbf{v})_t + \nabla \cdot (\rho \mathbf{v}\mathbf{v}^\top + p \mathbf{I}) - \rho \mathbf{g} = 0. \quad (2.2.2)$$

Applying the divergence operator and the product rule for differentiation, we can write out the mass equation (2.2.1a) as

$$\rho_t + \rho(u_x + v_y + w_z) + u\rho_x + v\rho_y + w\rho_z = 0. \quad (2.2.3)$$

In this project we consider incompressible fluids, meaning that the fluid density  $\rho$  is independent of the pressure  $p$ . We also assume that the fluid density only depends on temperature and salinity, and thus is independent of  $t, x, y$  and  $z$ . Additionally, we assume  $\rho$  is nonzero. Hence from (2.2.3) we obtain

$$u_x + v_y + w_z = 0, \quad (2.2.4)$$

also referred to as the mass conservation equation. Applying the divergence operator  $\nabla \cdot$ , the momentum conservation equation (2.2.2) can be written out as:

$$\rho_t \mathbf{v} + \rho \mathbf{v}_t + \rho \left[ \begin{array}{l} (u^2 + p)_x + (uv)_y + (uw)_z \\ (vu)_x + (v^2 + p)_y + (vw)_z \\ (wu)_x + (wv)_y + (w^2 + p)_z \end{array} \right] - \rho \mathbf{g} = 0. \quad (2.2.5)$$

We neglect all body forces in  $\mathbf{g}$ , except the gravitational force in the  $z$ -direction, i.e.,  $\mathbf{g} = [0 \ 0 \ -g]$ , where  $g$  is the gravity acceleration, which we assume to be constant. Hence, by using the product rule in (2.2.5) and that  $\rho_t = 0$  we obtain

$$\rho \begin{bmatrix} u_t \\ v_t \\ w_t \end{bmatrix} + \rho \begin{bmatrix} p_x + uu_x + vu_y + wu_z + u(u_x + v_y + w_z) \\ p_y + uv_x + vv_y + wv_z + v(u_x + v_y + w_z) \\ p_z + uw_x + vw_y + ww_z + w(u_x + v_y + w_z) \end{bmatrix} - \rho \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} = 0. \quad (2.2.6)$$

We apply (2.2.4) to (2.2.6) to remove terms, we move the pressure terms to the right hand side, and we divide by  $\rho$ . Putting it all together, the mass equation (2.2.1a) and the momentum equation (2.2.2), split in  $x, y$  and  $z$ -directions, simplify to

$$\left. \begin{aligned} u_x + v_y + w_z &= 0, \\ u_t + uu_x + vu_y + wu_z &= -\frac{1}{\rho} p_x, \\ v_t + uv_x + vv_y + wv_z &= -\frac{1}{\rho} p_y, \\ w_t + uw_x + vw_y + ww_z &= -\frac{1}{\rho} p_z - g. \end{aligned} \right\} \quad (2.2.7)$$

## Boundary conditions

The focus is on the flow of water with a free surface, meaning that the surface is not fixed and can move or change over time. To solve the SWE, it is essential to impose boundary conditions at both the bottom of the water column and at the free surface. We assume the bottom  $b$  is defined by a function

$$z = b(x, y),$$

meaning that the bottom is dependent on  $x$  and  $y$ , but not on the time  $t$ . Since the bottom is not moving over time, we refer to it as fixed. The free surface is defined by

$$z = s(x, y, t) \equiv b(x, y) + h(x, y, t),$$

where  $h(x, y, t)$  is the water depth at time  $t$ . The following illustration helps to visualize the setup:

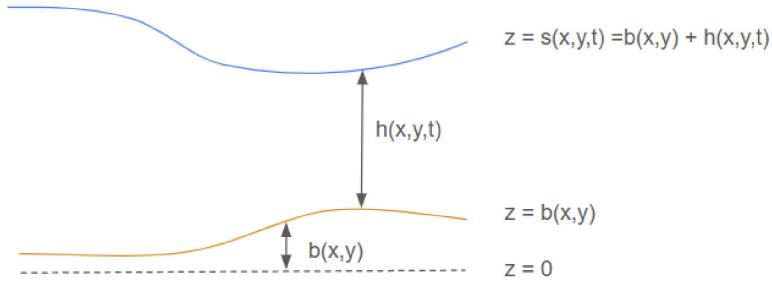


Figure 2.1: Illustration of a water column with a free surface.

We impose boundary conditions at the bottom and at the free surface, addressing both kinematic and dynamical conditions. To describe the boundaries mathematically, we introduce a boundary function  $\psi(x, y, z, t)$  that is zero on the boundaries:

$$\psi(x, y, z, t) = 0.$$

For the free surface, this boundary is given by

$$\psi|_{z=s} = z - s(x, y, t) = 0, \quad (2.2.8)$$

and for the bottom, it is described by

$$\psi|_{z=b} = z - b(x, y) = 0. \quad (2.2.9)$$

In deriving a kinematic condition, we assume that fluid particles on the boundary remain on the boundary over time. Mathematically this is expressed as

$$\frac{d}{dt}\psi(x, y, z, t) = 0.$$

Recall, that  $\frac{\partial\psi}{\partial t}$  is the partial derivative of  $\psi$  with respect to  $t$ , while the material derivative  $\frac{d\psi}{dt}$  accounts for the rate of change of  $\psi$  experienced by a fluid particle moving with the fluid. Hence, the material derivative of  $\psi$  wrt.  $t$  is given by

$$\begin{aligned} \frac{d\psi}{dt} &= \frac{\partial\psi}{\partial t} + \mathbf{v} \cdot \nabla\psi \\ &= \frac{\partial\psi}{\partial t} + \frac{dx}{dt} \frac{\partial\psi}{\partial x} + \frac{dy}{dt} \frac{\partial\psi}{\partial y} + \frac{dz}{dt} \frac{\partial\psi}{\partial z}. \end{aligned}$$

We see that  $\frac{dx}{dt}$  denotes the velocity in the  $x$ -direction, i.e.,  $u$ , and correspondingly  $\frac{dy}{dt}$  and  $\frac{dz}{dt}$  denotes  $v$  and  $w$  respectively. Thus, the kinematic condition is given by

$$\frac{d}{dt}\psi = \psi_t + u\psi_x + v\psi_y + w\psi_z = 0. \quad (2.2.10)$$

Applying this to the free surface by substituting (2.2.8) into the kinematic condition (2.2.10) yields

$$(s_t + us_x + vs_y - w)|_{z=s} = 0. \quad (2.2.11)$$

Similarly, for the bottom, substituting (2.2.9) into the kinematic condition (2.2.10) gives

$$(ub_x + vb_y - w)|_{z=b} = 0. \quad (2.2.12)$$

The dynamical condition is related to the pressure distribution at the free surface. We assume that the pressure at the free surface is equal to the pressure in the air above the surface, that is, the atmospheric pressure. Since absolute pressure levels are irrelevant, as we are primarily concerned with pressure differences, we set the pressure at the free surface to zero. This leads to the following expression for the pressure at the free surface:

$$p(x, y, z, t)|_{z=s} = 0. \quad (2.2.13)$$

This condition, known as the dynamical condition, relates to the forces acting on the boundaries of the fluid.

### Assumptions

To derive the SWE it is necessary to make some physical assumptions. The shallow water equations are an approximation to the full free-surface problem and result from the assumption that the vertical component of the acceleration is negligible. Therefore, we begin by assuming that the vertical acceleration, represented by the total derivative of the vertical velocity component  $w$  with respect to time, is negligible. This assumption leads to the condition

$$\frac{dw}{dt} = w_t + uw_x + vw_y + ww_z = 0.$$

Applying  $\frac{dw}{dt} = 0$  in the  $z$ -momentum conservation equation (2.2.7) simplifies it to

$$p_z = -\rho g.$$

By using properties of an integral, together with (2.2.13) we get

$$\int_{b(x,y)}^z -\rho g \, dt + \int_z^{s(x,y,t)} -\rho g \, dt = \int_{b(x,y)}^{s(x,y,t)} -\rho g \, dt = p|_{z=s(x,y,t)} = 0.$$

This implies that the pressure distribution follows

$$p = \int_{b(x,y)}^z -\rho g \, dt = \int_z^{s(x,y,t)} \rho g \, dt = \rho g(s - z), \quad (2.2.14)$$

where  $s$  is the surface height. Differentiating (2.2.14) with respect to  $x$  and  $y$  yields

$$p_x = \rho g s_x, \quad p_y = \rho g s_y.$$

Substituting these expressions into the  $x$ - and  $y$ -momentum conservation equations (2.2.7) leads to

$$\left. \begin{aligned} u_t + uu_x + vu_y + wu_z &= -gs_x, \\ v_t + uv_x + vv_y + wv_z &= -gs_y, \end{aligned} \right\} \quad (2.2.15)$$

which can be further simplified. We realize that both  $p_x$  and  $p_y$  are independent of  $z$ , implying that  $\frac{du}{dt}$  and  $\frac{dv}{dt}$  are also independent of  $z$ . Hence  $u_z = v_z = 0$ , implying that (2.2.15) can be simplified to

$$\left. \begin{aligned} u_t + uu_x + vu_y &= -gs_x, \\ v_t + uv_x + vv_y &= -gs_y. \end{aligned} \right\} \quad (2.2.16)$$

These are the momentum equations for the shallow water equations in two spatial dimensions.

### Integration over depth

The next step in deriving the SWE is to integrate the conservation equations over the vertical direction  $z$ . We integrate the mass conservation equation (2.2.4) and the momentum conservation equations in (2.2.16), from the bottom,  $z = b(x, y)$  to the free surface,  $z = s(x, y, t)$ . Starting with the mass conservation equation (2.2.4), we have

$$\int_b^s (u_x + v_y + w_z) \, dz = 0,$$

implying that, using linearity of the integral, we get

$$\int_b^s u_x \, dz + \int_b^s v_y \, dz + w|_{z=s} - w|_{z=b} = 0. \quad (2.2.17)$$

We will use Leibniz's integral rule [23], which is stated as follows:

$$\frac{d}{dx} \int_{a(x)}^{b(x)} f(x, t) dt = \int_{a(x)}^{b(x)} \frac{\partial}{\partial x} f(x, t) dt + f(x, b(x)) \frac{d}{dx} b(x) - f(x, a(x)) \frac{d}{dx} a(x), \quad (2.2.18)$$

to integrate the first two terms in (2.2.17), which yields

$$\left. \begin{aligned} \int_b^s u_x dz &= \frac{d}{dx} \int_b^s u dz - u|_{z=s} \frac{ds}{dx} + u|_{z=b} \frac{db}{dx}, \\ \int_b^s v_y dz &= \frac{d}{dy} \int_b^s v dz - v|_{z=s} \frac{ds}{dy} + v|_{z=b} \frac{db}{dy}. \end{aligned} \right\} \quad (2.2.19)$$

Note that since a change in  $x$  does not affect the  $y$ -component of the bottom or surface, we have that  $\frac{ds}{dx} = s_x$  and  $\frac{db}{dx} = b_x$ , and correspondingly for  $s_y$  and  $b_y$ . Likewise we can substitute  $\frac{d}{dx}$  with  $\frac{\partial}{\partial x}$  in the integrals, since the integrals are with respect to  $z$ , and  $u$  and  $v$  are independent of  $z$ . Inserting these results in (2.2.19) gives

$$\left. \begin{aligned} \int_b^s u_x dz &= \frac{\partial}{\partial x} \int_b^s u dz - u|_{z=s} s_x + u|_{z=b} b_x, \\ \int_b^s v_y dz &= \frac{\partial}{\partial y} \int_b^s v dz - v|_{z=s} s_y + v|_{z=b} b_y. \end{aligned} \right\} \quad (2.2.20)$$

We can now insert the integrals (2.2.20) into the integrated mass conservation equation (2.2.17) to get

$$\frac{\partial}{\partial x} \int_b^s u dz - u|_{z=s} s_x + u|_{z=b} b_x + \frac{\partial}{\partial y} \int_b^s v dz - v|_{z=s} s_y + v|_{z=b} b_y + w|_{z=s} - w|_{z=b} = 0. \quad (2.2.21)$$

To simplify this equation further, we consider the boundary conditions. From (2.2.12) we have

$$w|_{z=b} = (ub_x + vb_y)|_{z=b}, \quad (2.2.22)$$

and from (2.2.11) we have

$$w|_{z=s} = (s_t + us_x + vs_y)|_{z=s}. \quad (2.2.23)$$

We note that  $s = b + h$  and hence  $s_t = h_t$ , as the bottom is fixed. Recall that  $u$  and  $v$  are independent of  $z$ , and the water depth is  $h = s - b$ , meaning we have

$$\int_b^s u dz = u(s - b) = hu, \quad \int_b^s v dz = v(s - b) = hv.$$

Putting it all together the equation (2.2.21) simplifies to

$$h_t + (hu)_x + (hv)_y = 0, \quad (2.2.24)$$

which is also the first equation in the SWE stated in conservative form. When integrating the momentum equations (2.2.16) over the vertical direction, we see that since the equations are independent of  $z$ , the resulting equations are simply

$$\left. \begin{aligned} h(u_t + uu_x + vu_y + gs_x) &= 0, \\ h(v_t + uv_x + vv_y + gs_y) &= 0. \end{aligned} \right\} \quad (2.2.25)$$

We multiply (2.2.24) with  $u$  and  $v$  respectively, and add the resulting two equations to (2.2.25). Recall that  $s = h + b$ . By using the product rule for differentiation and collecting terms, we obtain the momentum equations stated in conservative form:

$$\left. \begin{aligned} (hu)_t + (hu^2 + \frac{1}{2}gh^2)_x + (huv)_y &= -ghb_x, \\ (hv)_t + (huv)_x + (hv^2 + \frac{1}{2}gh^2)_y &= -ghb_y. \end{aligned} \right\} \quad (2.2.26)$$

The three partial differential equations in (2.2.24) and (2.2.26) are the shallow water equations stated in conservative form.

## 2.3 The SWE in vector form

We present the SWE in vector form, which is useful as it allows all three partial differential equations to be combined into a single unified equation. The SWE can be written in differential conservation law form as the vector equation

$$\mathbf{U}_t + \mathbf{F}(\mathbf{U})_x + \mathbf{G}(\mathbf{U})_y = \mathbf{S}(\mathbf{U}), \quad (2.3.1)$$

where

$$\mathbf{U} = \begin{bmatrix} h \\ hu \\ hv \end{bmatrix}, \quad \mathbf{F}(\mathbf{U}) = \begin{bmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \\ huv \end{bmatrix}, \quad \mathbf{G}(\mathbf{U}) = \begin{bmatrix} hv \\ huv \\ hv^2 + \frac{1}{2}gh^2 \end{bmatrix} \quad \text{and} \quad \mathbf{S}(\mathbf{U}) = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 0 \\ -ghb_x \\ -ghb_y \end{bmatrix}.$$

We call  $\mathbf{U}$  the vector of conserved variables,  $\mathbf{F}(\mathbf{U})$  and  $\mathbf{G}(\mathbf{U})$  the flux vectors in the  $x$ - and  $y$ -direction, and  $\mathbf{S}(\mathbf{U})$  the source term vector. Note that the source term vector  $\mathbf{S}(\mathbf{U})$  often includes additional terms, such as friction and other external forces. However, since only gravity is considered in this case, the source term vector is simplified to the expression shown above. Moreover, if  $\mathbf{S}(\mathbf{U}) = 0$ , then we consider a homogeneous equation, and if  $\mathbf{S}(\mathbf{U}) \neq 0$ , we consider an inhomogeneous equation.

We also consider the one-dimensional case of the SWE in vector form, where the flow is only in the  $x$ -direction. The vector form of the SWE in 1D is given by

$$\mathbf{U}_t + \mathbf{F}(\mathbf{U})_x = \mathbf{S}(\mathbf{U}), \quad (2.3.2)$$

where

$$\mathbf{U} = \begin{bmatrix} h \\ hu \end{bmatrix}, \quad \mathbf{F}(\mathbf{U}) = \begin{bmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \end{bmatrix} \quad \text{and} \quad \mathbf{S}(\mathbf{U}) = \begin{bmatrix} 0 \\ -ghb_x \end{bmatrix}. \quad (2.3.3)$$

When deriving numerical schemes in the finite volume method (FVM), the vector form of the 1D SWE can be used as a foundation. The 1D formulation is preferred over the 2D version for this purpose because the principles underlying the numerical schemes remain the same in both dimensions. Using the 1D SWE provides a simpler framework to introduce and analyze core aspects of the method, such as flux evaluation, conservation laws, and the treatment of source terms. Once these concepts are well-understood in 1D, the method can be extended to 2D by applying similar principles, with modifications to account for the additional spatial dimension and the directional components of fluxes and source terms.

## 2.4 The 1D SWE in integral form

It is often more convenient to work with the integral form of the SWE, since the integral form of equations of the form (2.3.1) and (2.3.2) allows discontinuous solutions. We derive the integral form of the inhomogenous one-dimensional case of the SWE in vector form (2.3.2). The integral form is obtained by integrating the vector form (2.3.2) over a control volume  $V$  in the  $x, t$  plane

$$V = [x_L, x_R] \times [t_1, t_2].$$

The control volume is illustrated in Figure 2.2.

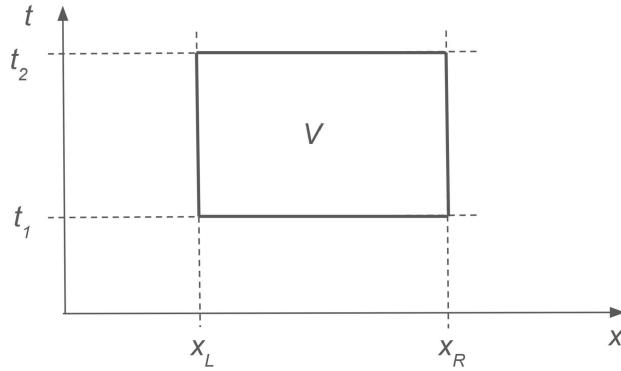


Figure 2.2: Illustration of a control volume  $V$  in the  $x, t$  plane. Illustration modified from [7].

First we integrate the vector form of the SWE (2.3.2) over  $x$  from  $x_L$  to  $x_R$  to obtain

$$\int_{x_L}^{x_R} \mathbf{U}_t \, dx + \int_{x_L}^{x_R} \mathbf{F}(\mathbf{U})_x \, dx = \int_{x_L}^{x_R} \mathbf{S}(\mathbf{U}) \, dx. \quad (2.4.1)$$

Using the fundamental theorem of calculus, we get that

$$\int_{x_L}^{x_R} \mathbf{F}(\mathbf{U}) \, dx = \mathbf{F}(\mathbf{U}(x_R, t)) - \mathbf{F}(\mathbf{U}(x_L, t)),$$

which we insert in (2.4.1):

$$\int_{x_L}^{x_R} \mathbf{U}_t \, dx = \mathbf{F}(\mathbf{U}(x_L, t)) - \mathbf{F}(\mathbf{U}(x_R, t)) + \int_{x_L}^{x_R} \mathbf{S}(\mathbf{U}) \, dx. \quad (2.4.2)$$

Then we integrate (2.4.2) over time from  $t_1$  to  $t_2$  to get

$$\int_{t_1}^{t_2} \int_{x_L}^{x_R} \mathbf{U}_t \, dx dt = \int_{t_1}^{t_2} \mathbf{F}(\mathbf{U}(x_L, t)) \, dt - \int_{t_1}^{t_2} \mathbf{F}(\mathbf{U}(x_R, t)) \, dt + \int_{t_1}^{t_2} \int_{x_L}^{x_R} \mathbf{S}(\mathbf{U}) \, dx dt.$$

Rewriting the left hand side using the fundamental theorem of calculus, we get

$$\int_{x_L}^{x_R} \mathbf{U}(x, t_2) \, dx = \int_{x_L}^{x_R} \mathbf{U}(x, t_1) \, dx + \int_{t_1}^{t_2} \mathbf{F}(\mathbf{U}(x_L, t)) \, dt - \int_{t_1}^{t_2} \mathbf{F}(\mathbf{U}(x_R, t)) \, dt + \int_{t_1}^{t_2} \int_{x_1}^{x_2} \mathbf{S}(\mathbf{U}) \, dx dt, \quad (2.4.3)$$

which is the integral form of the conservation laws for the SWE in 1D. An alternative integral form of (2.3.2) is stated in [7]:

$$\frac{d}{dt} \int_{x_L}^{x_R} \mathbf{U}(x, t) \, dx = \mathbf{F}(\mathbf{U}(x_L, t)) - \mathbf{F}(\mathbf{U}(x_R, t)) + \int_{x_L}^{x_R} \mathbf{S}(\mathbf{U}) \, dx. \quad (2.4.4)$$

From (2.4.4) we get that the integral form of the homogeneous SWE in 1D is given by

$$\frac{d}{dt} \int_{x_L}^{x_R} \mathbf{U}(x, t) \, dx = \mathbf{F}(\mathbf{U}(x_L, t)) - \mathbf{F}(\mathbf{U}(x_R, t)), \quad (2.4.5)$$

meaning that the rate of change of the integral over a spatial domain is equal to the net flux through the boundaries of the domain. In chapter 3 we will use the integral form of the conserved quantities in  $\mathbf{U}$  over a domain (2.4.3) to derive the FVM for the SWE in 1D.

## 2.5 The Shallow Water Equations in Spherical Coordinates

Until now we have derived the shallow water equations in cartesian coordinates. In this section, we will derive the shallow water equations in spherical coordinate, which is necessary if we want to model water flow on Earth. We will follow the methods used in [9] [10], [11] and [24]. To illustrate the spherical coordinates, we will use the latitude and longitude system, visualized in Figure 2.3.

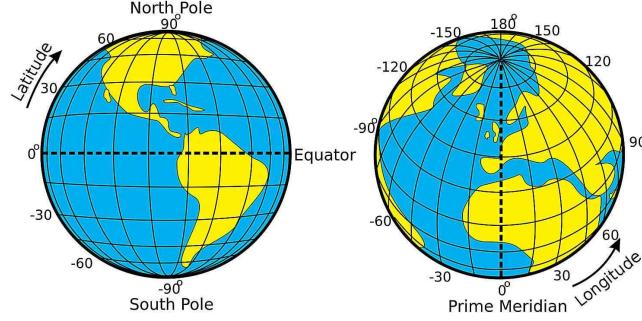


Figure 2.3: Illustration of latitude and longitude on the planet earth. Illustration from [25].

From Figure 2.3, we see that the latitude direction is the north-south component, whereas the longitude direction is the east-west component. The latitude angle, denoted by  $\phi$ , goes from  $-\frac{\pi}{2}$  radians at the south pole to  $\frac{\pi}{2}$  radians at the north pole, and the longitude angle, denoted by  $\theta$ , goes from 0 at the prime meridian through Europe and western Africa, increasing to the east, to  $2\pi$  radians. The spherical coordinates we use are  $(r, \theta, \phi)$ , where  $r$  is the radius from the center of the sphere,  $\theta$  is the longitude angle, and  $\phi$  is the latitude angle. Both angles are measured in radians. This also means, that any point on the surface of the sphere can be represented by the coordinates  $(\theta, \phi)$ , since we assume the radius  $r$  is constant. We consider a small domain of the sphere, as illustrated in Figure 2.4.

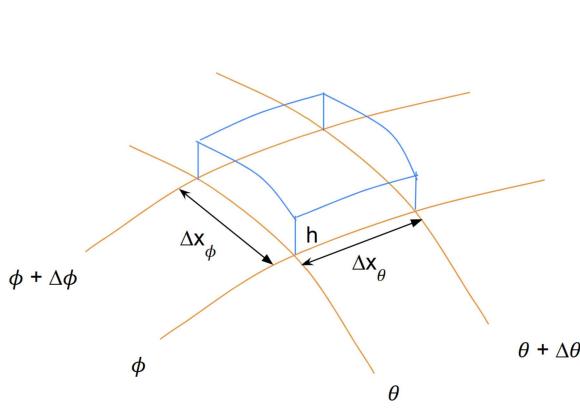


Figure 2.4: Illustrations of a small domain of the surface of the sphere.

The volume of the domain in Figure 2.4 is denoted by  $V$ . We want to find expressions for  $\Delta x_\phi$  and  $\Delta x_\theta$ , the distances in the  $\phi$  and  $\theta$  directions, respectively, as illustrated in Figure 2.4. We can find these distances by using

the arc length formula. Recall that the circumreference of a full circle is  $2\pi r$ , where  $r$  is the radius of the circle. The arc length is a fraction of the full circumreference, and it is given by the formula  $l = rv$ , where  $l$  is the arc length,  $r$  is the radius, and  $v$  is the angle in radians. Assuming Earth's latitude side is a circle, we can find the distance  $\Delta x_\phi$  by using the arc length formula, as:

$$\Delta x_\phi = r\Delta\phi,$$

where  $\Delta\phi$  is the change in the latitude angle and  $r$  is the radius of the sphere. We assume that Earth is a perfect sphere, meaning that the radius is constant. Considering the distance in the longitude dimension  $\Delta x_\theta$ , we need to make some adjustments, as we can see that the circumreference at equator is larger than at the poles. That is, we need to consider the radius of the circle at the given latitude  $\phi$ . To illustrate this, we consider Figure 2.5.

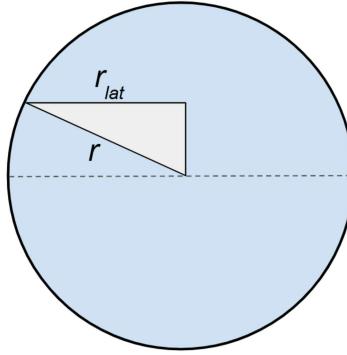


Figure 2.5: Illustration of the radius of the circle at the given latitude  $\phi$ .

In Figure 2.5, we consider a right triangle where the length of the hypotenuse is equal to the radius of the sphere. The length of the adjacent side  $r_{lat}$  is the radius of the longitude circle at the given latitude  $\phi$ . By using trigonometry, we can express the radius of the circle at the given latitude  $\phi$  as

$$r_{lat} = r \cos(\phi).$$

Using that, together with the arc length formula, we can find the distance  $\Delta x_\theta$  as:

$$\Delta x_\theta = r \cos(\phi) \Delta\theta.$$

We can then compute the volume of a small domain of the sphere, as shown in Figure 2.4. The volume  $V$  is given by

$$\begin{aligned} V &= \Delta x_\phi \Delta x_\theta h \\ &= r^2 h \cos(\phi) \Delta\phi \Delta\theta, \end{aligned}$$

assuming that the height of the domain is  $h$ , and that the domain is rectangular. This is a fair assumption for small values of  $\Delta x_\phi$  and  $\Delta x_\theta$ . We also assume that  $\phi$  is not too close to the poles, as  $\cos(\phi)$  will go to zero at the poles. We are interested in the rate of change of  $V$  with respect to time, and we can find this by taking the time derivative of the volume. That is, we consider the partial derivative with respect to time of the volume  $V$ :

$$\frac{\partial V}{\partial t} = r^2 \cos(\phi) \Delta\phi \Delta\theta \frac{\partial h}{\partial t}. \quad (2.5.1)$$

In (2.5.1) we have utilized that  $r$  is constant, and that  $\cos(\phi)$ ,  $\Delta\phi$  and  $\Delta\theta$  are independent of the time  $t$ . We use  $u_\theta$  and  $u_\phi$  to denote the velocities in the  $\theta$ - and  $\phi$ -directions. We are interested in the rate at which fluid volume enters the region from the sides. We can find this rate by considering the flux of fluid volume through the sides of

the domain. That is, we consider how much fluid volume enters the domain from the  $\theta$ -direction, and how much fluid volume enters the domain from the  $\phi$ -direction. We also consider how much fluid volume leaves the domain in the  $\theta$ - and  $\phi$ -directions. The net flux of fluid volume into the domain is the difference between the influx and the outflux. The rate of change of the volume  $V$  with respect to time is equal to the net flux of fluid volume into the domain. That is, we have that

$$V_t = F_{net,\theta} + F_{net,\phi}, \quad (2.5.2)$$

which combined with (2.5.1) gives us the equation

$$r^2 \cos(\phi) \Delta\phi \Delta\theta \frac{\partial h}{\partial t} = F_{net,\theta} + F_{net,\phi}. \quad (2.5.3)$$

To find the net flux in the  $\theta$  and  $\phi$  directions, we consider the influx and outflux in these directions. We calculate the influx at the  $\theta$ -line and the outflux at  $(\theta + \Delta\theta)$ -line, see Figure 2.4, to find the net flux. The influx is the area of the  $\theta$  line times the velocity in the  $\theta$  direction at the  $\theta$  line. The influx in the  $\theta$ -direction is given by

$$F_{in,\theta} = u_\theta(\theta) h(\theta) r \Delta\phi,$$

where  $h(\theta)$  is the height of the water at the  $\theta$ -line, assumed to be constant along the line, and  $u_\theta(\theta)$  is the velocity in the  $\theta$ -direction at the  $\theta$ -line. This way we can compute how much the volume changes due to the influx. For the outflux we do the same just for the  $\theta + \Delta\theta$  line, introducing the notation  $\theta' = \theta + \Delta\theta$ . The outflux is

$$F_{out,\theta} = u_\theta(\theta + \Delta\theta) h(\theta + \Delta\theta) r \Delta\phi = u_\theta(\theta') h(\theta') r \Delta\phi.$$

The net flux in the  $\theta$  direction is the difference between the influx and the outflux, and is given by

$$F_{net,\theta} = F_{in,\theta} - F_{out,\theta} = (u_\theta(\theta) h(\theta) - u_\theta(\theta') h(\theta')) r \Delta\phi. \quad (2.5.4)$$

We can do the same for the  $\phi$  direction, also using the notation  $\phi' = \phi + \Delta\phi$ . Hence, we obtain the net flux in the  $\phi$  direction as

$$F_{net,\phi} = F_{in,\phi} - F_{out,\phi} \quad (2.5.5)$$

$$= (u_\phi(\phi) h(\phi) \cos(\phi) - u_\phi(\phi') h(\phi') \cos(\phi')) r \Delta\theta. \quad (2.5.6)$$

By inserting (2.5.4) and (2.5.5) into (2.5.3) we can write:

$$h_t r^2 \cos(\phi) \Delta\phi \Delta\theta = (u_\theta(\theta) h(\theta) - u_\theta(\theta') h(\theta')) r \Delta\phi + (u_\phi(\phi) h(\phi) \cos(\phi) - u_\phi(\phi') h(\phi') \cos(\phi')) r \Delta\theta. \quad (2.5.7)$$

Since we are interested in  $\frac{\partial h}{\partial t} = h_t$ , we divide (2.5.7) by the area of the element  $r^2 \cos(\phi) \Delta\phi \Delta\theta$ . Hence we get

$$\frac{\partial h}{\partial t} = \frac{u_\theta(\theta) h(\theta) - u_\theta(\theta') h(\theta')}{r \cos(\phi) \Delta\theta} + \frac{u_\phi(\phi) h(\phi) \cos(\phi) - u_\phi(\phi') h(\phi') \cos(\phi')}{r \cos(\phi) \Delta\phi}, \quad (2.5.8)$$

where  $\phi \neq \pm\frac{\phi}{2}$ . By collecting terms to the left hand side and changing the order of the numerators, we can rewrite (2.5.8) as

$$\frac{\partial h}{\partial t} + \frac{u_\theta(\theta') h(\theta') - u_\theta(\theta) h(\theta)}{r \cos(\phi) \Delta\theta} + \frac{u_\phi(\phi') h(\phi') \cos(\phi') - u_\phi(\phi) h(\phi) \cos(\phi)}{r \cos(\phi) \Delta\phi} = 0. \quad (2.5.9)$$

Next step is to investigate the limit values, as  $\Delta\theta$  and  $\Delta\phi$  goes to zero. We can find the limit values by using the definition of the derivative. The derivative of a function  $f$  with respect to  $x$  is defined as

$$f'(x) = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}.$$

We can use this definition to find the limit values in (2.5.9).

$$\lim_{\Delta\theta \rightarrow 0} \frac{u_\theta(\theta') h(\theta') - u_\theta(\theta) h(\theta)}{\Delta\theta} = \frac{\partial}{\partial\theta} (h u_\theta),$$

and

$$\lim_{\Delta\phi \rightarrow 0} \frac{u_\phi(\phi')h(\phi')\cos(\phi') - u_\phi(\phi)h(\phi)\cos(\phi)}{\Delta\phi} = \frac{\partial}{\partial\phi}(hu_\phi\cos(\phi)).$$

Inserting these results in (2.5.9) yields

$$h_t + \frac{1}{r\cos(\phi)} \left( (hu_\theta)_\theta + (hu_\phi\cos(\phi))_\phi \right) = 0,$$

which is the mass conservation equation in spherical coordinates and is the first equation in the shallow water equations in spherical coordinates. The next step is to derive the momentum equations in spherical coordinates. In this case, we focus on the horizontal velocity components, specifically the velocity tangential to the surface of the sphere, i.e., the  $\theta$  and  $\phi$  velocities. The vertical velocity is neglected, as the key assumption in the shallow water equations is that the vertical component of the acceleration is negligible. Additionally, when considering Earth, the water layer is thin compared to the radius of the Earth, referred to as a thin-layer approximation. We need to express the horizontal velocity  $u_h$ , which is dependent on the variables  $\theta, \phi$  and  $t$ . Since  $\theta$  and  $\phi$  are angles, we introduce the unit vectors  $\mathbf{e}_\theta$  and  $\mathbf{e}_\phi$  on the surface in the  $\theta$ - and  $\phi$ -directions. The unit vectors are illustrated in Figure 2.6.



Figure 2.6: Illustration of the unit vectors  $\mathbf{e}_\theta$  and  $\mathbf{e}_\phi$ , tangential to the surface.

We can express the horizontal velocity  $u_h$  in terms of the unit vectors as

$$u_h(\theta, \phi, t) = u_\theta \mathbf{e}_\theta + u_\phi \mathbf{e}_\phi. \quad (2.5.10)$$

We are then interested in the total derivative of the horizontal velocity  $u_h$  in (2.5.10) with respect to time. The total derivative is given by

$$\frac{du_h}{dt} = \frac{\partial u_h}{\partial t} + \frac{d\theta}{dt} \frac{\partial u_h}{\partial \theta} + \frac{d\phi}{dt} \frac{\partial u_h}{\partial \phi}. \quad (2.5.11)$$

If we differentiate the longitude angle  $\theta$  with respect to time, we get the angular velocity  $\omega_\theta$  in the  $\theta$  direction, i.e.,

$$\frac{d\theta}{dt} = \omega_\theta,$$

meaning that if  $\omega_\theta > 0$ , the point is moving eastwards, and if  $\omega_\theta < 0$ , the point is moving westwards. Similarly, if we differentiate the latitude angle  $\phi$  with respect to time, we get the angular velocity  $\omega_\phi$  in the  $\phi$  direction, i.e.,

$$\frac{d\phi}{dt} = \omega_\phi,$$

meaning that if  $\omega_\phi > 0$ , the point is moving northwards, and if  $\omega_\phi < 0$ , the point is moving southwards. By using the derivative of the arc length formula, we get that

$$\frac{d\theta}{dt} = \frac{u_\theta}{r\cos(\phi)}, \quad \frac{d\phi}{dt} = \frac{u_\phi}{r}. \quad (2.5.12)$$

We can now insert (2.5.12) into (2.5.11) to find the total derivative of the horizontal velocity split into the  $\theta$  and  $\phi$  directions:

$$\left. \begin{aligned} \frac{du_\theta}{dt} &= \frac{\partial u_\theta}{\partial t} + \frac{u_\theta}{r \cos(\phi)} \frac{\partial u_\theta}{\partial \theta} + \frac{u_\phi}{r} \frac{\partial u_\theta}{\partial \phi}, \\ \frac{du_\phi}{dt} &= \frac{\partial u_\phi}{\partial t} + \frac{u_\theta}{r \cos(\phi)} \frac{\partial u_\phi}{\partial \theta} + \frac{u_\phi}{r} \frac{\partial u_\phi}{\partial \phi}. \end{aligned} \right\} \quad (2.5.13)$$

We know that the right hand side of (2.5.13) are the given physical forces acting on the fluid. Earlier in this project, we focused on the shallow water equations in cartesian coordinates, accounting solely for gravitational forces. However, in spherical coordinates, additional physical forces must be considered. These include the Coriolis force, centripetal acceleration, and the effects of Earth's curvature. First we consider the gravitational force acting on the fluid, described as  $-g\nabla h$ , where  $g$  is the gravity acceleration, and  $\nabla h$  is the gradient of the height  $h$ . We consider the gradient of  $h$ :

$$\nabla h = \frac{\partial h}{\partial x_\theta} \mathbf{e}_\theta + \frac{\partial h}{\partial x_\phi} \mathbf{e}_\phi \quad (2.5.14)$$

$$= \frac{1}{r \cos(\phi)} h_\theta \mathbf{e}_\theta + \frac{1}{r} h_\phi \mathbf{e}_\phi, \quad (2.5.15)$$

meaning that the gravity force acting on the fluid in the  $\theta$  and  $\phi$  directions are given by:

$$\left. \begin{aligned} \theta - \text{direction: } & -\frac{g}{r \cos(\phi)} h_\theta, \\ \phi - \text{direction: } & -\frac{g}{r} h_\phi. \end{aligned} \right\}$$

Hence, we obtain the two momentum equations in spherical coordinates as

$$\begin{aligned} (u_\theta)_t + \frac{u_\theta}{r \cos(\phi)} (u_\theta)_\theta + \frac{u_\phi}{r} (u_\theta)_\phi &= -\frac{g}{r \cos(\phi)} h_\theta + \text{other forces}, \\ (u_\phi)_t + \frac{u_\theta}{r \cos(\phi)} (u_\phi)_\theta + \frac{u_\phi}{r} (u_\phi)_\phi &= -\frac{g}{r} h_\phi + \text{other forces}. \end{aligned} \quad (2.5.16)$$

The next force we consider is the Coriolis force, which is a force that acts on moving objects on the surface of the earth [26]. The Coriolis force is given by  $f = 2\Omega \sin(\phi)$ , where  $\Omega$  is the angular velocity of the earth. The Coriolis force in the  $\theta$  and  $\phi$  directions are then given by:

$$\left. \begin{aligned} \theta - \text{direction: } & fu_\phi, \\ \phi - \text{direction: } & -fu_\theta. \end{aligned} \right\}$$

The last thing we need to take into account when working in the spherical domain is the curvature of the earth. This adds the following terms:

$$\left. \begin{aligned} \theta - \text{direction: } & \frac{u_\theta u_\phi}{r} \tan(\phi), \\ \phi - \text{direction: } & -\frac{u_\theta^2}{r} \tan(\phi). \end{aligned} \right\}$$

There are several formulations of the SWE in spherical coordinates. Inserting these forces into the momentum equations, we get the following formulation of the shallow water equations in spherical coordinates:

$$\left. \begin{aligned} h_t + \frac{1}{r \cos(\phi)} \left( (hu_\theta)_\theta + (hu_\phi \cos(\phi))_\phi \right) &= 0, \\ (u_\theta)_t + \frac{u_\theta}{r \cos(\phi)} (u_\theta)_\theta + \frac{u_\phi}{r} (u_\theta)_\phi - \frac{u_\theta u_\phi}{r} \tan(\phi) + \frac{g}{r \cos(\phi)} h_\theta - fu_\phi &= 0, \\ (u_\phi)_t + \frac{u_\theta}{r \cos(\phi)} (u_\phi)_\theta + \frac{u_\phi}{r} (u_\phi)_\phi + \frac{u_\theta^2}{r} \tan(\phi) + \frac{g}{r} h_\phi + fu_\theta &= 0, \end{aligned} \right\} \quad (2.5.17)$$

where  $r$  is the radius,  $(\theta, \phi)$  are the longitude and latitude angles,  $h$  is the height of the water,  $u_\theta$  and  $u_\phi$  are the velocities in the  $\theta$ - and  $\phi$ -directions and  $g$  is the gravitational constant. We can also write the spherical SWE (2.5.17) in vector form as

$$\mathbf{U}_t + \frac{1}{r \cos(\phi)} \mathbf{F}(\mathbf{U})_\theta + \frac{1}{r} \mathbf{G}(\mathbf{U})_\phi = \mathbf{S}, \quad (2.5.18)$$

where  $\frac{1}{r \cos(\phi)} \mathbf{F}(\mathbf{U})_\theta$  and  $\frac{1}{r} \mathbf{G}(\mathbf{U})_\phi$  are the flux terms in the  $\theta$ - and  $\phi$ -directions and  $\mathbf{S}$  is the source term.

## 2.6 The Linearized Shallow Water Equations in Spherical Coordinates

In this section, we derive the linearized shallow water equations (LSWE) in spherical coordinates, from the nonlinear SWE in spherical coordinates, given in (2.5.17). The derivation and methodology presented here are based on the sources [27] and [28]. The LSWE are employed for their simplicity, providing a foundation for understanding wave dynamics in a spherical geometry. Later in this project, we will apply the finite volume method to solve these equations numerically. We focus on one spatial dimension, namely the longitude  $\theta$ . This approach assumes a constant latitude  $\phi$ , which simplifies the equations to a one-dimensional framework. Hence we can neglect the third equation in (2.5.17) and the velocity component  $u_\phi$ . Thus, we introduce  $u = u_\theta$  to denote the fluid velocity in the  $\theta$  direction. We also neglect all forces, except for the gravitational force. Thus we have the following SWE in spherical coordinates for one spatial dimension:

$$\left. \begin{aligned} h_t + \frac{1}{r \cos(\phi)} (hu)_\theta &= 0, \\ u_t + \frac{u}{r \cos(\phi)} u_\theta + \frac{g}{r \cos(\phi)} h_\theta &= 0. \end{aligned} \right\} \quad (2.6.1)$$

To perform the linearization, we use perturbation theory, meaning we assume that the water height  $h$  and the velocity  $u$  are small perturbations from a state of rest. Let the perturbations in water height and velocity be represented by:

$$h = h_0 + h', \quad u = u_0 + u', \quad (2.6.2)$$

where  $h_0$  and  $u_0$  are the equilibrium height and velocity, and  $h'$  and  $u'$  are the perturbations. We assume that the equilibrium height  $h_0$  is constant and that the equilibrium velocity  $u_0$  is zero. By substituting the perturbed variables (2.6.2) into the 1D spherical SWE (2.6.1) and utilizing that  $u_0 = 0$ , we obtain

$$\left. \begin{aligned} (h_0 + h')_t + \frac{1}{r \cos(\phi)} ((h_0 + h')(u')_\theta + (h_0 + h')_\theta(u')) &= 0, \\ (u')_t + \frac{u'}{r \cos(\phi)} (u')_\theta + \frac{g}{r \cos(\phi)} h'_\theta &= 0. \end{aligned} \right\} \quad (2.6.3)$$

We neglect the terms involving products of perturbations arising in (2.6.3), as they are second-order terms. Hence, we obtain the linearized shallow water equations in spherical coordinates with one spatial dimension  $\theta$  and time  $t$  as:

$$\left. \begin{aligned} h'_t + \frac{h_0}{r \cos(\phi)} u'_\theta &= 0, \\ u'_t + \frac{g}{r \cos(\phi)} h'_\theta &= 0. \end{aligned} \right\} \quad (2.6.4)$$

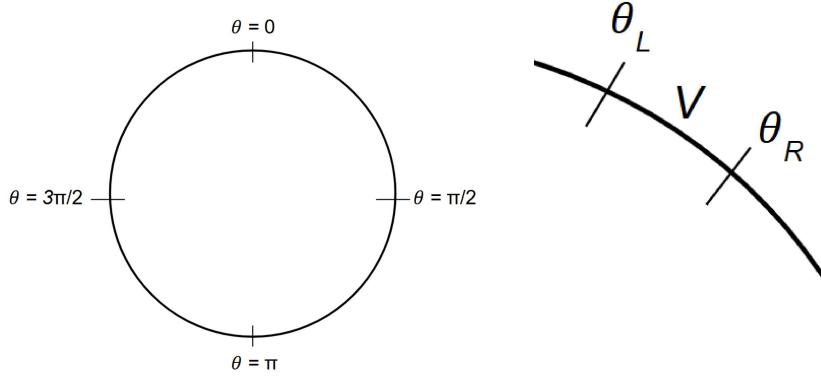
These equations describe the evolution of small perturbations in water height and velocity on a spherical surface. We can also write the spherical LSWE in vector form as

$$\mathbf{W}_t + \mathbf{A}\mathbf{W}_\theta = 0, \quad (2.6.5)$$

where  $\mathbf{W} = \begin{bmatrix} h' \\ u' \end{bmatrix}$  and the coefficient matrix  $\mathbf{A}$  is constant and given as:  $\mathbf{A} = \begin{bmatrix} 0 & \frac{h_0}{r \cos(\phi)} \\ \frac{g}{r \cos(\phi)} & 0 \end{bmatrix}$ .

### Integral form of the 1D spherical LSWE

As in the previous sections it is beneficial to state the integral form of the 1D spherical LSWE, as it will be used when deriving the finite volume scheme. We consider the longitude  $\theta$  as the spatial dimension on a full circle, meaning the domain is  $[0, 2\pi]$  radians. The domain is divided into  $N$  cells or control volumes, that each has a length of  $\Delta\theta = \frac{2\pi}{N}$  radians. We consider a control volume  $V$  with cell interfaces at  $\theta_L$  and  $\theta_R$ , as illustrated in Figure 2.7b.



(a) Illustration of the grid for the 1D SWE with small cells.  
 (b) Illustration of the grid for the 1D SWE with a small domain.

Figure 2.7: Grid illustrations for the 1D LSWE in spherical coordinates.

In Figure 2.7a, we see the circle, which is the full domain, and in Figure 2.7b we see a small subdomain of the circle.

The integral form of the LSWE is obtained by integrating the control volume given by  $V = [\theta_L, \theta_R] \times [t_1, t_2]$ , where  $t_2$  denotes the time step after  $t_1$ , over space and time. We begin by integrating the vector form of the 1D spherical LSWE in (2.6.5) over the spatial dimension  $\theta$  from  $\theta_L$  to  $\theta_R$  to obtain

$$\int_{\theta_L}^{\theta_R} \mathbf{W}_t \, d\theta + \int_{\theta_L}^{\theta_R} \mathbf{A} \mathbf{W}_\theta \, d\theta = 0. \quad (2.6.6)$$

Since the matrix  $\mathbf{A}$  is constant, we can place it outside of the integral in (2.6.6) to get

$$\int_{\theta_L}^{\theta_R} \mathbf{W}_t \, d\theta + \mathbf{A} \int_{\theta_L}^{\theta_R} \mathbf{W}_\theta \, d\theta = 0. \quad (2.6.7)$$

We use the fundamental theorem of calculus to rewrite (2.6.7) as

$$\int_{\theta_L}^{\theta_R} \mathbf{W}_t \, d\theta = \mathbf{A} (\mathbf{W}(\theta_L, t) - \mathbf{W}(\theta_R, t)). \quad (2.6.8)$$

We then integrate (2.6.8) over time from  $t_1$  to  $t_2$  to get

$$\int_{t_1}^{t_2} \int_{\theta_L}^{\theta_R} \mathbf{W}_t \, d\theta \, dt = \mathbf{A} \left( \int_{t_1}^{t_2} \mathbf{W}(\theta_L, t) \, dt - \int_{t_1}^{t_2} \mathbf{W}(\theta_R, t) \, dt \right). \quad (2.6.9)$$

Rewriting (2.6.9) gives

$$\int_{\theta_L}^{\theta_R} \mathbf{W}(\theta, t_2) \, d\theta = \int_{\theta_L}^{\theta_R} \mathbf{W}(\theta, t_1) \, d\theta - \mathbf{A} \left( \int_{t_1}^{t_2} \mathbf{W}(\theta_R, t) \, dt - \int_{t_1}^{t_2} \mathbf{W}(\theta_L, t) \, dt \right), \quad (2.6.10)$$

which we refer to as the integral form of the LSWE in spherical coordinates with one spatial dimension. The integral form (2.6.10) is the foundation for the finite volume method, which we will use to solve the linearized shallow water equations in spherical coordinates numerically.

## Chapter 3

# The Finite Volume Method

In this section, we present the finite volume method (FVM) for solving nonlinear systems of balance laws, specifically focusing on the shallow water equations (SWE). Nonlinear problems are more challenging than linear problems, as stability and convergence theory are more difficult to establish. In particular, when dealing with hyperbolic systems such as the SWE, the presence of discontinuities such as shock waves, hydraulic jumps, and rarefaction waves requires special treatment. Our focus is on discontinuous solutions, which can accurately capture these shocks and other discontinuities. The FVM provides a framework for dealing with such solutions in a robust and efficient manner. Unlike traditional methods, which may struggle to capture sharp gradients or discontinuities, FVM inherently incorporates a mechanism to resolve such features by solving the local Riemann problem at the interfaces between computational cells. This characteristic makes it particularly well-suited for problems involving abrupt changes in the flow, such as those encountered in the modeling of floods, tsunamis, or dam breaks.

The approach described here is based on the work of LeVeque [29], who developed a framework for using the FVM in the context of hyperbolic systems of partial differential equations. The key idea is to discretize the computational domain into small cells or control volumes and compute the fluxes across the boundaries of these cells. The solution within each cell is then updated using the information from neighboring cells, ensuring that conservation laws are satisfied. By solving the local Riemann problem at the cell interfaces, the FVM can accurately capture discontinuities and preserve the sharp transitions between different flow regimes. In addition to its ability to handle discontinuities, the FVM also has several other advantages, such as its ability to conserve mass, momentum, and energy in a discrete sense. This property is crucial when solving balance laws, as it ensures that the physical conservation laws are respected even at the discrete level. The method is also highly flexible, allowing for the use of unstructured grids, which can be beneficial for complex geometries and irregular domains. Despite its advantages, the FVM also has limitations. The accuracy of the method depends heavily on the choice of numerical flux function and the resolution of the computational grid. In the presence of strong discontinuities, additional techniques such as limiters or high-order schemes may be required to maintain stability and avoid spurious oscillations. Furthermore, while the FVM is well-suited for handling hyperbolic problems, it may not be as effective for problems with smooth solutions or for capturing fine-scale features in the solution without sufficient grid resolution.

In this chapter, we derive the finite volume scheme for the 1D SWE and extend it to 2D in cartesian coordinates. We introduce the MUSCL (monotonic upstream-centered scheme for conservation laws) scheme for higher-order accuracy and discuss the FVM for the 1D linearized SWE in spherical coordinates. Additionally, we examine the Riemann problem, including the dam break problem, and present numerical flux functions for solving the local Riemann problem at cell interfaces.

### 3.1 The Finite Volume Method for the 1D Shallow Water Equations

We begin by introducing finite volume methods for the SWE in one spatial dimension. Specifically, we work in the  $x, t$ -plane, discretizing the domain into finite control volumes or cells. In chapter 2, we derived the integral form for a single control volume  $V = [x_L, x_R] \times [t_1, t_2]$ . Here, we extend this approach to a global domain consisting of multiple cells. To accommodate this, we update the notation:  $x_{i-1/2}$  and  $x_{i+1/2}$  denote the cell interfaces, while  $t_n$  and  $t_{n+1}$  represent the time levels. The control volume for cell  $i$  at time level  $n$  is defined as:

$$V_i^n = [x_{i-1/2}, x_{i+1/2}] \times [t_n, t_{n+1}],$$

where  $\Delta x = x_{i+1/2} - x_{i-1/2}$  is the length of the cell and  $\Delta t = t_{n+1} - t_n$  is the time step size. The cell  $V_i^n$  is illustrated in Figure 3.1.

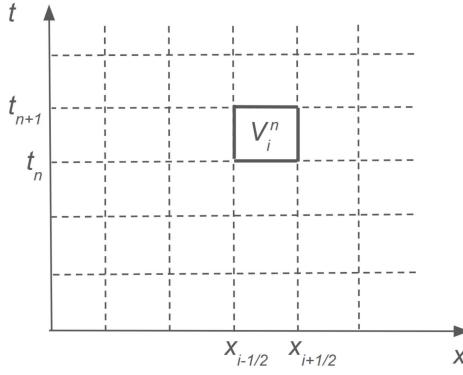


Figure 3.1: Illustration of the control volume  $V_i^n$  in the  $x, t$  plane.

The grid can be uniform or non-uniform, depending on the application. For now, we will assume a uniform grid for simplicity. The finite volume formula is derived from the integral form of the 1D SWE (2.4.3). The integral form stated in the new variables  $(x_{i-1/2}, x_{i+1/2}, t_n, t_{n+1})$  over the cell  $V_i^n$  is given by

$$\begin{aligned} \int_{x_{i-1/2}}^{x_{i+1/2}} \mathbf{U}(x, t_{n+1}) dx &= \int_{x_{i-1/2}}^{x_{i+1/2}} \mathbf{U}(x, t_n) dx + \int_{t_n}^{t_{n+1}} \mathbf{F}(\mathbf{U}(x_{i-1/2}, t)) dt - \int_{t_n}^{t_{n+1}} \mathbf{F}(\mathbf{U}(x_{i+1/2}, t)) dt \\ &\quad + \int_{t_n}^{t_{n+1}} \int_{x_{i-1/2}}^{x_{i+1/2}} \mathbf{S}(\mathbf{U})(x, t) dx dt. \end{aligned} \tag{3.1.1}$$

We divide the integral form (3.1.1) by the cell length  $\Delta x$  to obtain

$$\begin{aligned} \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} \mathbf{U}(x, t_{n+1}) dx &= \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} \mathbf{U}(x, t_n) dx \\ &\quad - \frac{\Delta t}{\Delta x} \left[ \frac{1}{\Delta t} \int_{t_n}^{t_{n+1}} \mathbf{F}(\mathbf{U}(x_{i+1/2}, t)) dt - \frac{1}{\Delta t} \int_{t_n}^{t_{n+1}} \mathbf{F}(\mathbf{U}(x_{i-1/2}, t)) dt \right] \\ &\quad + \frac{\Delta t}{\Delta x \Delta t} \int_{x_{i-1/2}}^{x_{i+1/2}} \int_{t_n}^{t_{n+1}} \mathbf{S}(\mathbf{U})(x, t) dx dt. \end{aligned}$$

To simplify the numerical implementation, we introduce cell-averaged values. These are defined as the averages of the conserved variables  $\mathbf{U}$ , the fluxes  $\mathbf{F}$ , and the source terms  $\mathbf{S}$  over the volume  $V_i^n$ . This leads to the explicit conservative form of the finite volume scheme:

$$\mathbf{U}_i^{n+1} = \mathbf{U}_i^n - \frac{\Delta t}{\Delta x} (\mathbf{F}_{i+1/2}^n - \mathbf{F}_{i-1/2}^n) + \Delta t \mathbf{S}_i. \tag{3.1.2}$$

The value  $\mathbf{U}_i^n$  is the average value over the  $i$ -th cell at time  $t_n$ :

$$\mathbf{U}_i^n = \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} \mathbf{U}(x, t_n) dx,$$

also known as the cell average. The flux  $\mathbf{F}_{i-1/2}^n$  is the average flux across the line  $x = x_{i-1/2}$  from time  $t_n$  to  $t_{n+1}$ :

$$\mathbf{F}_{i-1/2}^n = \frac{1}{\Delta t} \int_{t_n}^{t_{n+1}} \mathbf{F}(\mathbf{U}(x_{i-1/2}, t)) dt,$$

and correspondingly the flux  $\mathbf{F}_{i+1/2}^n$  is the average flux across the line  $x = x_{i+1/2}$  from time  $t_n$  to  $t_{n+1}$ :

$$\mathbf{F}_{i+1/2}^n = \frac{1}{\Delta t} \int_{t_n}^{t_{n+1}} \mathbf{F}(\mathbf{U}(x_{i+1/2}, t)) dt.$$

The source term  $\mathbf{S}_i$  is the average source term over the  $i$ -th cell at time  $t_n$ :

$$\mathbf{S}_i = \frac{1}{\Delta t \Delta x} \int_{t_n}^{t_{n+1}} \int_{x_{i-1/2}}^{x_{i+1/2}} \mathbf{S}(x, t) dx dt.$$

The values are illustrated in Figure 3.2.

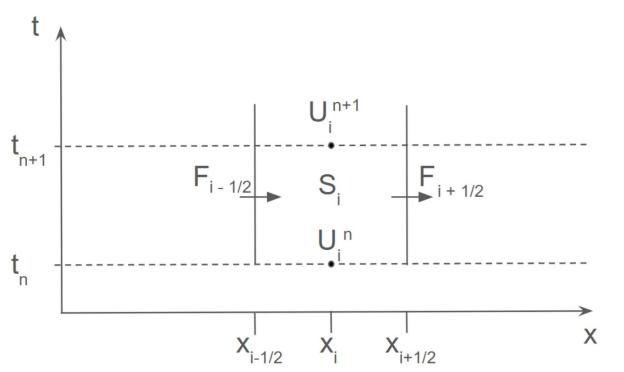


Figure 3.2: Illustration of the grid for the 1D SWE.

The central idea of the FVM is to define the numerical flux  $\mathbf{F}_{i+1/2}^n$ , at the cell interface, as a function of the cell averages  $\mathbf{U}_i^n$  and  $\mathbf{U}_{i+1}^n$ , since the solution is known only in terms of these cell averages. Consequently, the FVM does not provide pointwise values of the solution, i.e.,  $\mathbf{U}(x, t)$ , but instead gives cell-averaged values,  $\mathbf{U}_i^n$ , over the control volume. One of the main challenges in the FVM is to determine appropriate numerical flux functions that, based on the available cell averages, can reasonably approximate the fluxes at the cell interfaces. Later in the thesis, we will consider several numerical flux functions that can be used to solve the local Riemann problem at the cell interfaces.

The finite volume method are closely related to the finite difference method (FDM), but they differs as they are based on the integral form of the conservation laws. Where finite difference methods tend to break down near discontinuities in the solution, finite volume methods are more suited, since they are based on the integral form of the conservation laws. The key distinction between the FVM and the FDM lies in their formulation: while the FVM is based on the integral conservation over finite volumes, the FDM is based on satisfying the differential equations point-wise at mesh points.

## 3.2 The Finite Volume Method for the 2D Shallow Water Equations

We now extend the FVM to two spatial dimensions, meaning we consider both the  $x$ - and  $y$ -direction. Consider the 2D SWE in vector form (2.3.1) with  $\mathbf{S}(\mathbf{U}) = 0$ :

$$\mathbf{U}_t + \mathbf{F}(\mathbf{U})_x + \mathbf{G}(\mathbf{U})_y = 0. \quad (3.2.1)$$

Following the methods outlined in [6], an explicit finite volume scheme to solve (3.2.1) is given by

$$\mathbf{U}_{i,j}^{n+1} = \mathbf{U}_{i,j}^n - \frac{\Delta t}{\Delta x} (\mathbf{F}_{i+1/2,j} - \mathbf{F}_{i-1/2,j}) - \frac{\Delta t}{\Delta y} (\mathbf{G}_{i,j+1/2} - \mathbf{G}_{i,j-1/2}). \quad (3.2.2)$$

This is the unsplit finite volume method, meaning that, in a single step, the cell average  $\mathbf{U}_{i,j}^n$  in cell  $V_{i,j}$  is updated using the fluxes from all intercell boundaries. This means, that in each time step, we need to solve the Riemann problem at all cell interfaces, and then use the fluxes to update the cell averages. The fluxes  $\mathbf{F}_{i-1/2,j}$  and  $\mathbf{F}_{i+1/2,j}$  are the average fluxes across the lines  $x = x_{i-1/2}$  and  $x = x_{i+1/2}$ . Correspondingly, the fluxes  $\mathbf{G}_{i,j-1/2}$  and  $\mathbf{G}_{i,j+1/2}$  are the average fluxes across the lines  $y = y_{j-1/2}$  and  $y = y_{j+1/2}$ . We work on a domain discretized into rectangular cells of size  $\Delta x \times \Delta y$ , where  $\Delta x = x_{j+1/2} - x_{j-1/2}$  and  $\Delta y = y_{j+1/2} - y_{j-1/2}$ . The fluxes are illustrated in Figure 3.3.

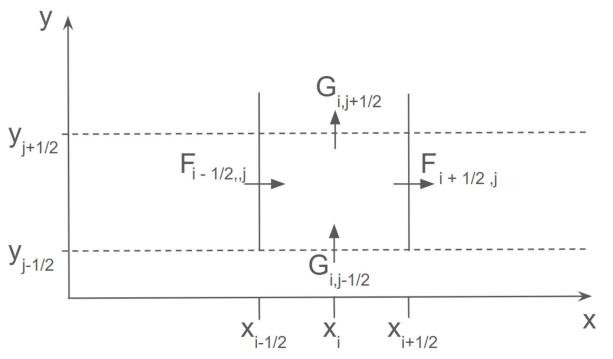


Figure 3.3: Illustration of the fluxes for the 2D SWE.

When using the simultaneous update scheme (3.2.2), we must consider potential stability issues. To address this, several methods can be employed.

The first method we consider is the weighted average flux (WAF) scheme. The basic WAF scheme, without any nonlinear TVD modification, is prone to oscillations. To mitigate this, a total variation diminishing (TVD) constraint can be enforced. To achieve the TVD condition, additional constraints are imposed on the numerical flux or reconstruction process, often through the use of slope limiters. A limiter function is a technique used to reduce oscillations in the solution, especially near sharp edges or discontinuities, such as shock waves. For this project, we use the minmod limiter, defined as

$$\text{minmod}(a,b) = \begin{cases} \max(0, \min(a,b)) & \text{if } a > 0 \\ \min(0, \max(a,b)) & \text{if } a < 0. \end{cases}$$

Another method we consider is the MUSCL scheme. This method is second-order accurate in both time and space, while the WAF scheme is second-order accurate in space but relies on the time-stepping scheme for temporal accuracy. The MUSCL scheme, combined with slope limiters, ensures the solution remains satisfies the TVD constraint, effectively preventing oscillations that could occur with high-order spatial discretization near shocks, discontinuities, or sharp changes in the solution. The key idea in MUSCL is that, instead of using cell averages as in the 1D finite volume method, we reconstruct the solution in each cell using piecewise first-degree polynomials.

This reconstruction provides a higher-order representation of the solution within each cell. In this project, we use the MUSCL scheme with the TVD criteria to ensure smooth and stable solutions. A numerical method  $U_{i,j}^{n+1}$  is called TVD if it satisfies the following condition [30]:

$$TV(U^{n+1}) \leq TV(U^n), \quad (3.2.3)$$

for all grid functions  $U^n$ . For a discrete function  $U_{i,j}$  defined on a grid  $i = 0, 1, \dots, N$  and  $j = 0, 1, \dots, N$ , the total variation  $TV(U)$  is defined as

$$TV(U) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |U_{i+1,j} - U_{i,j}| + |U_{i,j+1} - U_{i,j}|. \quad (3.2.4)$$

The total variation measures the sum of the absolute differences between neighboring grid points. A TVD scheme ensures that the total variation of the solution does not increase in time, which is crucial for maintaining stability and preventing oscillations. The TVD property is used in shock-capturing schemes to prevent nonphysical oscillations near discontinuities, helping to accurately capture the shock location.

### 3.3 The Finite Volume Method for the spherical 1D Linearized Shallow Water Equations

In this section, we derive the finite volume method for the 1D shallow water equations in spherical coordinates, by following the same method as in section 3.1. We begin by stating the integral form of the 1D LSWE in spherical coordinates on a global domain, meaning we now consider a control volume given by

$$V_i^n = [\theta_{i-1/2}, \theta_{i+1/2}] \times [t_n, t_{n+1}],$$

where  $\Delta\theta = \theta_{i+1/2} - \theta_{i-1/2}$  is the length of the cell and  $\Delta t = t_{n+1} - t_n$  is the time step size. We rewrite the previous derived integral form (2.6.10) to be in global variables:

$$\int_{\theta_{i-1/2}}^{\theta_{i+1/2}} \mathbf{W}(\theta, t_{n+1}) d\theta = \int_{\theta_{i-1/2}}^{\theta_{i+1/2}} \mathbf{W}(\theta, t_n) d\theta - \mathbf{A} \left( \int_{t_n}^{t_{n+1}} \mathbf{W}(\theta_{i+1/2}, t) dt - \int_{t_n}^{t_{n+1}} \mathbf{W}(\theta_{i-1/2}, t) dt \right). \quad (3.3.1)$$

We divide the integral form (3.3.1) with the cell length  $\Delta\theta$  to obtain

$$\begin{aligned} \frac{1}{\Delta\theta} \int_{\theta_{i-1/2}}^{\theta_{i+1/2}} \mathbf{W}(\theta, t_{n+1}) d\theta &= \frac{1}{\Delta\theta} \int_{\theta_{i-1/2}}^{\theta_{i+1/2}} \mathbf{W}(\theta, t_n) d\theta \\ &\quad - \frac{\Delta t}{\Delta\theta} \mathbf{A} \left( \frac{1}{\Delta t} \int_{t_n}^{t_{n+1}} \mathbf{W}(\theta_{i-1/2}, t) dt - \frac{1}{\Delta t} \int_{t_n}^{t_{n+1}} \mathbf{W}(\theta_{i+1/2}, t) dt \right). \end{aligned}$$

Averaging over the terms for a finite volume gives the first-order explicit time-stepping finite volume scheme:

$$\mathbf{W}_i^{n+1} = \mathbf{W}_i^n - \frac{\Delta t}{\Delta\theta} (\mathbf{F}_{i+1/2}^n - \mathbf{F}_{i-1/2}^n). \quad (3.3.2)$$

The scheme uses the cell averages:

$$\mathbf{W}_i^n = \frac{1}{\Delta\theta} \int_{\theta_{i-1/2}}^{\theta_{i+1/2}} \mathbf{W}(\theta, t_n) d\theta$$

The flux  $\mathbf{F}_{i-1/2}^n$  is the average flux across the line  $\theta = \theta_{i-1/2}$  from time  $t_n$  to  $t_{n+1}$ :

$$\mathbf{F}_{i-1/2}^n = \frac{1}{\Delta t} \mathbf{A} \int_{t_n}^{t_{n+1}} (\mathbf{W}(\theta_{i-1/2}, t)) dt,$$

and correspondingly the flux  $\mathbf{F}_{i+1/2}^n$  is the average flux across the line  $\theta = \theta_{i+1/2}$  from time  $t_n$  to  $t_{n+1}$ :

$$\mathbf{F}_{i+1/2}^n = \frac{1}{\Delta t} \mathbf{A} \int_{t_n}^{t_{n+1}} (\mathbf{W}(\theta_{i+1/2}, t)) dt.$$

## The Explicit Runge-Kutta 4th Order Method

We will now introduce how we can use the explicit Runge-Kutta 4th order (ERK4) method to solve the 1D spherical LSWE. The ERK4 method solves the general initial value problem (IVP) for ordinary differential equations (ODEs):

$$\frac{dy(t)}{dt} = f(y(t), t), \quad y(t_0) = y_0. \quad (3.3.3)$$

To solve the IVP (3.3.3) using ERK4 we choose a step size  $\Delta t > 0$  and define the update formula:

$$y_{n+1} = y_n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4), \\ t_{n+1} = t_n + \Delta t, \quad n = 0, 1, \dots, N - 1,$$

where the stages  $k_1, k_2, k_3$  and  $k_4$  are given by [31]:

$$k_1 = f(t_n, y_n), \\ k_2 = f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{\Delta t}{2}k_1\right), \\ k_3 = f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{\Delta t}{2}k_2\right), \\ k_4 = f(t_n + \Delta t, y_n + \Delta t k_3).$$

This means that to use ERK4 to solve the 1D spherical LSWE, we must bring the equations in the same form as the IVP (3.3.3). Recall, that the linearized 1D LSWE in spherical coordinates are given by

$$\frac{\partial h'}{\partial t} = -\frac{h_0}{r \cos \phi} \frac{\partial u}{\partial \theta}, \quad (3.3.6a)$$

$$\frac{\partial u}{\partial t} = -\frac{g}{r \cos \phi} \frac{\partial h'}{\partial \theta}, \quad (3.3.6b)$$

where  $h'$  is the perturbation in water height,  $u$  is the velocity in the  $\theta$ -direction,  $h_0$  is the mean water depth,  $g$  is the gravitational acceleration,  $r$  is the Earth's radius, and  $\phi$  is the fixed latitude. Since the ERK4 method solves ODEs, and the LSWE are PDEs, we must use the method of lines (MOL) to bring the LSWE in the form of the IVP (3.3.3). The MOL is a technique for solving PDEs by discretizing all variables but one (usually time), transforming the PDE into a system of ODEs. The system of ODEs can then be solved using standard ODE solvers, such as the ERK4 method. We discretize the spatial dimension  $\theta$  into  $N$  cells, and approximate the spatial derivatives using central differences:

$$\frac{\partial u}{\partial \theta}|_{\theta=\theta_i} = \frac{u_{i+1} - u_{i-1}}{2\Delta\theta}, \quad (3.3.7a)$$

$$\frac{\partial h'}{\partial \theta}|_{\theta=\theta_i} = \frac{h'_{i+1} - h'_{i-1}}{2\Delta\theta}, \quad (3.3.7b)$$

for  $i = 0, 1, \dots, N - 1$ . To perform the time integration, we use the ERK4 method. This method is particularly effective for time-stepping and is an alternative approach to traditional time integration schemes. RK4 provides higher accuracy in time while relying on the same spatial discretization, such as the finite volume method (FVM). Substituting the spatial derivatives (3.3.7) into the LSWE (3.3.6) gives the system of ODEs:

$$\frac{dh'}{dt} = -\frac{h_0}{r \cos \phi} \frac{u_{i+1} - u_{i-1}}{2\Delta\theta}, \quad (3.3.8a)$$

$$\frac{du}{dt} = -\frac{g}{r \cos \phi} \frac{h'_{i+1} - h'_{i-1}}{2\Delta\theta}. \quad (3.3.8b)$$

Hence, we can write the 1D spherical LSWE in the form of the IVP (3.3.3) by defining

$$y(t) = \begin{bmatrix} h' \\ u \end{bmatrix}, \quad f(y(t), t) = \begin{bmatrix} -\frac{h_0}{r \cos \phi} & \frac{u_{i+1}-u_{i-1}}{2\Delta\theta} \\ -\frac{g}{r \cos \phi} & \frac{h'_{i+1}-h'_{i-1}}{2\Delta\theta} \end{bmatrix} \quad \text{for } i = 0, 1, \dots, N-1. \quad (3.3.9)$$

As we consider a circle as the domain, we must impose periodic boundary conditions. We do this by setting  $h'_{-1} = h'_{N-1}$ ,  $h'_N = h'_0$ ,  $u_{-1} = u_{N-1}$ , and  $u_N = u_0$ . The initial conditions for  $y$  are a Gaussian perturbation in water height and zero velocity:

$$y_0 = \begin{bmatrix} \exp(-\frac{(\theta_i-\theta_0)^2}{\sigma^2}) \\ 0 \end{bmatrix}, \quad i = 0, 1, \dots, N-1. \quad (3.3.10)$$

where  $\theta_0$  is the center of the Gaussian, and  $\sigma$  is the width of the Gaussian. This means, that we can employ the four-stage time-stepping scheme in the ERK4 method to update the solution for  $h'$  and  $u$ . The final values of  $h'$  and  $u$  are obtained by combining the contributions from all four stages, by using the general update formula:

$$\mathbf{W}_i^{n+1} = \mathbf{W}_i^n + \frac{\Delta t}{\Delta\theta} \frac{1}{6} (k_1 + 2k_2 + 2k_3 + k_4),$$

where the state variable  $\mathbf{W}_i = \begin{bmatrix} h'_i \\ u_i \end{bmatrix}$  represents the perturbation height and velocity at the  $i$ 'th cell. The intermediate stages  $k_1, k_2, k_3$  and  $k_4$  are computed at each time step. The scheme integrates the equations in time while relying on flux differences to update the solution. This approach allows the method to achieve high temporal accuracy while maintaining the spatial resolution provided by the finite volume discretization.

## 3.4 The Riemann problem

We will now define the Riemann problem, since it plays a crucial role in the finite volume method. The Riemann problem for the SWE with a zero source term, i.e.  $\mathbf{S}(\mathbf{U}) = 0$ , is defined as the initial-value problem (IVP) [7]:

$$\begin{aligned} \text{PDEs: } & \mathbf{U}_t + \mathbf{F}(\mathbf{U})_x = 0, \\ \text{ICs: } & \mathbf{U}(x, 0) = \begin{cases} \mathbf{U}_L, & \text{if } x < x_0, \\ \mathbf{U}_R, & \text{if } x > x_0. \end{cases} \end{aligned} \quad (3.4.1)$$

The vectors  $\mathbf{U}$  and  $\mathbf{F}(\mathbf{U})$  in (3.4.1) are given by

$$\mathbf{U} = \begin{bmatrix} h \\ hu \\ hv \end{bmatrix}, \quad \mathbf{F}(\mathbf{U}) = \begin{bmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \\ hvu \end{bmatrix}, \quad (3.4.2)$$

and the initial conditions  $\mathbf{U}_L$  and  $\mathbf{U}_R$  are

$$\mathbf{U}_L = \begin{bmatrix} h_L \\ h_L u_L \\ h_L v_L \end{bmatrix}, \quad \mathbf{U}_R = \begin{bmatrix} h_R \\ h_R u_R \\ h_R v_R \end{bmatrix},$$

which represents the conditions at time  $t = 0$  s in the left and right states of  $x = x_0$  m. The function  $\mathbf{U}$  is piecewise constant, with a discontinuity at  $x = x_0$  m. The Riemann problem is solved at the interface between two cells, where the left and right states are known, and the solution is used to update the solution at each timestep in the finite volume method. The Riemann problem can be solved either exactly or approximately. When solving the Riemann problem exactly we distinguish between what we call a wet bed and a dry bed. A wet bed is the case where the water depth is positive everywhere, whereas a dry bed is the case where the water depth is zero in some cells. In this project, we focus on approximate Riemann solvers, which are able to solve the Riemann problem with high accuracy and efficiency. Various approximate Riemann solvers exist, based on finding an approximate solution to the Riemann problem, and some of these solvers will be considered in the next section. An interesting example of a Riemann problem is the so-called dam break problem, which is presented next.

### 3.4.1 The Dam-Break problem

We now introduce the dam-break problem, a scenario of significant physical interest. This problem models the sudden release of water following the collapse of a dam, making it highly relevant for studying natural disasters such as floods and tsunamis. The problem also serves as a classic test case for numerical methods, to test their ability to capture discontinuities in the solution. The dam-break problem is a special case of the Riemann problem (3.4.1). The difference is that in the dam-break problem, the initial velocity components,  $u_L, u_R, v_L$  and  $v_R$ , are zero, whereas in the Riemann problem they are allowed to be distinct from zero. The initial setup is visualized in Figure 3.4.

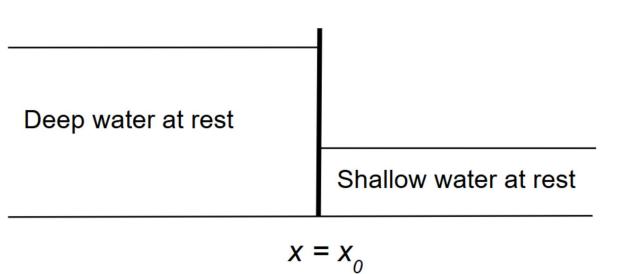


Figure 3.4: Initial conditions for the dam-break problem. An infinitely thin wall at  $x = x_0$  divides two sections of water with different water levels. Illustration modified from [7].

We can use the shallow water equations to model the flow of water in the dam-break problem, approximately, if we assume that the wall collapses instantaneously at  $t = 0$  s. In this project we solve both the dam-break problem and cases of the Riemann problem, where the initial fluid velocity is nonzero. The results are presented in chapter 6.

### 3.4.2 Wave decomposition in the Riemann problem

To get a better understanding of the flow in shallow water, we provide some very short background information about the wave structures in the solution of the Riemann problem. In general, the wave structure in the solution of the Riemann problem (3.4.1) consists of three wave families separating four regions. The wave families are denoted  $W_1, W_2, W_3$  and the regions are the spaces between the wave families, denoted by  $R_0, R_1, R_2$  and  $R_3$ . The wave structure is illustrated in Figure 3.5.

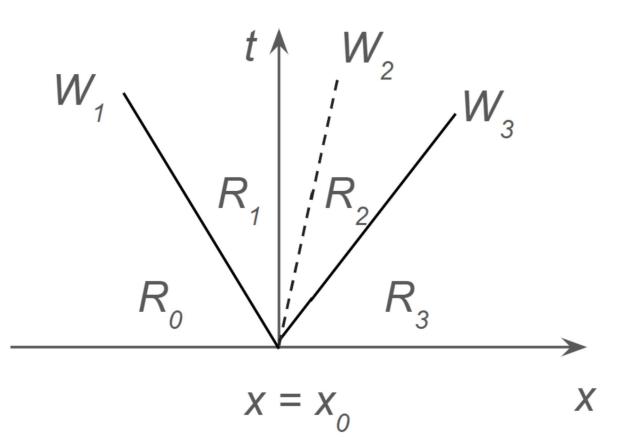


Figure 3.5: General wave structure in the solution of the Riemann problem. The wave families are denoted by  $W_1, W_2$  and  $W_3$ , and the regions between the wave families are denoted by  $R_1, R_2, R_3$  and  $R_4$ . The star region consists of the regions  $R_1$  and  $R_2$ .

From Figure 3.5 we see how the solution consists of three waves,  $W_1, W_2$  and  $W_3$ , also referred to as the left wave, middle wave and right wave. The left and right waves are either shock waves or rarefaction waves, and correspond to the one-dimensional shallow water equations. The middle wave arises from the  $y$ -momentum equation in (3.4.1), and is always a shear wave. The region between the left and right wave is called the star region and is interesting, since we do not know the solution in this region. The star region is divided into two subregions  $R_1$  and  $R_2$ . The states in the regions are (from left to right)  $U_L, U_{*L}, U_{*R}$  and  $U_R$ , where  $U_L$  and  $U_R$  are known, as these are the initial conditions. The states  $U_{*L}$  and  $U_{*R}$  are in the regions  $R_1$  and  $R_2$ , i.e., the star region and are unknown. In the star region, we use  $h_*$  to denote the water depth and  $u_*$  to denote the velocity. We can use  $h_*$  to determine whether the left and right waves are shock waves or rarefaction waves. Since we are considering the wet bed case, we can use the following characteristic:

$$\begin{cases} \text{The left wave is a shock wave} & \text{if } h_* > h_L, \\ \text{The left wave is a rarefaction wave} & \text{if } h_* \leq h_L, \end{cases} \quad (3.4.3)$$

and similarly for the right wave:

$$\begin{cases} \text{The right wave is a shock wave} & \text{if } h_* > h_R, \\ \text{The right wave is a rarefaction wave} & \text{if } h_* \leq h_R. \end{cases} \quad (3.4.4)$$

A shock wave is characterized by a discontinuity in the solution. On each side of the shock wave, the water properties, such as height and velocity, differ significantly. In contrast, a rarefaction wave represents a smooth transition in the solution. The water height and velocity change gradually across the wave, and the properties are more similar on each side of the wave, without the sharp edges seen in a shock wave. In the solution of the Riemann problem (3.4.1) there are four possible wave patterns outcomes, which are combinations of shock waves and rarefaction waves. The left and right waves are either shock waves or rarefaction waves. The four possible wave patterns are as follows:

- (a) Left rarefaction, right shock,
- (b) Left shock, right rarefaction,
- (c) Both left and right rarefaction,
- (d) Both left and right shock.

In the example of the dam-break problem, with initial conditions as in Figure 3.4, the solution for the water height  $h$  at time  $t = t_*$  is shown in Figure 3.6.

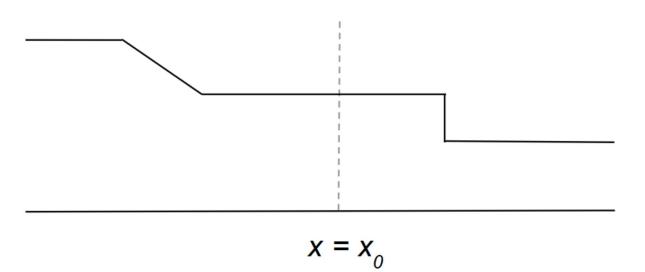


Figure 3.6: Water depth at time  $t = t_*$  in the dam-break problem. The solution consists of a left rarefaction wave and a right shock wave.

From Figure 3.6 we see that the solution consists of a left rarefaction wave and a right shock wave. We can see that after the wall collapses, the water flows to the right, meaning the water height in the star/middle region,  $h_*$ , will be between  $h_L$  and  $h_R$ . Based on the definitions (3.4.3) and (3.4.4), we can therefore conclude that the left wave is a rarefaction wave and the right wave is a shock wave. The right shock wave is characterized by a discontinuity in the solution and a high speed, whereas the left rarefaction wave is characterized by a more smooth transition in the solution and a lower speed.

In the finite volume framework, the Riemann solver is used to compute the flux across the interface between two cells, considering the left and right states. This way, we ensure that the numerical method handles the discontinuities in the solution correctly, and that the solution is stable and accurate.

### 3.5 Numerical fluxes

In this section we will study the numerical fluxes used to solve the SWE in 1D. At each cell interface, we need to solve the Riemann problem (3.4.1) to find the numerical flux. There are several numerical fluxes that can be used to solve the local Riemann problem, and we will consider some of them in this section. The fluxes we will consider are the Godunov method with an exact Riemann solver, the HLL, HLLC, Rusanov, Lax-Friedrichs, Lax-Wendroff and FORCE fluxes. All of them are later implemented and tested in the numerical experiments in chapter 6.

#### Godunov method with exact Riemann solver

We consider the Godunov Upwind method, which is a first-order accurate method to solve non-linear systems of hyperbolic conservation laws [7]. In the method we solve the non-linear Riemann problem at each cell interface. The Godunov flux is given by

$$\mathbf{F}_{i+\frac{1}{2}} = \mathbf{F}(\mathbf{U}_{i+\frac{1}{2}}),$$

meaning that we solve the Riemann problem exactly to find  $h^*$  and  $u^*$ , and then use these values to compute the flux as

$$\mathbf{F}_{i+\frac{1}{2}} = \begin{bmatrix} h^* u^* \\ h^*(u^*)^2 + \frac{1}{2}g(h^*)^2 \end{bmatrix}.$$

## HLL

The HLL (Harten, Lax and van Leer) approach assumes a two-wave structure of the Riemann problem. The solver is based on the data  $\mathbf{U}_L := \mathbf{U}_i^n$ ,  $\mathbf{U}_R := \mathbf{U}_{i+1}^n$  and fluxes  $\mathbf{F}_L := \mathbf{F}(\mathbf{U}_L)$ ,  $\mathbf{F}_R := \mathbf{F}(\mathbf{U}_R)$ . The HLL flux is given by

$$\mathbf{F}_{i+\frac{1}{2}} = \begin{cases} \mathbf{F}_L & \text{if } S_L \geq 0, \\ \mathbf{F}^{HLL} \equiv \frac{S_R \mathbf{F}_L - S_L \mathbf{F}_R + S_L S_R (\mathbf{U}_R - \mathbf{U}_L)}{S_R - S_L} & \text{if } S_L \leq 0 \leq S_R, \\ \mathbf{F}_R & \text{if } S_R \leq 0. \end{cases} \quad (3.5.1)$$

The wave speeds  $S_L$  and  $S_R$  must be estimated in some way, and one possibility is to use

$$S_L = u_L - a_L q_L, \quad S_R = u_R + a_R q_R,$$

where  $a_L = \sqrt{gh_L}$ ,  $a_R = \sqrt{gh_R}$  and  $q_K (K = L, R)$  is given by

$$q_K = \begin{cases} \sqrt{\frac{1}{2} \left( \frac{(\hat{h} + h_K)\hat{h}}{h_K^2} \right)} & \text{if } \hat{h} > h_K, \\ 1 & \text{if } \hat{h} \leq h_K. \end{cases}$$

Here  $\hat{h}$  is an estimate for the water depth in the star region,  $h_*$ . In the two-rarefaction Riemann Solver, the water depth  $h$  in the star region is given by

$$h_* = \frac{1}{g} \left( \frac{1}{2} (a_L + a_R) + \frac{1}{4} (u_L - u_R) \right)^2, \quad (3.5.2)$$

which is what we use in this project for  $\hat{h}$  in the HLL solver. Since this is a two-wave model, it is complete for one dimensional problems, but for the augmented system of equations in two dimensions, the HLL solver is not complete, as it ignores the middle wave, the shear wave. This motivates the use of the HLLC solver, which is a modification of the HLL solver.

## HLLC

The HLLC (Harten, Lax, van Leer, Contact) solver is an extension of the HLL solver, which includes the middle wave, i.e., it is a three-wave model. In addition to the wave speeds  $S_L$  and  $S_R$ , the HLLC solver also requires the speed of the middle wave  $S^*$ . We can write the HLLC numerical flux as

$$\mathbf{F}_{i+\frac{1}{2}}^{HLLC} = \begin{cases} \mathbf{F}_L & \text{if } 0 \leq S_L, \\ \mathbf{F}_{*L} & \text{if } S_L \leq 0 \leq S^*, \\ \mathbf{F}_{*R} & \text{if } S^* \leq 0 \leq S_R, \\ \mathbf{F}_R & \text{if } S_R \leq 0. \end{cases}$$

The fluxes  $\mathbf{F}_{*L}$  and  $\mathbf{F}_{*R}$  are given by

$$\begin{aligned} \mathbf{F}_{*L} &= \mathbf{F}_L + S_L (\mathbf{U}_L - \mathbf{U}_{*L}), \\ \mathbf{F}_{*R} &= \mathbf{F}_R + S_R (\mathbf{U}_R - \mathbf{U}_{*R}), \end{aligned}$$

and the middle states  $\mathbf{U}_{*L}$  and  $\mathbf{U}_{*R}$  are given by

$$U_{*K} = h_K \left( \frac{S_K - u_K}{S_K - S_*} \right) \begin{bmatrix} 1 \\ S_* \\ \psi_K \end{bmatrix}.$$

The function  $\psi_K$  can represent either a passive scalar  $\psi(x, t)$  or the velocity component  $v(x, t)$  if we consider the two-dimensional shallow water equations. Mathematically  $\psi(x, t)$  and  $v(x, t)$  behave identically. An estimate for the middle wave speed  $S^*$  can be calculated as

$$S^* = \frac{S_L h_R (u_R - S_R) - S_R h_L (u_L - S_L)}{h_R (u_R - S_R) - h_L (u_L - S_L)},$$

where  $S_L$  and  $S_R$  are the wave speeds of the left and right waves, respectively.

## Rusanov

The Rusanov flux uses the HLL framework, but with a different choice of wave speeds. To obtain the flux, we assume that an estimate  $S^+$  for the positive wave speed is available. Then we set

$$S_L = -S^+, \quad S_R = S^+. \quad (3.5.3)$$

By substituting (3.5.3) into the  $\mathbf{F}^{HLL}$  in (3.5.1), we obtain the Rusanov flux as

$$\mathbf{F}_{i+\frac{1}{2}}^{Rus} = \frac{1}{2} (\mathbf{F}_L + \mathbf{F}_R) - \frac{1}{2} S^+ (\mathbf{U}_R - \mathbf{U}_L), \quad (3.5.4)$$

where a simple estimate for the wave speed  $S^+$  is given by

$$S^+ = \max(|S_L|, |S_R|).$$

There are some requirement for  $S^+$  in (3.5.4) to ensure stability. It must hold that

$$S^+ \leq \frac{\Delta x}{\Delta t},$$

where  $\frac{\Delta x}{\Delta t}$  is called the mesh speed.

## Lax-Friedrichs

In the Lax-Friedrichs method, we use the Rusanov flux, but with a different choice of wave speed. That is, we set the wave speed  $S^+$  as the largest possible speed, while still ensuring stability, i.e.,

$$S^+ = \frac{\Delta x}{\Delta t}. \quad (3.5.5)$$

By inserting the wave speed (3.5.5) into the Rusanov flux (3.5.4), we obtain the Lax-Friedrichs flux as

$$\mathbf{F}_{i+\frac{1}{2}}^{LF} = \frac{1}{2} (\mathbf{F}_L + \mathbf{F}_R) - \frac{1}{2} \frac{\Delta x}{\Delta t} (\mathbf{U}_R - \mathbf{U}_L),$$

where  $\mathbf{F}_L = \mathbf{F}(\mathbf{U}_L)$  and  $\mathbf{F}_R = \mathbf{F}(\mathbf{U}_R)$ . The Lax-Friedrichs method is a centred method, which is first-order accurate.

## Lax-Wendroff

There are several versions of the Lax-Wendroff flux, but in this thesis we will use the following flux:

$$\begin{aligned} \mathbf{U}_{i+\frac{1}{2}}^{LW} &= \frac{1}{2} (\mathbf{U}_L + \mathbf{U}_R) - \frac{1}{2} \frac{\Delta t}{\Delta x} (\mathbf{F}_R - \mathbf{F}_L), \\ \mathbf{F}_{i+\frac{1}{2}}^{LW} &= \mathbf{F}(\mathbf{U})_{i+\frac{1}{2}}^{LW}. \end{aligned} \quad (3.5.6)$$

The Lax-Wendroff method is a centred method, which is second-order accurate in space and time.

## FORCE

The FORCE scheme (First-Order Centred) is a combination of Lax-Friedrichs and Lax-Wendroff fluxes. The numerical flux is given by

$$\mathbf{F}_{i+\frac{1}{2}}^{FO} = \frac{1}{2} \left( \mathbf{F}_{i+\frac{1}{2}}^{LF} + \mathbf{F}_{i+\frac{1}{2}}^{LW} \right),$$

where  $\mathbf{F}_{i+\frac{1}{2}}^{LF}$  is the Lax-Friedrichs flux and  $\mathbf{F}_{i+\frac{1}{2}}^{LW}$  is the Lax-Wendroff flux. The FORCE scheme is first-order accurate. It is possible to extend the FORCE scheme to multiple dimensions on structured meshes by using dimensional splitting.

## Chapter 4

# Neural Networks and Fourier Neural Operators

So far, we have studied numerical methods for solving partial differential equations (PDEs). The finite volume method (FVM), along with other numerical solvers such as the finite difference method (FDM) and finite element method (FEM), solves PDEs by discretizing the domain into a grid. A finer grid improves the accuracy of the solution but also increases the computational cost, creating a trade-off between accuracy and efficiency. Complex PDEs often require a fine grid to accurately capture the solution, which can be computationally expensive. This motivates the use of data-driven methods, which have shown great potential in handling large data sets and learning complex patterns in the data.

In this chapter, we introduce the use of data-driven methods for solving PDEs. The hope of these methods is that they are able to reduce computational costs while maintaining high accuracy by learning the underlying dynamics of the solution, for making predictions for the solution of the SWE outside the training data. We will introduce the concepts of convolutional neural networks (CNNs) and Fourier neural operators (FNOs).

### 4.1 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a specialized class of artificial neural networks (ANNs) designed to process and analyze data with a grid-like topology, such as images or time-series data represented by 2D grids. CNNs excel at extracting spatial features from data through the use of convolutional layers, which apply learnable filters to detect patterns such as edges, shapes or textures. These layers are typically followed by pooling layers for dimensionality reduction and fully connected layers for classification or regression tasks. A key advantage of CNNs is their ability to reduce the number of parameters compared to fully connected neural networks by sharing weights across spatial regions, making them computationally efficient and less prone to overfitting when working with large inputs. Although CNNs are traditionally used for image recognition tasks, their architecture is adaptable for time-series analysis, especially when the data is structured spatially or sequentially [21].

In this project, CNNs are used to solve the shallow water equations, by training on the solution data generated by the FVM solver. We aim to learn the underlying dynamics of the system by training the CNN on sequences of input-output pairs, where the input  $\mathbf{x}(t)$  is the state of the system at time  $t$  and the output is the state at time  $t + \Delta t$ , denoted by  $\mathbf{x}(t + \Delta t)$ . More specific, this means that the CNN model is trained on input-output pairs of

sequences of data, with a sequence length of  $n$ , illustrated in the vectors below:

$$\begin{bmatrix} x_i \\ x_{i+1} \\ \vdots \\ x_{i+n-1} \end{bmatrix} \rightarrow \begin{bmatrix} x_{i+1} \\ x_{i+2} \\ \vdots \\ x_{i+n} \end{bmatrix}, \text{ for } i = 0, 1, \dots, N-n,$$

where  $x_i$  is the state of the system at time  $i$ . The reason for these input-output pairs, is that we want train the CNN to predict the state of the system at the next time step, based on the current state and the  $n - 1$  previous states. We also want to predict the solution of the SWE at future time steps, which is why we use sequences of input data. Hence, the network is trained to construct a flow map. The flow map  $\Phi : \mathbb{R}^n \rightarrow \mathbb{R}^n$  is defined such that, for all  $x \in X$  and all  $t \in \mathbb{R}$ :

$$\Phi(x_0) = x(t),$$

where  $x(t)$  is the solution of the PDEs with initial condition  $x(0) = x_0$ . The flow map satisfies

$$\Phi(\Phi(x, t), s) = \Phi(x, s + t),$$

A CNN can approximate the flow map  $\Phi$  by training on data that pairs a state  $\mathbf{x}(t)$  with its state at later times  $\mathbf{x}(t + \Delta t)$ . The goal is to learn the mapping:

$$x_{i+1} = \Phi_{CNN}(x_i),$$

where  $\Phi_{CNN}$  is the CNNS' approximation of the flow map.

Meaning the network takes sequences of input data, and predicts the corresponding output data. The output of the CNN is the solution at the next time step, effectively learning the dynamics of the SWE through the time-series data. This setup allows the model to capture both spatial and temporal dependencies in the data, leveraging the CNN's ability to learn localized features while processing sequential information. A pro of CNN's is that they are efficient. By processing data in parallel using convolutional layers, the CNN efficiently handles large datasets without requiring excessive computational resources. Another pro is the adaptability. The model's ability to learn from sequential data makes it adaptable for time-series predictions in dynamic systems like the shallow water equations. A con is the data representation. Representing time-series data as sequences may require preprocessing, which can introduce complexity or a potential loss of information. We will test and evaluate the performance of the CNN model in chapter 7.

## 4.2 Fourier Neural Operators

In this section, we introduce the concept of Fourier Neural Operators (FNOs). The theory and method described here are based on the paper [16]. This method is special, as where a classical neural network primarily is focused on learning a mapping between finite-dimensional spaces, FNOs approximate mappings between infinite-dimensional function spaces. The goal with the FNO is to learn a mapping between two infinite dimensional spaces from a finite collection of input-output pairs. Consider the operator  $G : A \rightarrow U$ , which maps from an infinite-dimensional function space  $A$  to another infinite-dimensional function space  $U$ . We aim to approximate the exact operator  $G$  by constructing the map

$$G_\theta : A \mapsto U, \quad \theta \in \Theta, \tag{4.2.1}$$

where  $\Theta$  is a finite-dimensional parameter space. Consider the functions  $a \in A$  and  $u \in U$ . We assume access to data in the form of pointwise evaluations of these functions, i.e., we have access to the observations  $\{a_j, u_j\}_{j=0}^{N-1}$ , in a domain  $D \subset \mathbb{R}^d$ , which is a bounded open set. The observations are time shifted, meaning that  $u_j = G(a_j)$ , where  $G$  is the true operator, and that  $u_i = a_{i+1}$  for all  $i = 0, 1, \dots, N - 1$ . That is, our goal is to approximate the

mapping  $a_j \mapsto u_j$  for all  $j = 0, 1, \dots, N - 1$ . The first step is to create  $v_0(x) = P(a(x))$  by the input layer  $P$ , which is typically a shallow fully-connected neural network. The neural operator is iterative, meaning we apply several iterations of updates to obtain  $v_1, v_2, \dots$  up to  $v_T$ . In the context of this project we have that  $a(x) = v_t(x)$  and  $u(x) = v_{t+1}(x)$ . An update  $v_t \mapsto v_{t+1}$  is defined as

$$v_{t+1}(x) := \sigma(Wv_t(x) + (\mathcal{K}(a; \phi)v_t)(x)), \quad \forall x \in D, \quad (4.2.2)$$

where  $W : \mathbb{R}^{d_v} \rightarrow \mathbb{R}^{d_v}$  is a linear transformation, and  $\sigma : \mathbb{R} \rightarrow \mathbb{R}$  is a non-linear activation function. The output  $u(x) = Q(v_T(x))$  is the final update  $v_T$  transformed by the output layer  $Q$ , that transforms  $v_T$  into the correct output dimensions. That is, like for the CNN, we collect state pairs  $\{v_t, v_{t+1}\}_{j=0}^N$  for training a flowmap, in this case the map  $G_\theta$ . Feeding this data into the model allows it to learn the flow map for the system.

There are various types of operators, but the core of the Fourier neural operator lies in its kernel function  $\mathcal{K}$ . We define the Fourier integral operator  $\mathcal{K}$  as

$$(\mathcal{K}(\phi)v_t)(x) := \mathcal{F}^{-1}(R_\phi \cdot (\mathcal{F}v_t))(x), \quad \forall x \in D, \quad (4.2.3)$$

where  $\mathcal{F}$  is the Fourier transform,  $\mathcal{F}^{-1}$  is the inverse Fourier transform, and  $R$  is the linear transformation applied on the lower Fourier modes. Recall that the Fourier transform of a function  $f(t)$  is defined as

$$\mathcal{F}(f(t)) = \hat{f}(s) = \int_{-\infty}^{\infty} e^{-2\pi i s t} f(t) dt, \quad (4.2.4)$$

and correspondingly the inverse Fourier transform of a function is

$$\mathcal{F}^{-1}(\hat{f}(s)) = f(t) = \int_{-\infty}^{\infty} e^{2\pi i s t} \hat{f}(s) ds.$$

By transforming the data into the Fourier domain, FNOs can take advantage of the periodicity and smoothness properties of the Fourier basis, which simplifies the learning process for functions defined over continuous domains. This approach leverages the fact that differentiation with respect to time is equivalent to multiplication in the Fourier domain, not to be interchanged with the convolution theorem. CITE. The network architecture for the FNO model is illustrated in Figure 4.1.

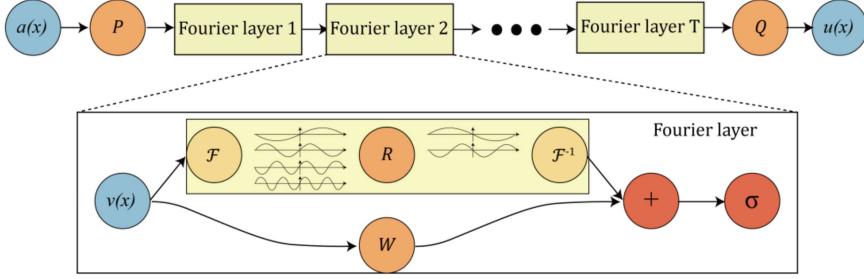


Figure 4.1: An overview of the network architecture with several Fourier layers. Illustration from [16].  
 Top: Overall structure with input function  $a$ , input layer  $P$ , several Fourier layers, output layer  $Q$  and output function  $u$ .  
 Bottom: A Fourier layer consists of two parallel paths. Top is a Fourier transformation  $\mathcal{F}$ , a linear transformation  $R$  and an inverse Fourier transformation  $\mathcal{F}^{-1}$ . The bottom path consists of a linear transformation  $W$ . The parts meet and undertake an activation function  $\sigma$ .

From the top in Figure 4.1 we see that the network consists of an input function  $a(x)$ , an input layer  $P$ , several Fourier layers, an output layer  $Q$  and some output function  $u(x)$ . In the bottom of Figure 4.1 we see the structure

of a Fourier layer, which consists of two parallel paths. In the top path, the data undergoes a Fourier transform  $\mathcal{F}$ , decomposing it into a sum of Fourier basis functions (sines and cosines) with varying frequencies, amplitudes and phases. A linear transformation  $R$  is applied to filter out the higher Fourier modes, as illustrated in the figure, where high-frequency components are removed. When implementing the model, we choose the number of Fourier modes to retain, depending on how much information we want to preserve. Retaining more modes keeps more information, but may also introduce more noise and oscillations. After filtering, the inverse Fourier transform  $\mathcal{F}^{-1}$  is applied to reconstruct the data in its original form. The bottom path involves a linear transformation  $W$ , and the two paths merge before applying a non-linear activation function  $\sigma$ .

A key advantage of FNOs is that since they learn mapping between function spaces, they are not constrained by a specific grid or mesh. This allows them to transfer solutions across different grids, a process known as zero-shot super-resolution. The learned operator can generalize from the coarse grid to the fine grid without needing to be retrained, making it grid-independent. This is a major advantage because it reduces the computational cost associated with fine-grid simulations. In chapter 7, we will evaluate the performance of the implemented FNO model in this context, testing its ability to maintain high accuracy when making predictions on finer grids.

## Multistep prediction

Another potential use of the FNO model is long-term prediction, where the model is trained to predict the solution of the SWE at multiple future time steps. We aim to develop a multi-step prediction model that can predict a specified number of time steps ahead. The original form of the FNO can predict one time step ahead. However, for many applications, predicting multiple time steps ahead is more useful. The first, naive approach to multi-step prediction is to use the model to predict the solution at the next time step, then use the predicted solution as input to predict the solution at the next time step, and so on. However, as one can imagine this approach quite fast lead to inaccurate predictions, as errors accumulate over time.

To improve the accuracy of multi-step predictions, we train the model on sequences of input data, where the input data consists of several previous time steps. The model is trained to predict the output state based on data from several previous time steps, with a specified sequence length. We will conduct experiments to determine the optimal sequence length for our model. That is, we now train the model on input-output pairs of sequences of data, with a sequence length of  $n$ , as illustrated in the vectors below:

$$\begin{bmatrix} v_{t-n} \\ v_{t-n+1} \\ \vdots \\ v_{t-1} \\ v_t \end{bmatrix} \rightarrow \begin{bmatrix} v_{t-n+1} \\ v_{t-n+2} \\ \vdots \\ v_t \\ v_{t+1} \end{bmatrix}, \text{ for } t = 0, 1, \dots, N - n.$$

Training the model on sequences of input data allows it to learn the dynamics of the system over time, capturing both short-term and long-term dependencies in the data. It also makes the model more accurate by providing more context for making predictions, ensuring that recent predictions do not disproportionately influence future predictions. When making multi-step predictions, the model's output is added to the input as the newest data point, replacing the oldest data. This process is repeated until predictions for the desired number of future time steps are made. The process is illustrated in Figure 4.2.

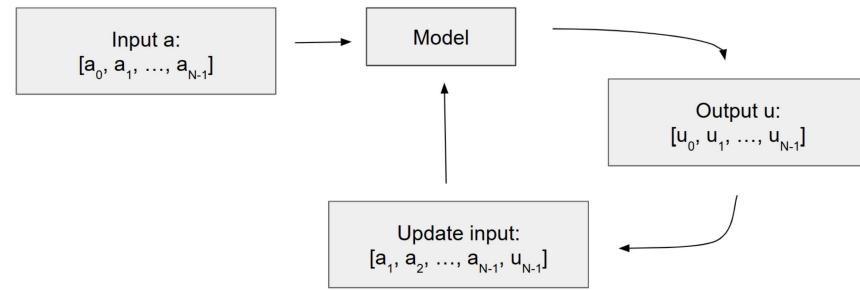


Figure 4.2: Flowchart of the multi-step prediction process.

The results of the multi-step prediction model will be presented in chapter 7.

# Chapter 5

## Data generation

In this chapter, we outline the process of generating the data required for the data-driven methods. This step is also a big part of the project, as we generate all the data ourselves using our numerical solver. This step is critical, as the quality and relevance of the data directly impact the training and performance of the models. We begin by detailing the data generation process for the 1D SWE using the FVM with fluxes determined from an exact Riemann solver. Next, we describe the approach for generating data for the 1D SWE in spherical coordinates. Finally, we present the methodology for data generation in the context of the 2D SWE.

### 5.1 Data generation using the FVM

In this section we clarify the data generation using our numerical solver based on the Finite Volume Method (FVM). The FVM is used to solve the SWE in 1D, in 1D with spherical coordinates, and in 2D. We specify the initial conditions, the domain, and the parameters used in the data generation process.

#### 1D SWE with exact Riemann solver

In this section, we present how the so-called true solution is found in the code by solving the Riemann problem exactly. The true solution is found by solving the Riemann problem exact, with 200 cells, and distinguishing between the wetbed or drybed case, and also identifying the shock and rarefaction waves. First we caculate the wave speeds for the left and right states, respectively, as

$$c_L = \sqrt{gh_L}, \quad c_R = \sqrt{gh_R},$$

which are used to determine the critical water height  $h_{\text{crit}}$  as

$$h_{\text{crit}} = (u_R - u_L) - 2(c_L + c_R).$$

If either  $h_L \leq 0$  or  $h_R \leq 0$ , we are in a drybed case. If  $h_{\text{crit}} \geq 0$  it means the water depth is somehow critical, and we are in a drybed case, If none of the above conditions are met, we are in a wetbed case. Summarized:

$$\begin{cases} \text{Dry-bed case} & \text{if } h_L \leq 0, \quad h_R \leq 0 \text{ or } h_{\text{crit}} \geq 0, \\ \text{Wet-bed case} & \text{otherwise.} \end{cases}$$

For a dry bed case, we then identify where the dry is located, i.e., if the left side is dry, the right side is dry, or the middle is dry, and calculate the wave speeds accordingly. For a wet bed case, we compute the characteristics  $h_*$

and  $u_*$  for the star region. We then identify the shock and rarefaction waves, and calculate the wave speeds for the left and right states, respectively. The process is illustrated in Figure 5.1.

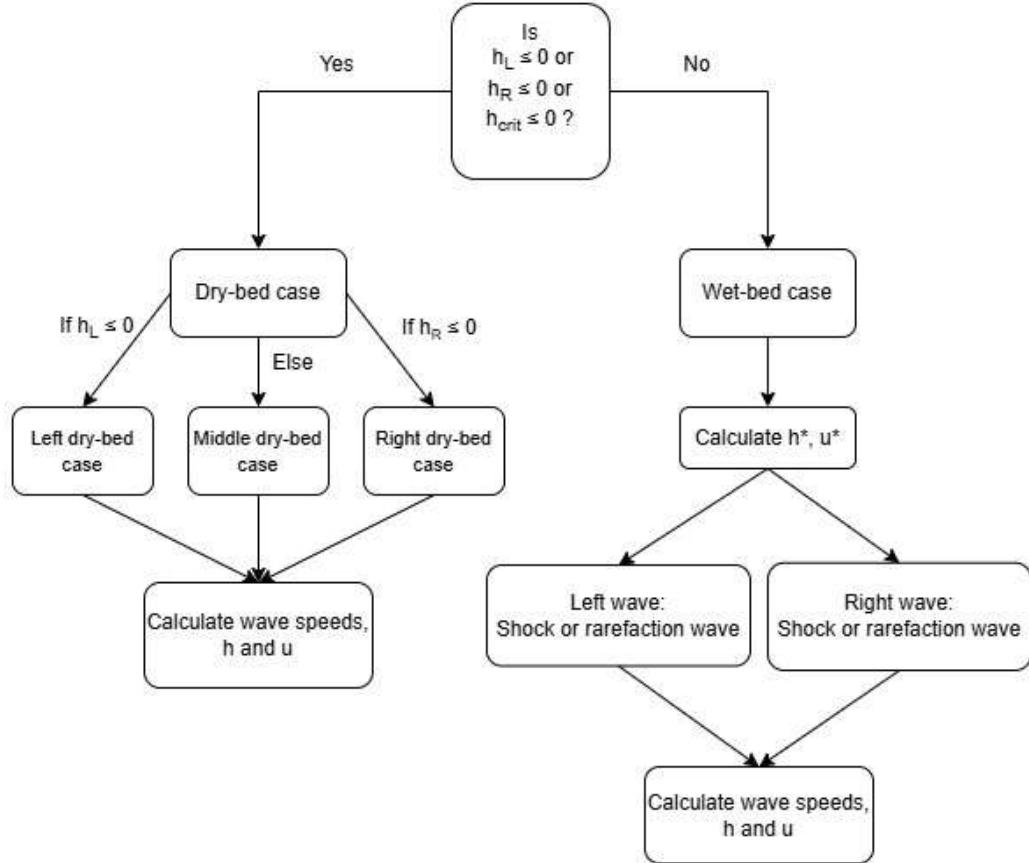


Figure 5.1: Flowchart for generating the solution.

When generating the data, we need an initial condition for the water height  $h$  and the velocity  $u$ . In this study, we use the Gaussian function as the initial condition for the water height  $h$ , that is, we use the function

$$h(x, 0) = a \exp\left(\frac{-(x - \mu)^2}{2\sigma^2}\right), \quad (5.1.1)$$

where  $a$  is the amplitude of the Gaussian,  $\mu$  is the mean value, and  $\sigma$  is the standard deviation. We are working on the domain  $x \in [0, 1]$  m and the parameters are set to  $g = 9.81 \text{m/s}^2$  and  $\sigma = 0.1$  m. The value of  $\mu$  is varied to generate different initial conditions, as seen in Figure 5.2.

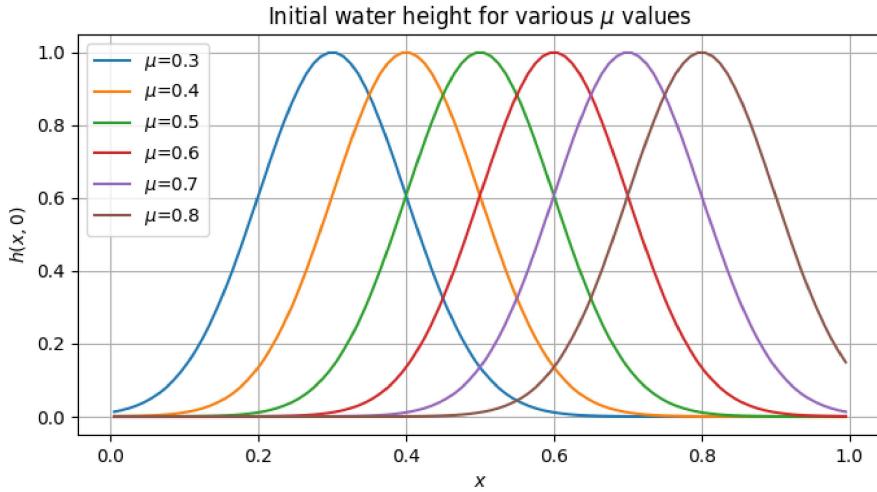


Figure 5.2: Initial conditions for the data generation.

For the initial velocity  $u$ , we set it to zero, i.e.,  $u(x, 0) = 0$  m. This means that the water is initially at rest. The solver is validated by comparing the results with known test cases, such as the dam break problem. We use a CFL number of 0.9 and variable time steps. We generate the data from  $t = 0.0$  to  $t_{\text{end}} = 1.0$ s.

## Truncation error

When generating data, it is essential to be aware of the truncation error. Truncation error arises from the approximation of the solution of the differential equation by the numerical method. Specifically, it is the difference between the exact solution and the numerical approximation. The critical question is: after a certain number of time steps, how significant is this error? If the error becomes too large, it raises concerns about the reliability of the generated data. Excessive truncation error could compromise the accuracy of the model trained on this data. Therefore, we must carefully evaluate and mitigate these risks to ensure the data's quality. To assess the truncation error, we generate a more accurate solution using a finer grid. This high-resolution solution serves as a reference for evaluating the numerical approximation. By comparing the high-resolution solution with the numerical solution, we gain insights into the error introduced by the approximation.

In this study, we generate a reference solution for  $N = 1000$  and compare it with the solution for  $N = 100$  at the final time step,  $t = 1.0$  s. The results are shown in Figure 5.3.

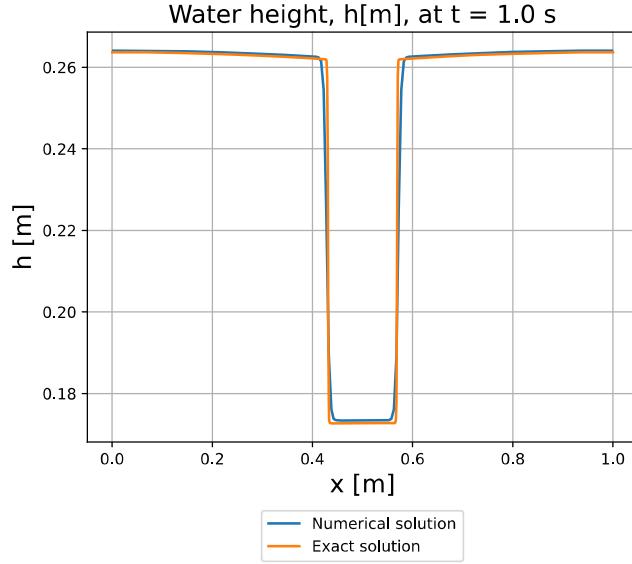


Figure 5.3: Truncation error for the 1D SWE.

From Figure 5.3, we observe that there is a small difference between the high-resolution solution and the numerical solution, as the high-resolution solution has a more discontinuous behavior. However, overall the two solutions are almost identical, indicating that the truncation error is negligible. This suggests that the data generated using the numerical solver is of high quality and can be used for training the data-driven models.

## 1D SWE in spherical coordinates

We also consider the spherical shallow water equations in a 1D setting, focusing on the linearized SWE on a circular domain. The length of the domain corresponds to a full circle,  $L = 2\pi$  radians and is discretized into  $N = 500$  points. The initial condition for the water height  $h$  is specified as a Gaussian function wrapped around the circle, expressed as:

$$h(\theta, 0) = a \exp \left( \frac{-(\theta - \mu)^2}{2\sigma^2} \right), \quad (5.1.2)$$

where  $h$  is the water height in meters,  $a$  is the amplitude of the Gaussian,  $\mu$  is the mean value, and  $\sigma$  is the standard deviation. The parameters are  $a = 1$  and  $\mu = \frac{\pi}{4}$ . We generate data for varying values of  $\sigma$  to investigate the effect of the standard deviation on the model performance. The data is generated for  $\sigma = \frac{\pi}{8}$ ,  $\sigma = \frac{\pi}{16}$  and  $\sigma = \frac{\pi}{32}$ . The initial velocity  $u$  is set to zero, i.e.,  $u(\theta, 0) = 0$  m. The time step size is fixed and set to  $\Delta t = 0.0025$  s. The initial conditions can be seen in Figure 5.4.

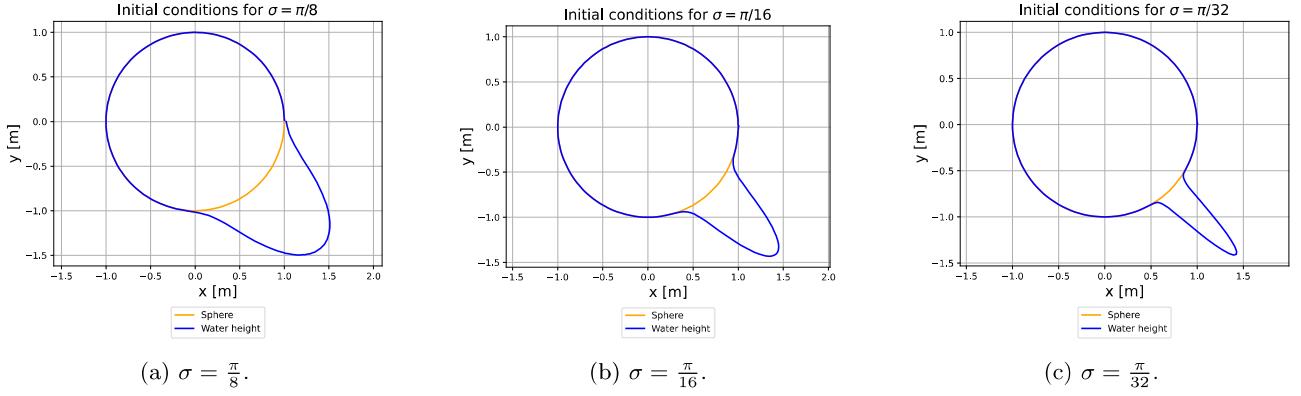


Figure 5.4: Initial conditions for the 1D linearized shallow water equations in spherical coordinates for different  $\sigma$  values.

From Figure 5.4, we observe that the standard deviation  $\sigma$  affects the width of the Gaussian function. The smaller the  $\sigma$ , the narrower the Gaussian function, meaning the curves are steeper. This is to test the different models abilities to handle steep gradients. The data is generated from  $t = 0.0$  to  $t_{\text{end}} = 1.0$  s.

## 2D SWE

For the 2D SWE, we also use the Gaussian function as initial condition for the water height  $h$ , but now in two dimensions:

$$h(x, y, 0) = h_0 + a \cdot \exp\left(-\frac{(x - x_c)^2 + (y - y_c)^2}{2\sigma^2}\right), \quad (5.1.3)$$

where  $h_0$  is the initial water height outside of the Gaussian,  $a$  is the amplitude of the Gaussian in meters,  $(x_c, y_c)$  is the center of the Gaussian in meters, and  $\sigma$  is the standard deviation. The domain is  $x, y \in [0, 40]$  m and is discretized into  $N$  points in each direction. We use the parameters  $h_0 = 1$  m,  $a = 2$  m,  $(x_c, y_c) = (20\text{m}, 20\text{m})$ , and  $\sigma = 2$  m. The initial velocity  $u$  is set to zero, i.e.,  $u(x, y, 0) = 0$ . The data is generated for different values of  $N$  to investigate the effect of the grid resolution on the model performance. To generate data like this also makes it possible to test the models abilities to transfer solutions to different grid resolutions. The values of  $N$  are  $N = 64$ ,  $N = 128$ , and  $N = 256$ . The time step size is variable and is set to satisfy the CFL condition, where the CFL number is set to 0.9. The initial conditions can be seen in Figure 7.17.

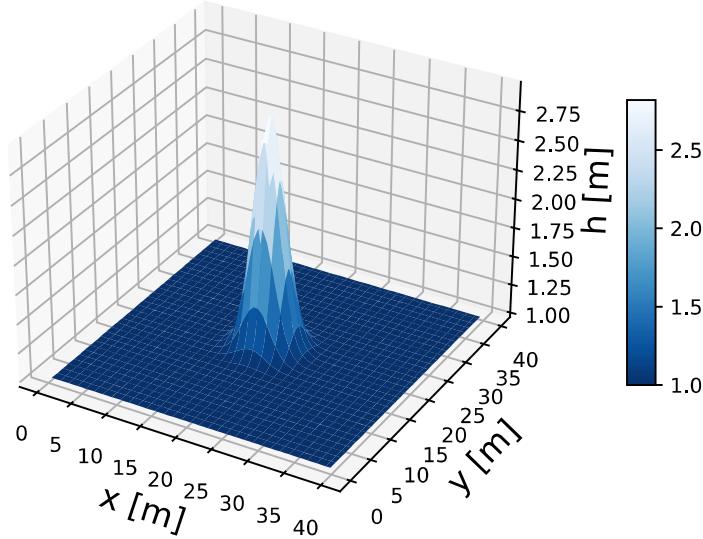
Initial water height,  $h[m]$ 

Figure 5.5: Initial condition for the 2D problem.

To generate the data we use our FVM solver to solve the 2D SWE with the initial conditions given above, which is validated by comparing the results with known test cases. The data is generated from  $t = 0.0$  to  $t_{\text{end}} = 10.0$  s. In most of our test cases the models are using the data from  $t = 0.0$  to  $t_{\text{end}} = 5.0$  s, but we need the data to be generated for a longer time period to test the models ability to make long-term predictions. Since we are making predictions beyond the data, the time step size is unknown, and we must work with a fixed time step size. For generating data for long-term predictions, we employed a constant time step size of  $\Delta t = 0.025$  s. To ensure stability, the time step size must be sufficiently small. To determine  $\Delta t$ , we analyzed the time step sizes used in the variable step data generation. By halving the smallest observed time step size, we obtained  $\Delta t = 0.025$  s.

## 5.2 Data Copernicus

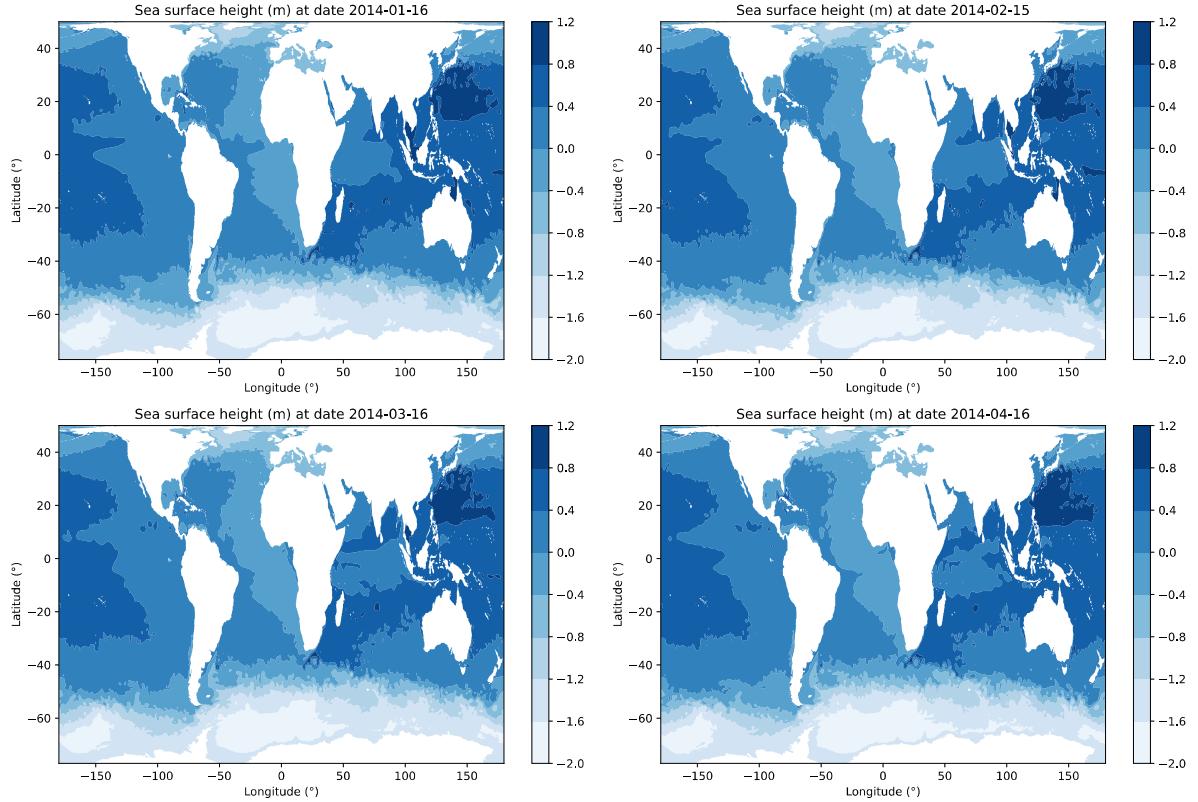


Figure 5.6: Sea surface height as the difference from reference sea surface height for the months Jan-Apr 2014.

## 5.3 Mesh generation for the sphere

To solve the SWE on the sphere, we must use another grid, than the regular grid used in the 2D case. Hence, we use the icosahedral grid, which is a grid that approximates the sphere with triangles. The grid can be generated in different generations, depending on the level of detail we need. For each time we refine to a higher level, the number of triangles increases by a factor of four, i.e., we split each triangle into four smaller triangles. Meaning that the number of triangles is increasing drastically with each level of refinement. The grid is generated by the Github: <https://github.com/siddharth-maddali/SphereMesh/tree/master> and then rewritten to Python. The grid for the first 4 levels of refinement is shown in Figure 5.7. For simplicity, we will begin by considering the first level of refinement, which consists of 20 triangles. The matrix  $\text{tri}$  is a  $20 \times 3$  matrix, where each row represents a triangle and the three columns represent the three vertices of the triangle. The vertices are stored in the matrix  $P$ , which is a  $12 \times 3$  matrix, where each row represents a vertex and the three columns represent the x, y, and z coordinates of the vertex. The vertices are normalized to the unit sphere, i.e., the radius of the sphere is 1. The idea is now, that similar to the case in cartesian coordinates, we loop through each cell, in this case triangles, and calculate the fluxes between the cells. We must be aware of which cells/triangles are neighbors, and we must also be aware of the orientation of the triangles, i.e., the normal vector of the triangle. The normal vector is calculated by the cross product of the two vectors that span the triangle. The normal vector is then normalized to the unit sphere. The matrix  $\text{tri}$  is my Element to Vertex matrix. I also need an Element-to-Face table and a Element-to-Element table. The Element-to-Face table is used to define which edges or faces belong to each triangle. Each face of a

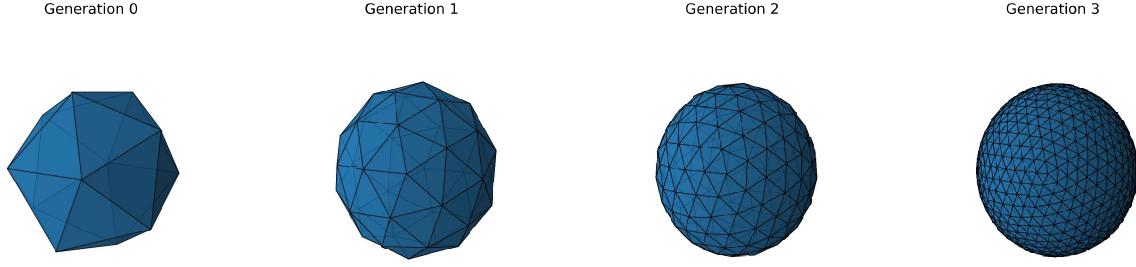


Figure 5.7: Icosahedral grid for the first 4 levels of refinement.

triangle is an edge shared between two triangles. The Element-to-Element table is used to define which triangles are neighbors. That is, it indicates which triangles share an edge. This is important when calculating the fluxes between the triangles. To construct this table we loop through each triangle and check if the edge of the triangle is shared with another triangle.

We make the FVM to solve SWE for a triangular grid on the sphere. For each triangle, we consider each face (edge). For each face we define the normal vector in terms of the spherical unit vectors  $\mathbf{e}_\theta$  and  $\mathbf{e}_\phi$ . The spherical unit vectors at a point are:

$$\mathbf{e}_\theta = \begin{bmatrix} -\sin(\theta) \\ \cos(\theta) \\ 0 \end{bmatrix},$$
$$\mathbf{e}_\phi = \begin{bmatrix} \cos(\phi) \cos(\theta) \\ \cos(\phi) \sin(\theta) \\ -\sin(\phi) \end{bmatrix}.$$

Where  $\mathbf{e}_r$  is the radial direction, going outward from the origin.  $\mathbf{e}_\theta$  is the longitude direction, and  $\mathbf{e}_\phi$  is the latitude direction. The unit vectors are tangential to the sphere. We do not need to consider the radial direction, as the SWE is only in the tangential directions.

Now we have the cartesian edge vectors in tangential directions,  $\mathbf{e}_0, \mathbf{e}_1, \mathbf{e}_2$ .

The projection of an edge vector  $\mathbf{e}_i = (e_{ix}, e_{iy}, e_{iz})$  onto  $\mathbf{e}_\theta$  is:

$$\mathbf{e}_i \cdot \mathbf{e}_\theta = e_{ix} (-\sin \theta \cos \phi) + e_{iy} (-\sin \theta \sin \phi) + e_{iz} \cos \theta$$

Similarly, the projection onto  $\mathbf{e}_\phi$  is:

$$\mathbf{e}_i \cdot \mathbf{e}_\phi = e_{ix} (-\sin \phi) + e_{iy} \cos \phi$$

Recall that the cross product between two vectors in 3D produces a third vector that is orthogonal to the two input vectors, i.e., the plane spanned by the two input vectors.

For a given face  $f$  of a triangle, we first calculate the Cartesian face normal  $\mathbf{n}_f$  as the cross product of the two

vectors that span the face. For a given face  $f$  of a triangle, we define the normal vector  $\mathbf{n}_f$  is decomposed into the spherical unit vectors as

# Chapter 6

## Numerical results

In this chapter we present the results of the numerical experiments, where we have implemented the FVM for solving the SWE and tested it on several problems. A key focus is to validate the implementation, as it will generate data for the data-driven methods, including the convolutional neural networks and Fourier neural operators. In section 6.1, we solve the 1D dam break problem and compare the numerical solution against the true solution. In section 6.2, we present the results from the five test cases from Toro (2001) [5]. These problems are all discontinuous in either the water height  $h$  or the fluid velocity  $u$ . The idea is, that if the numerical solution can capture the discontinuities, it should be well-suited to handle smoother solutions as well. We also compare the results from the different fluxes used in the FVM.

We extent to consider 2D problems, and present the results from the 2D idealised circular dam break problem in section 6.4. The results from the 2D problem are compared to the results from the book to validate the implementation of the 2D FVM. In section 6.5, we test the scalability of the FVM to solve the 2D SWE, by running the 2D problem with a Gaussian initial condition for different values of  $N$ , i.e., the number of cells in each direction. Finally, in section 6.6, we present animations for the 2D idealised circular dam break problem and the solution of the SWE with a Gaussian initial condition projected on a sphere.

The used computer for running simulations is a Windows 11 computer with an Intel Core i7 CPU and 16 GB of RAM. All code and data can be found at the Github repository [32].

### 6.1 The 1D Dam Break Problem

We consider the 1D dam break problem, a special case of the Riemann problem (3.4.1), with the following initial conditions:

$$h(x, 0) = \begin{cases} h_L, & \text{if } x < x_0, \\ h_R, & \text{if } x > x_0, \end{cases}$$

where  $x \in [0 \text{ m}, 50 \text{ m}]$ ,  $h_L = 3.5 \text{ m}$ ,  $h_R = 1.25 \text{ m}$  and  $x_0 = 20 \text{ m}$ . Since it is a dam break problem the initial fluid velocity is zero, i.e.,  $u(x, 0) = 0 \text{ m/s}$ . We solve the problem starting at  $t = 0 \text{ s}$  and ending at  $t = 2.5 \text{ s}$ . The numerical solution to the 1D dam break problem using the FVM, together with the true solution, provided from the Ph.D. course *An Introduction to Discontinuous Galerkin Methods for solving Partial Differential Equations* [33], can be seen in Figure 6.1.

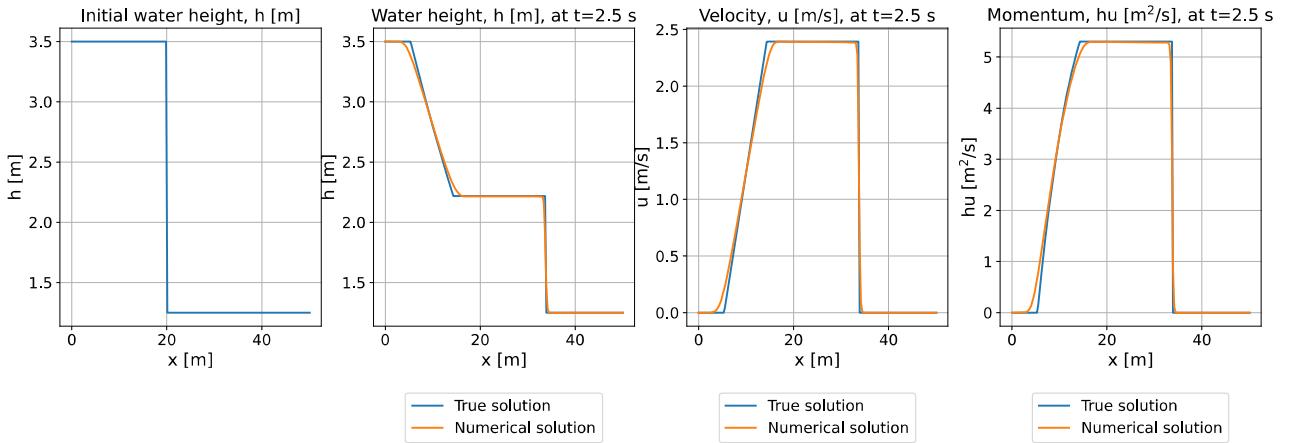


Figure 6.1: The initial water height  $h$  at  $t = 0$  s, together with the water height, the fluid velocity  $u$  and the momentum  $hu$  at  $t = 2.5$  s.

From Figure 6.1 we see that the numerical solution aligns well with the true solution, and overall successfully captures the discontinuity. We also see that the solution consists of a right shock wave and a left rarefaction wave, as expected from the initial conditions. This is seen as, the shock wave to the right has a high speed and represent a discontinuity in the solution for the water height. The rarefaction wave to the left is a smooth transition from the high water height to a lower water height.

## 6.2 Toro test cases

We have tested the FVM on the five test cases for Riemann problems from Toro (2001) [5]. The initial conditions for the five test cases are given in Table 6.1.

Test case	$h_L$ [m]	$u_L$ [m/s]	$h_R$ [m]	$u_R$ [m/s]	$x_0$ [m]	$t_{end}$ [s]
1	1.0	2.5	0.1	0.0	10.0	7.0
2	1.0	-5.0	1.0	5.0	25.0	2.5
3	1.0	0.0	0.0	0.0	20.0	4.0
4	0.0	0.0	1.0	0.0	30.0	4.0
5	0.1	-3.0	0.1	3.0	25.0	5.0

Table 6.1: Initial conditions for the five test cases.

The domain is  $x \in [0 \text{ m}, 50 \text{ m}]$  for all test cases. The Riemann problems are chosen to test the numerical method on different combinations of shock waves and rarefaction waves. To solve the test cases we have used the following fluxes:

1. Godunov method with exact Riemann solver,
2. Lax-Friedrich flux,
3. Lax-Wendroff flux,
4. FORCE flux,

5. HLL flux,

### Test case 1

The initial conditions for test case 1 are visualised in Figure 6.2 and the solution after  $t = 7.0$  s are given in Figure 6.3.

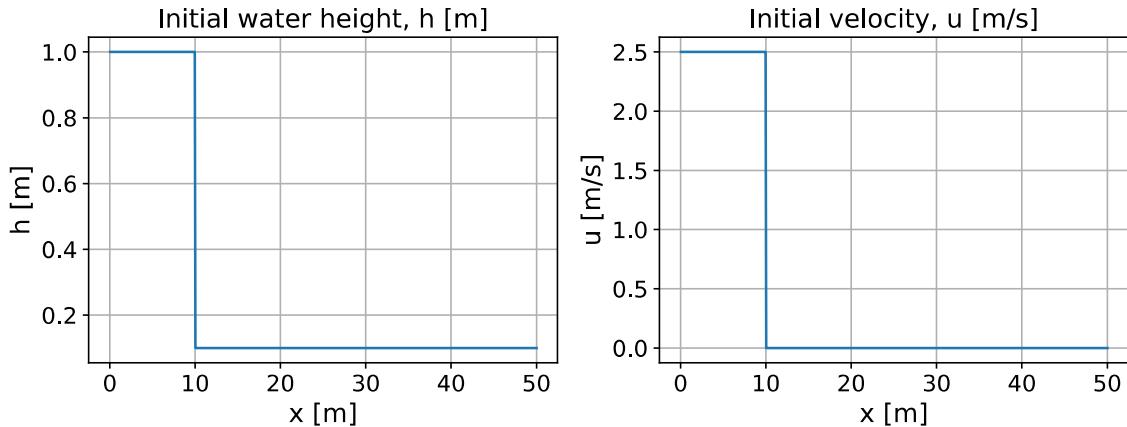


Figure 6.2: Initial conditions for the test case.

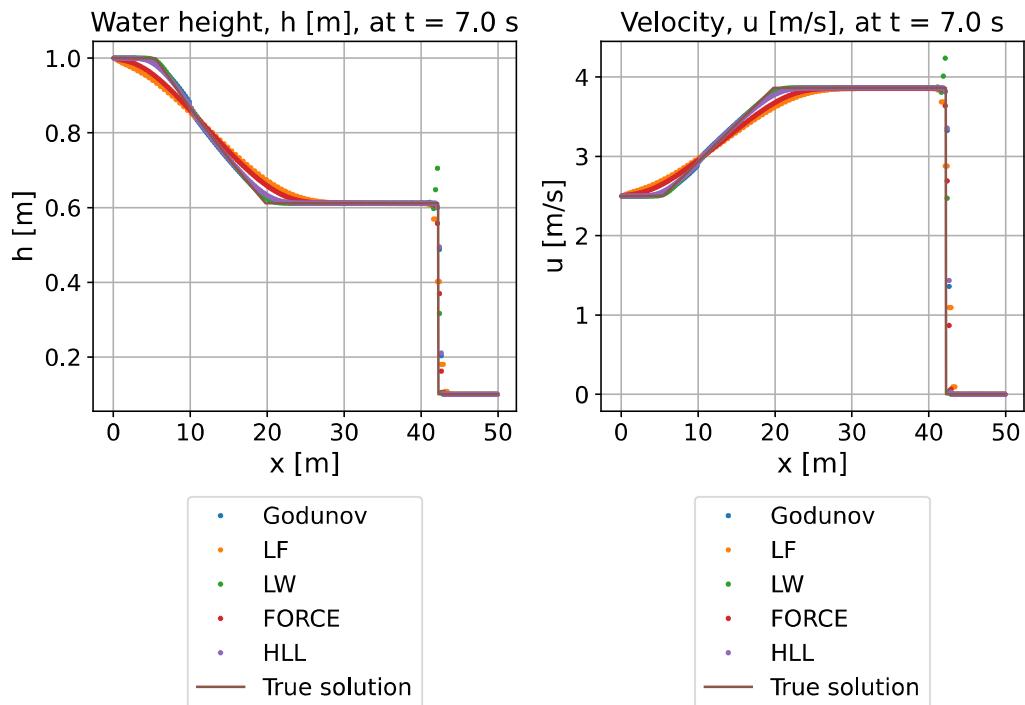


Figure 6.3: Final solution for the test case after  $t = 7.0$  s.

For this test case all the fluxes work well, but there are minor differences in the solution, which can be seen in Figure 6.3. For instance, we see that the Lax-Wendroff (LW) flux has some oscillations in the solution at the discontinuity, which is not present in the other fluxes. As in the case of the 1D dam break problem in section 6.1, the solution consists of a right shock wave and a left rarefaction wave.

## Test case 2

The initial conditions for test case 2 are illustrated in Figure 6.4.

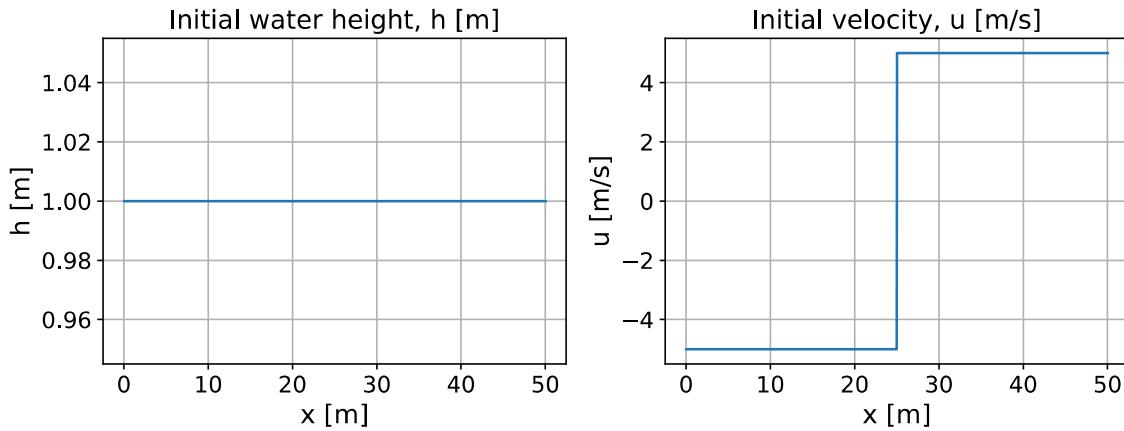


Figure 6.4: Initial conditions for the test case.

In test case 2 we have two rarefaction waves, one on the left side and one on the right side. As they are travelling in opposite directions (away from each other), due to the initial velocity, there will be created a nearly dry bed in the middle of the domain. Many methods have difficulties with this test case as they may compute a negative water height. For these experiments we were able to get close to the true solution, using Lax-Friedrich flux, FORCE flux and HLL flux. The solution after  $t = 2.5$  s are illustrated in Figure 6.5.

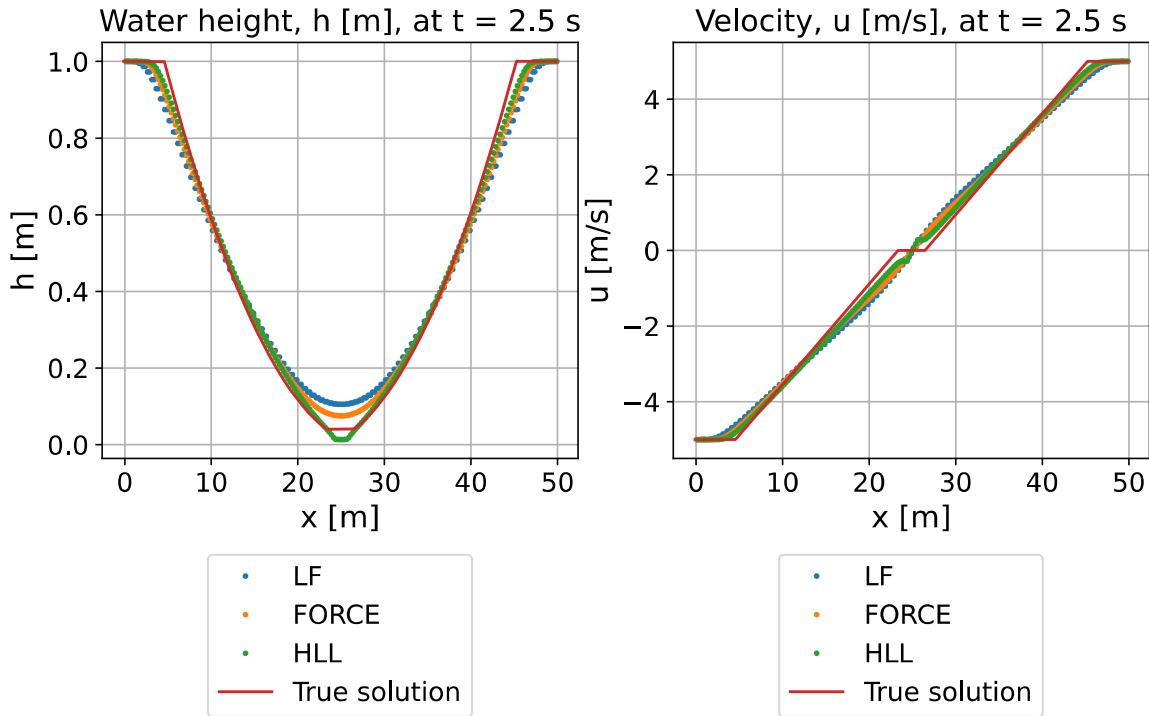


Figure 6.5: Final solution for the test case.

For the fluxes, Godunov method with exact Riemann solver, and Lax-Wendroff, it was not possible to get an acceptable solution.

### Test case 3

The initial conditions for test case 3 are given in Figure 6.6.

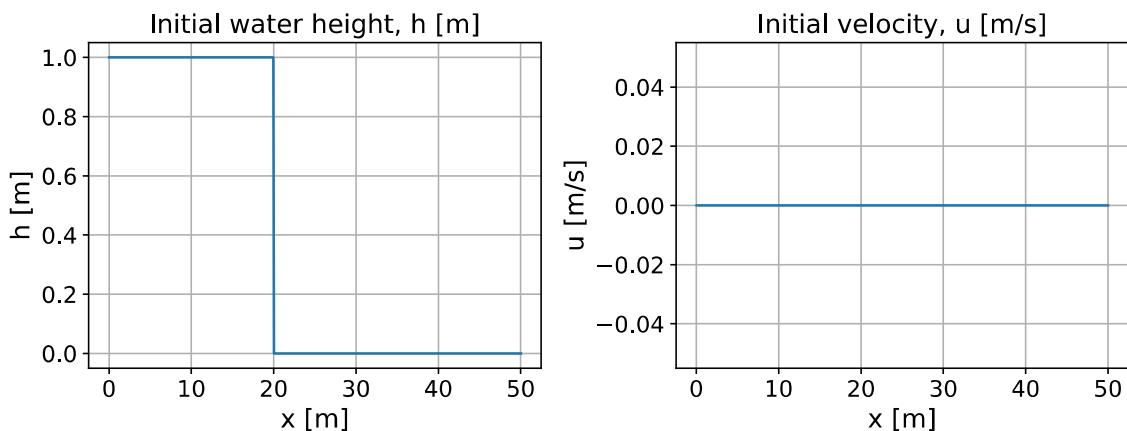


Figure 6.6: Initial conditions for the test case.

In Figure 6.6 we see that this is a dam break problem, as the initial velocity is zero everywhere. The solution after  $t = 4.0$  seconds are given in Figure 6.7.

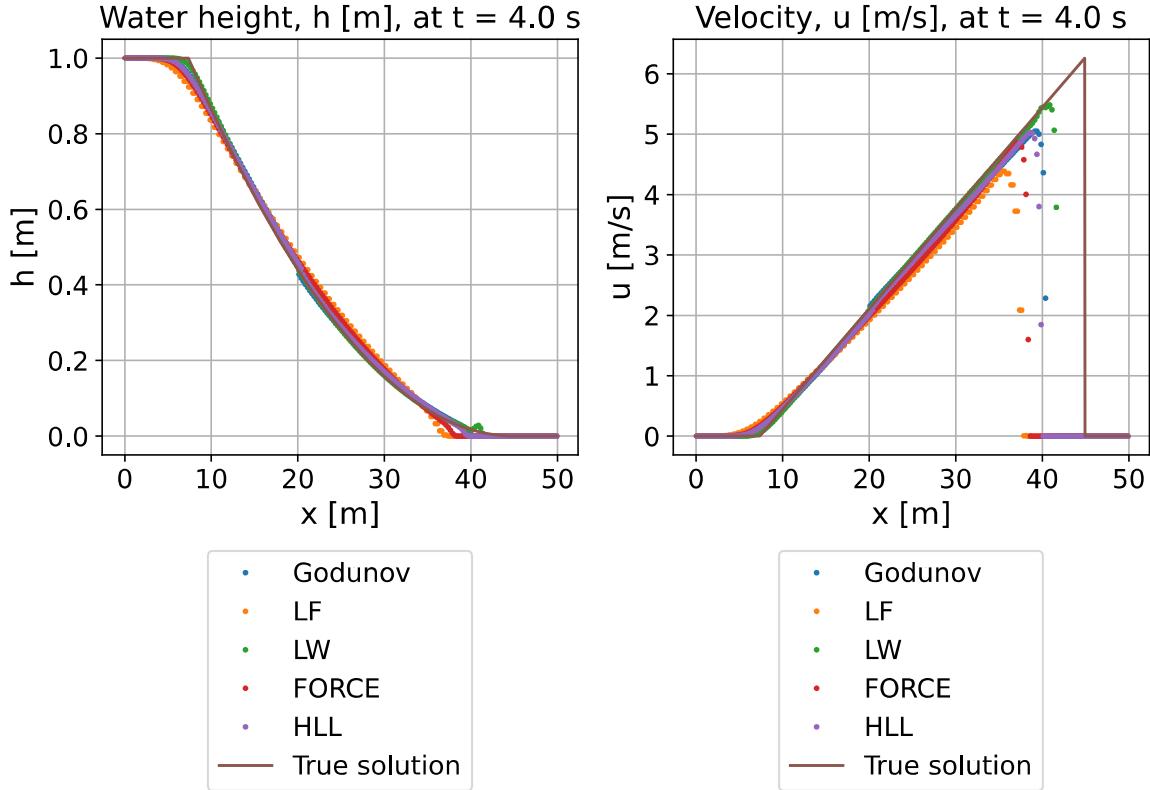


Figure 6.7: Final solution for the test case.

In this case, we are challenged with a dry-bed region in the right part of the domain. To solve case 3, with the FVM we must add a small amount to  $h_R$ , since the code does not handle  $h_R = 0$  well. We set  $h_R = 0.00005$  to solve it numerically, but the true solution is for  $h_R = 0$ . By running experiments with different values of  $h_R$ , we see that the solution converges to the true solution as  $h_R$  approaches 0. From Figure 6.7 we see that when it comes to predicting the velocity, there are some differences in how the different fluxes perform. Overall, using the Lax-Wendroff (LW) flux gives the closest solution to the true solution for the velocity.

#### Test case 4

The initial conditions for test case 4 are given in Figure 6.8, and the solution after  $t = 4.0$  s are given in Figure 6.9.

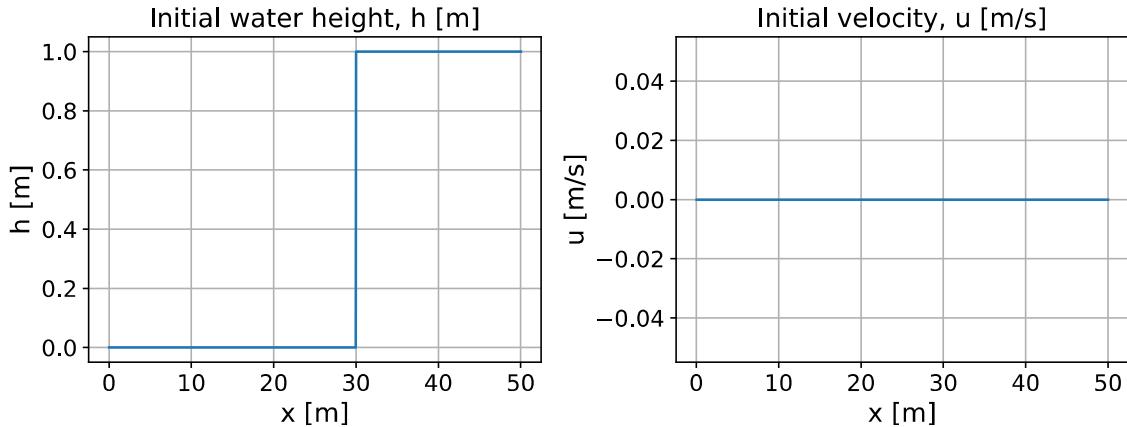


Figure 6.8: Initial conditions for the test case.

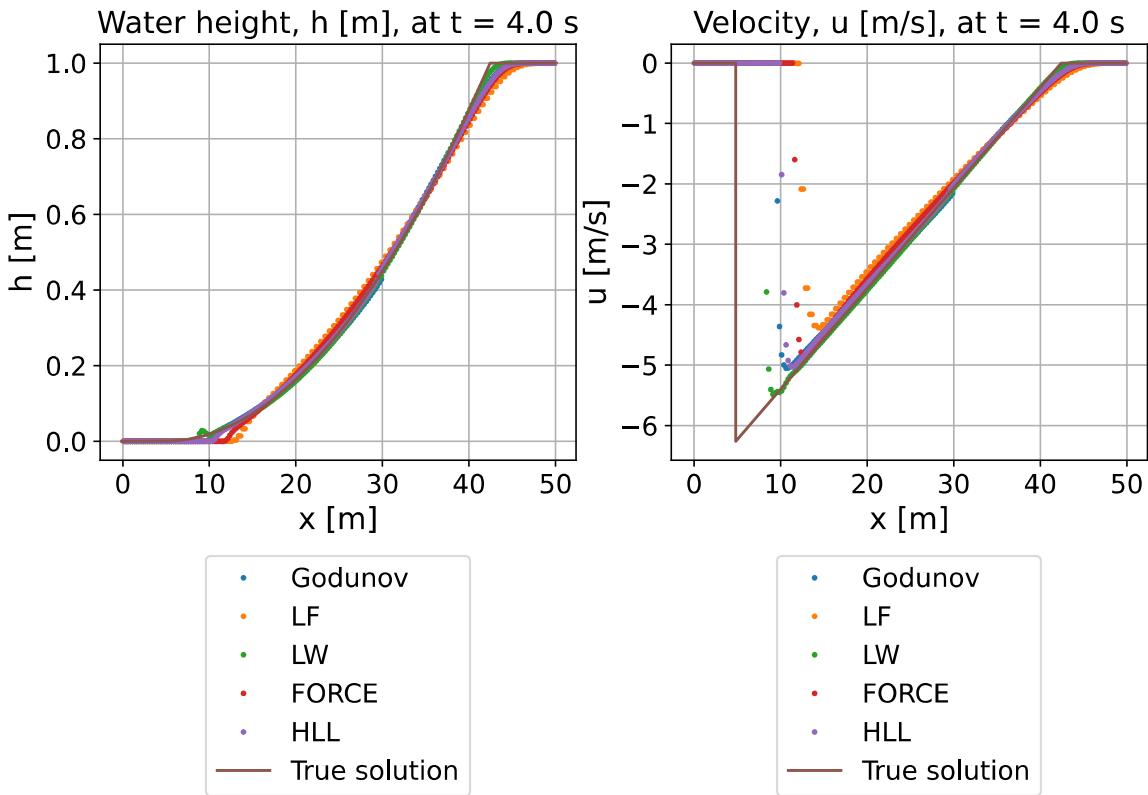


Figure 6.9: Final solution for the test case.

This test problem is symmetric to test case 3, meaning we face the same challenges of a dry-bed region, now located in the left part of the domain. We set  $h_L = 0.00005$ , and the solution converges to the true solution as  $h_L$  approaches 0. This case is included to test if the results are as expected. As in test case 4, we observe differences in the fluxes performance, and again we observe that the Lax-Wendroff (LW) flux comes closest to the true solution

for the velocity.

### Test case 5

The initial conditions for test case 5 are given in Figure 6.10, and the final solution after  $t = 5.0$  s are given in Figure 6.11.

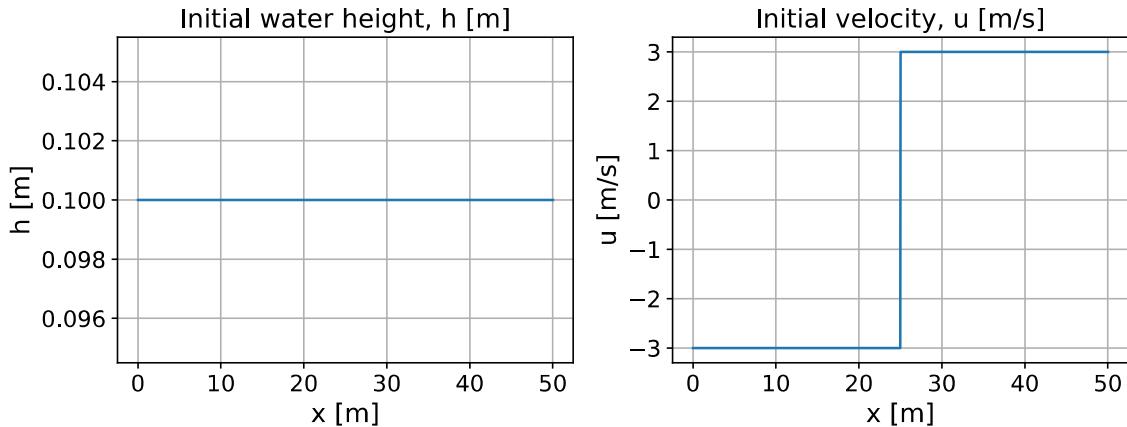


Figure 6.10: Initial conditions for the test case.

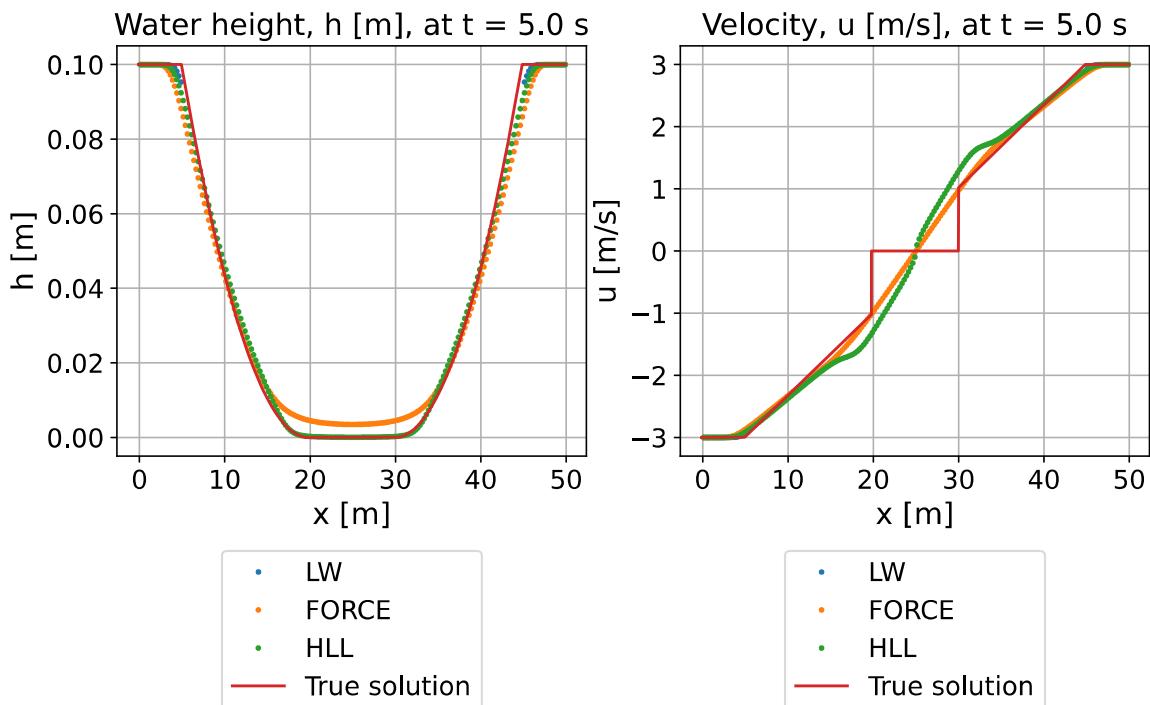


Figure 6.11: Final solution for the test case.

From Figure 6.11 we see that the numerical solutions for the velocity  $v$  at  $t = 5.0$  are smooth, where the true solution is discontinuous. In this test case there are also challenges with some of the fluxes due to the generation of a dry-bed region in the middle of the domain. The fluxes that are not able to solve this case are Godunov method with exact Riemann solver and Lax-Wendroff flux, the same as in test case 2. The solution consists of two rarefaction waves, one on the left side and one on the right side, and a dry-bed region in the middle.

To get an overview of which fluxes that were able to produce solutions for the test cases, consider Table 6.2. The table shows which fluxes that were able to produce a solution for the test cases.

Test case	Godunov	LF	LW	FORCE	HLL
1	✓	✓	✓	✓	✓
2	✗	✓	✗	✓	✓
3	✓	✓	✓	✓	✓
4	✓	✓	✓	✓	✓
5	✗	✓	✗	✓	✓

Table 6.2: Overview of which fluxes that were able to produce solutions for the test cases.

Note, that as we see in the results, there are still differences in the solutions accuracy between the fluxes that were able to solve the test cases.

### 6.3 1D linearized Shallow Water Equations on a sphere

In this section we consider the 1D LSWE on a sphere. We consider the spatial dimension  $\theta$ , which is the longitude angle and keep the latitude  $\phi$  constant. The LSWE on a sphere are given by (2.6.4) with the initial conditions given by (5.1.2). The LSWE on a sphere are solved using the FVM with the ERK4 time-stepping method. The initial conditions, and the solution after  $t = 0.25$  s and  $t = 0.5$  s are visualised in Figure 6.12.

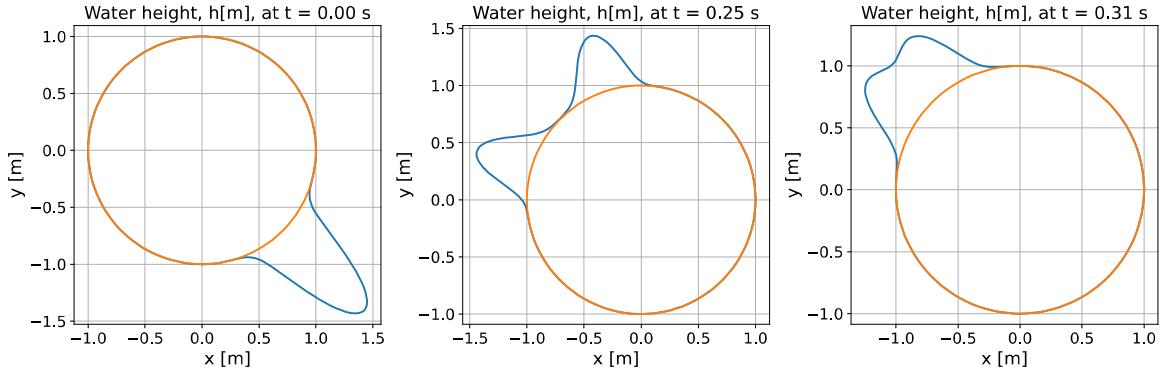


Figure 6.12: The water height,  $h[m]$ , at  $t = 0$ s,  $t = 0.25$  and  $t = 0.31$ .

From Figure 6.12 we see how the water height evolves over time. We observe that the initial wave split into two waves, travelling in opposite directions. When the two waves meet, we see how the periodic boundary conditions operate, as the waves are melting together, and then split again into two waves, as in the beginning. This is more clear visualised in the animation for this case, a link and qr-code can be found in Figure 6.15. Since we do not account for friction and have neglecting external forces except gravity, the waves will continue to travel around the sphere. In this case we do not have a true solution to compare with, but the numerical solution seems reasonable.

## 6.4 2D idealised Circular Dam Break Problem

We now proceed to the 2D case, considering an idealised circular dam break problem over a horizontal bottom. This problem is also from Toro (2024) [7]. We assume there is an infinitely thin circular wall at radius  $R = 2.5$  m in a square domain of size  $40 \text{ m} \times 40 \text{ m}$  with centre at  $(x_c, y_c) = (20 \text{ m}, 20 \text{ m})$ . The initial conditions are

$$h(x, y, 0) = \begin{cases} 2.5 \text{ m}, & \text{if } \sqrt{(x - x_c)^2 + (y - y_c)^2} \leq R, \\ 0.5 \text{ m}, & \text{otherwise,} \end{cases}$$

$$u(x, y, 0) = 0 \text{ m/s},$$

$$v(x, y, 0) = 0 \text{ m/s}.$$

We use a mesh of size  $128 \times 128$ . The problem is solved using the FVM with the Rusanov flux, and the results after  $t = 0.0 \text{ s}, 0.4 \text{ s}, 0.7 \text{ s}$  and  $1.4 \text{ s}$  are visualised in Figure 6.13.

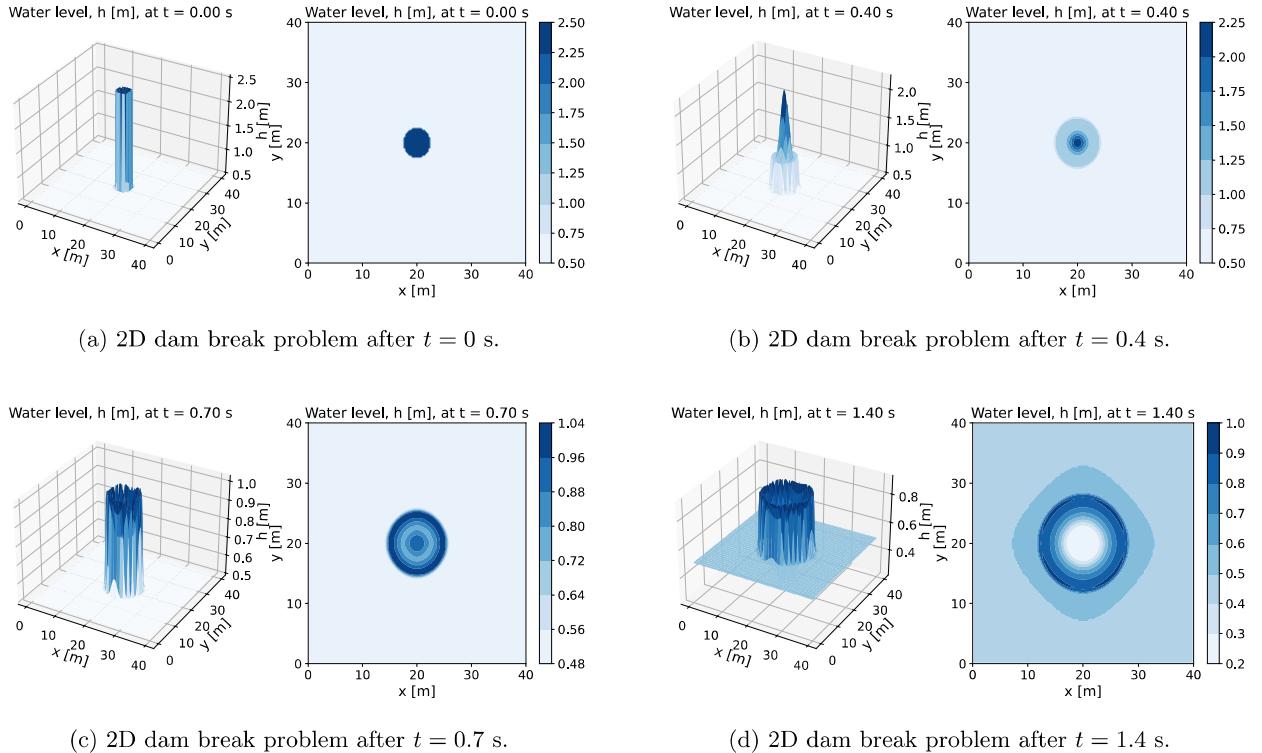


Figure 6.13: Snapshots of the 2D dam break problem at different times.

By comparing Figure 6.13 with the results from Toro (2024) [7], we see that the numerical solution aligns well with the true solution.

## 6.5 Scalability

Numerical methods are good as they can be more or less as accurate as we want them to be, but the computational cost increases with the number of cells. Previously in the thesis, we have indicated that a disadvantage of the FVM

is that the computational cost increases with the number of cells. To test the scalability of the FVM to solve the 2D SWE, we have solved the 2D SWE with a Gaussian initial condition, as described in chapter 5, for different values of  $N$ , i.e., the number of cells in each direction, and measured the run time. The results are presented in Table 6.3.

$N$	16	32	64	128	256
Time (s)	0.37	2.73	19.90	170.08	4205.45

Table 6.3: Running time for the FVM to solve 2D SWE for different values of  $N$ .

The run time [s] dependent on the number of cells  $N$  is illustrated in Table 6.3.

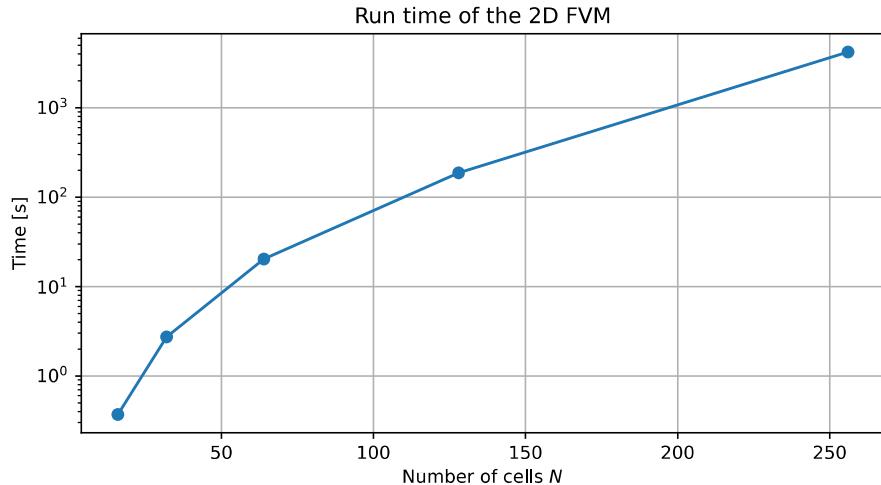


Figure 6.14: Run time [s] of the 2D FVM to solve the SWE, depending on number of grid cells  $N$ . Note that the axis with time is in log-scale.

From Table 6.3 and Figure 6.14 we see that the run time increases drastically with the number of cells  $N$ . This is expected, as when the number of cells increase, the number of computations increase as well. This also means that the computational cost increases with the number of cells, and the method is not scalable for large values of  $N$ . Ultimately, if we want to model floods or tsunamis for the real world, we need a scalable method. This is also some of the motivation for using data-driven methods, as we will investigate if they can be more scalable.

## 6.6 Animations

To visualise the results from the numerical experiments, we have created animations for the 2D idealised circular dam break problem and the SWE with a Gaussian initial condition on a sphere. The following QR-codes can be used to access the animations. The QR-code in Figure 6.15a leads to the animation for the 2D idealised circular dam break problem for  $t = 0$  s to  $t = 10$  s, and the QR-code in Figure 6.15b leads to the animation for the SWE with initial Gaussian conditions, projected on a sphere, for  $t = 0$  s to  $t = 0.5$  s. There is also a link to the animations in the caption of the QR-codes, pointing to a dedicated Github repository for the animations.



(a) QR-code for the 2D idealised circular dam break problem.



(b) QR-code for the SWE with initial Gaussian conditions, projected on a sphere.

Figure 6.15: QR-codes for the animations. All animations can be found at: <https://github.com/MelissaJessen/Shallow-Water-Equations-Animations/blob/main/README.md>.

In the animation for the 2D idealised circular dam break problem, we observe what happens after the time steps in Figure 6.13. We note that when the waves hit the boundaries, they are reflected back into the domain, demonstrating the behaviour of the boundary conditions. In the animation for the SWE with a Gaussian initial condition on a sphere, note that since the SWE is solved in a 2D cartesian domain and wrapped to a sphere, we observe non-physical behaviour, especially close to the poles. This motivates the use of an alternative method for solving the SWE on a sphere.

# Chapter 7

## Data-driven results

In this chapter we present the results of the data-driven models, that is, the convolutional neural network (CNN) and the Fourier neural operator (FNO), for solving the shallow water equations. We analyse three main cases:

1. 1D SWE with Gaussian initial conditions.
2. 1D linearized SWE in spherical coordinates.
3. 2D SWE with Gaussian initial conditions.

In each case, we evaluate the performance of the CNN and the FNO in terms of mean squared error (MSE), mean absolute error (MAE), training time and prediction time. In addition, we analyze the models' predictions for various initial conditions and their ability to generalize to unseen data. We also explore their capability to transfer solution across different grid resolutions, such as transitioning from a coarse grid to a fine grid. Finally, we examine the models' long-term predictive performance, assessing how well they maintain accuracy over extended time steps. The convolutional neural networks are implemented in python version 3.12.6, using PyTorch version 2.4.1. The Fourier neural operators are implemented using the Github repository neuraloperator which is a library for learning neural operators in PyTorch [34].

In chapter 6, we presented the results of discontinuous test cases, operating under the premise that a solver capable of handling discontinuous solutions should also perform well with smooth solutions. In this chapter, we shift our focus to evaluating the performance of data-driven models under smooth initial conditions, as these are more representative of real-world scenarios.

### 7.1 1D Shallow Water Equations with Gaussian initial conditions

In this section, we consider the 1D shallow water equations with Gaussian initial conditions, where the data is generated as described in section 5.1. For both the CNN and FNO models, we train the models on the data generated by the numerical solution of the shallow water equations. The initial condition for the water height  $h$  is given in (5.1.1) and illustrated in Figure 7.1.

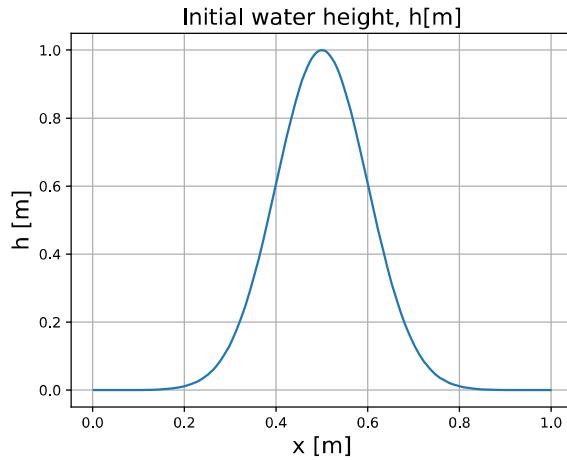


Figure 7.1: The initial conditions for water level  $h$  in the 1D SWE is a Gaussian function.

The domain is  $x \in [0, 1]$  with  $N = 200$  points and the data covers the time interval  $t \in [0, 1]$  s. In Figure 7.1, we see that the initial condition is a Gaussian function with center in the middle of the domain. To get an overview of how the solutions evolve over time, we have plotted the numerical solution in the  $x, t$ -plane, shown in Figure 7.2, in both a contour plot and a 3D plot.

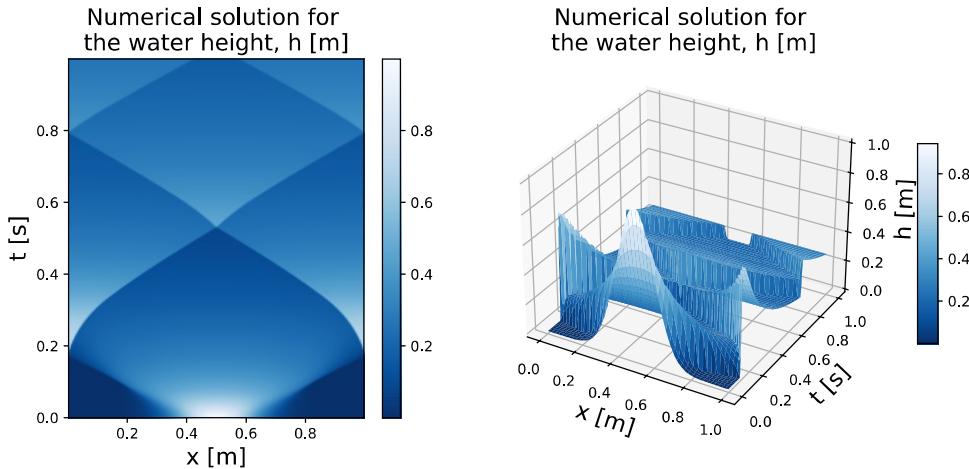


Figure 7.2: Numerical solution of the 1D SWE from  $t = 0$  s to  $t = 1$  s.

In Figure 7.2, we see how the water height in the domain evolves over time. The 3D plot provides an overview of the solution, whereas the contour plot offers a more detailed view of the water level. From the contour plot, we observe that even though the initial condition is smooth, the solution develops sharp edges, close to discontinuities, over time, represented by the lines in the plot. This behavior is typical for the nonlinear shallow water equations, where the solution can become discontinuous due to the formation of shock waves. Some details of the data can be seen in the following table.

Case	n_train	n_val	n_test	N	$\Delta x$	$\Delta t$
1D SWE	369	123	123	200	0.005 m	[0.0008 s, 0.00225 s]

Table 7.1: Details of the used data for the case with the 1D SWE with a Gaussian initial condition.

In Table 7.1, there are listed some details of the CNN model. The time step size  $\Delta t$  is not constant, but varies between 0.0008 s and 0.00225 s. This is due to the CFL condition, which requires the time step to be small enough to ensure stability.

## CNN Model

In the convolutional neural network, we train the model using the data generated by the numerical solution of the shallow water equations. The model uses the data from the numerical solution to predict the solution at the next time step. Meaning that the input and output data are the same, but shifted one time step. This way, the model is supposed to learn the flowmap. The model takes input with 10 channels, equivalent to the sequence length, and applies a series of 1D convolutional layers. The model uses three convolutional layers with ReLU activation functions. The final convolutional layer reduce the output to a single channel, and a fully connected layer maps the processed features to the output. The model has been trained using the Adam optimizer with a learning rate of 0.001 and a batch size of 32. The criteria is to minimize the mean squared error (MSE). The model is trained on the first 60% of the data, validated on the next 20%, and tested on the last 20%. The exact number of training points can be seen in Table 7.1. The model is trained for 500 epochs, but early stopping is used to prevent overfitting. Meaning that the model is saved if the validation loss is lower than the previous best validation loss. The training and validation loss for the CNN model is shown in Figure 7.3.

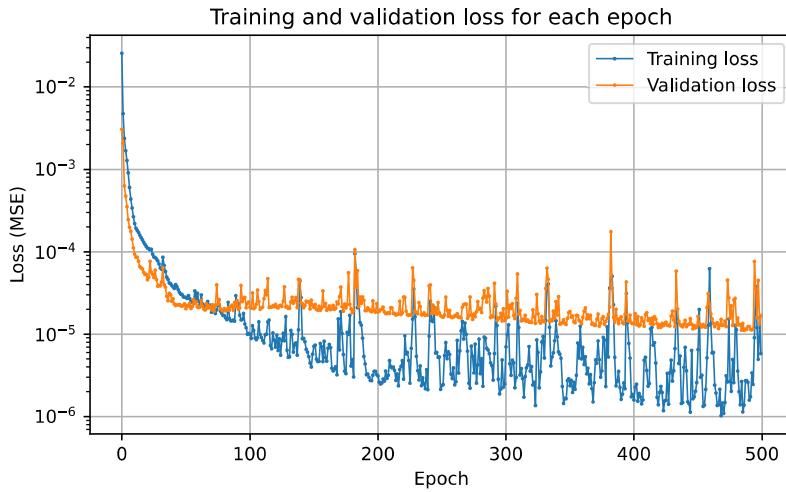


Figure 7.3: Training and validation loss (MSE) for the CNN model.

In Figure 7.3, we see that the training and validation loss decrease over the epochs, demonstrating that the model is learning the dynamics of the solution. However, while the training loss continues to decrease, the validation loss has largely stabilized. This indicates that further training is unlikely to improve the model's performance and may lead to overfitting. Additionally, we assess the accuracy of the model's predictions by examining the error, as shown in Figure 7.4.

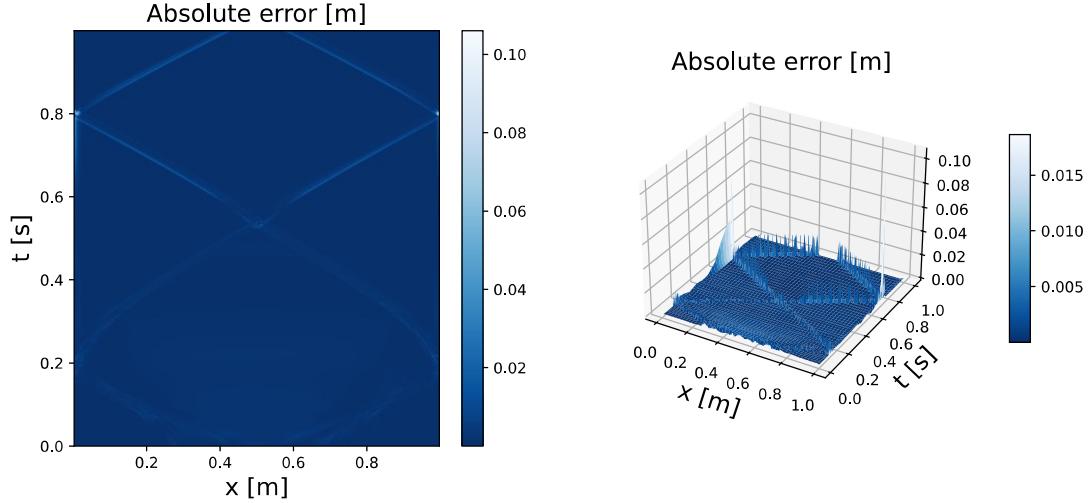


Figure 7.4: Error plot for the predictions for the CNN model.

In Figure 7.4, the largest errors are observed in regions where the solution exhibits discontinuities, which is expected as the model struggles to accurately predict in these areas. We also see that, the error tends to increase over time, reflecting the model's limitations when extrapolating beyond the training data. To gain deeper insight into the model's performance, we examine its predictions at specific time steps, as shown in Figure 7.5.

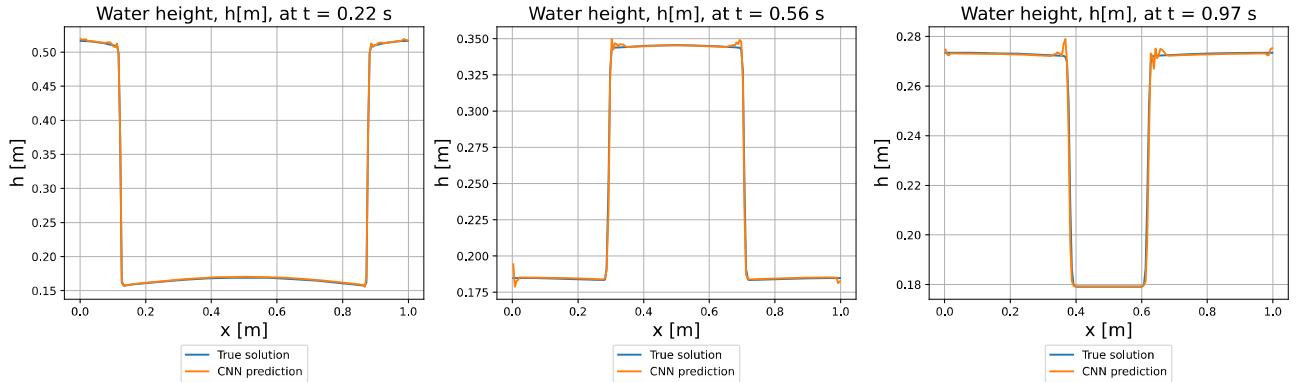


Figure 7.5: Predictions for the CNN model for some given time steps.

From Figure 7.5, we observe that the CNN model overall captures the dynamics of the solution, but struggles to predict the sharp edges. This is especially illustrated in the prediction at  $t = 0.9$  s, where we observe oscillations in the solution that are not present in the true solution.

## FNO Model

We define a FNO model, which consists of an input channel, 64 hidden channels and an output channel. We use a Fourier basis with 16 modes and a batch size of 32. The model is trained using the Adam optimizer with a learning rate of 0.001 and the criteria is to minimize the mean squared error (MSE). We use the same train/validation/test

split as for the CNN model. The model is trained for 500 epochs, but the current best model is saved if the validation loss is lower than the previous best validation loss. The training and validation loss for the FNO model is shown in Figure 7.6.

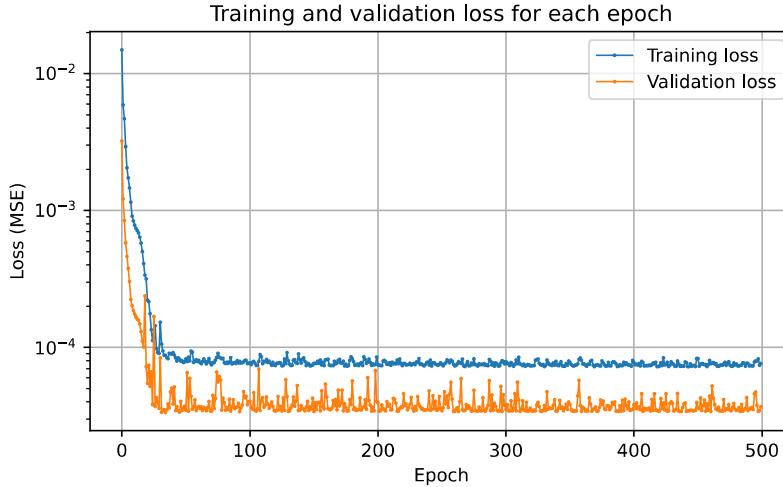


Figure 7.6: Training and validation loss for the FNO model.

From Figure 7.6, we see that the training and validation loss decrease over the epochs, indicating that the model is learning the dynamics of the solution. As with the CNN model, the validation loss has largely stabilized, suggesting that further training is unlikely to improve the model's performance. To see how the errors are distributed in the solution, we plot the error in Figure 7.7.

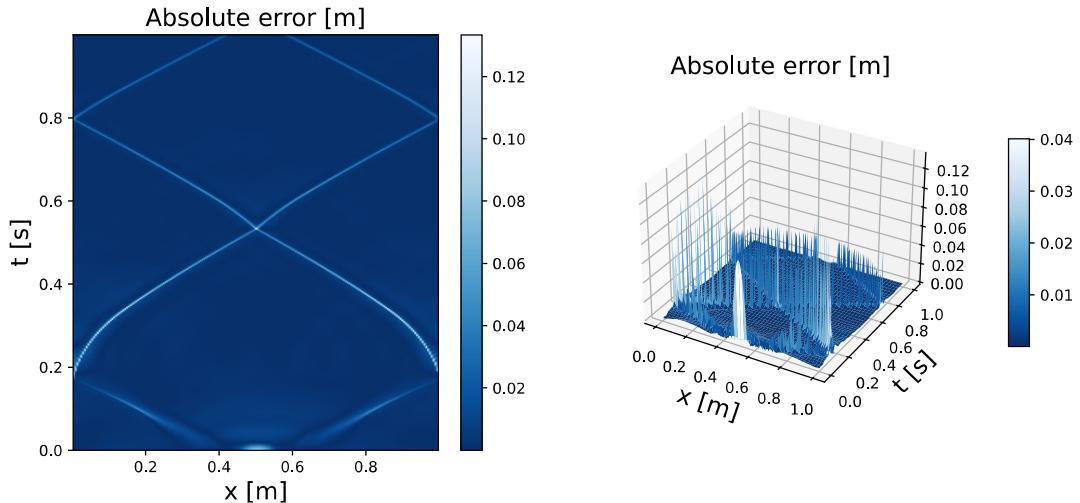


Figure 7.7: Error plot for the predictions for the FNO model.

In Figure 7.7, we see more or less the same error distribution as for the CNN model, with the largest errors at the discontinuities, and the error increasing over time. To get an overview of the performance of the model, we consider the predictions for some given time steps, shown in Figure 7.8.

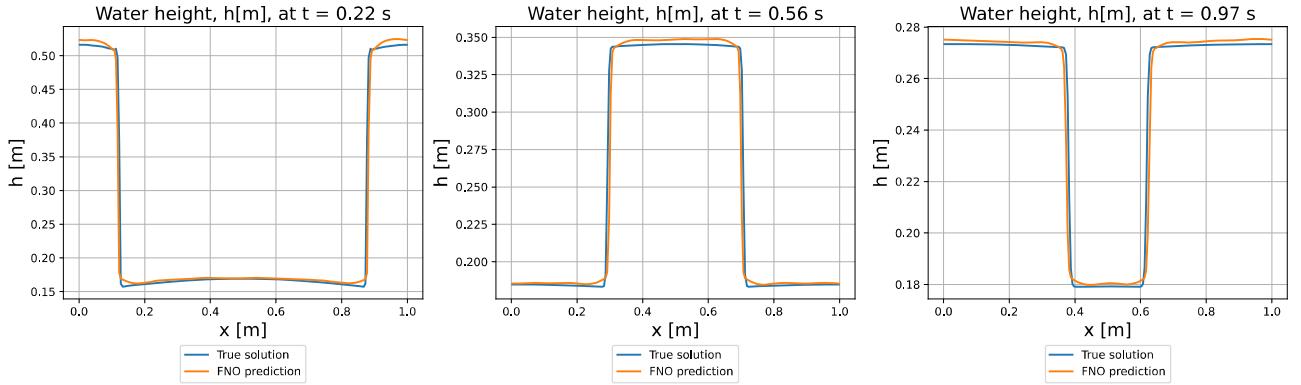


Figure 7.8: Predictions for the FNO model for some given time steps.

From Figure 7.8, we see that the FNO model overall provides a smooth and accurate prediction.

## New initial condition

To evaluate the models' ability to generalize to unseen initial conditions, we introduce a new initial condition for the water height  $h$ . This new condition retains the Gaussian form described in (5.1.1), but with a different mean parameter  $\mu$ . Specifically,  $\mu$  is set to  $\mu = 0.3$  m, shifting the initial condition to the left. The new initial condition is illustrated in Figure 7.9.

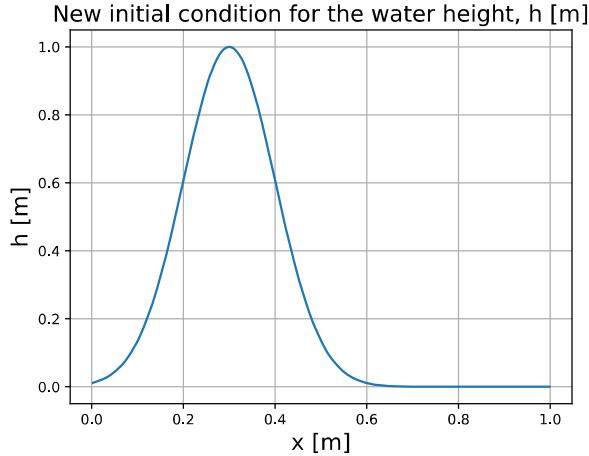


Figure 7.9: New initial condition for the 1D SWE with a Gaussian function.

The performance of the CNN and FNO models for this new initial condition is summarized in Table 7.2. The predictions are for one time step.

## Comparison

To compare the performance of the CNN and FNO models, we consider the MSE and MAE for the predictions of the 1D SWE with Gaussian initial conditions. The reason we consider both MSE and MAE is that the MSE is more sensitive to outliers, while the MAE provides a more general overview of the error. The results are summarized in Table 7.2.

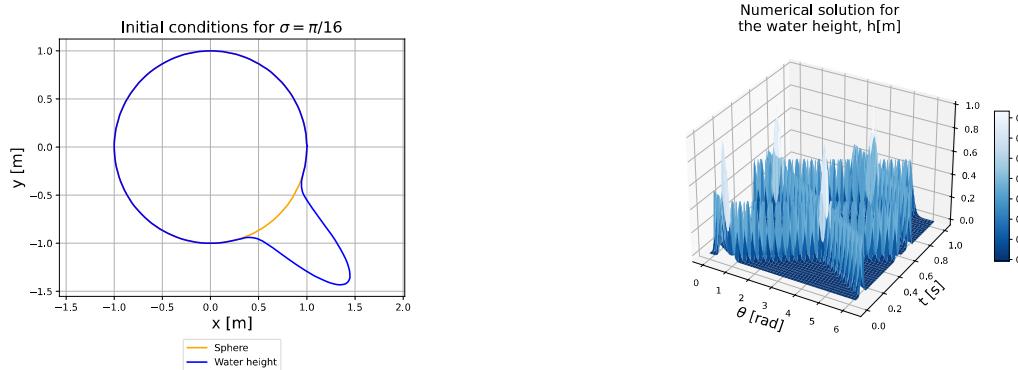
Model	Gauss initial condition				New initial condition		
	Epochs	MSE	MAE	Training time (s)	MSE	MAE	Prediction time (s)
CNN	500	5.16e-06	7.86e-04	116.47	2.60e-05	1.29e-03	0.08
FNO	500	1.75e-05	1.35e-03	970.65	1.38e-05	1.40e-03	0.39

Table 7.2: Test loss in terms of MSE and MAE, and time for training the models for the 2D SWE.

From Table 7.2, we see that both models has a low MSE and MAE for the Gaussian initial conditions, and hence perform well. We note that the training time for the FNO model is significantly higher than for the CNN model.

## 7.2 1D linearized Shallow Water Equations in Spherical Coordinates

In this section, we consider the linearized shallow water equations (LSWE) in spherical coordinates in a 1D setting on a circular domain. The length of the domain corresponds to the circumference of the circle,  $L = 2\pi$  m, and is discretized into  $N = 500$  points. The initial conditions is specified as a Gaussian function wrapped around the circle as given in (5.1.2). We use the middle value of  $\sigma$ , i.e.,  $\sigma = \frac{\pi}{16}$ , to generate the initial conditions. The initial conditions can be seen in Figure 7.10a. To get an overview of how the solution evolves, we have plotted the numerical solution in the  $\theta, t$ -plane from  $t = 0$  s to  $t = 1.0$  s, shown in Figure 7.10b.



(a) Initial conditions for the 1D LSWE on a sphere.  
 (b) Numerical solution of the 1D spherical LSWE in the  $\theta, t$ -space.

Figure 7.10: Visualization of the 1D LSWE on a sphere: (a) shows the initial conditions, and (b) presents the numerical solution.

In Figure 7.10b, we see that the solution has some steep descents, and it is interesting to see how the data-driven models handle these sharp edges. Some details of the data such as the number of training, validation and test points, and the grid spacing can be seen in Table 7.3.

Case	n_train	n_val	n_test	N	$\Delta\theta$	$\Delta t$
1D spherical LSWE	240	80	81	500	$\frac{2\pi}{500} = 0.0126 \text{ rad}$	0.0025 s

Table 7.3: Details of the used data for the case with the 1D LSWE in spherical coordinates.

In Table 7.3, we see that in contrast to the 1D SWE, the time step size  $\Delta t$  is constant. From the number of training, validation and test points, we see that the data is split into 60% training, 20% validation and 20% test data, like for the 1D SWE with Gaussian initial conditions.

### CNN Model

We define and train a CNN model to solve the 1D LSWE in spherical coordinates. The CNN model is trained on the data from  $t = 0$  to  $t = 0.6$  s, validated on the data from  $t = 0.6$  to  $t = 0.8$ , and tested on the data from  $t = 0.8$  to  $t = 1.0$  s, corresponding to the The model is trained using 500 epochs. The training and validation loss is shown in Figure 7.11.

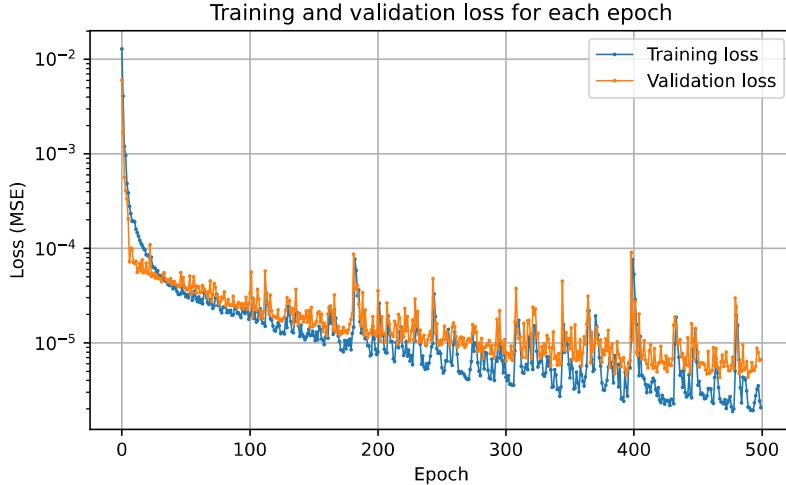


Figure 7.11: Training and validation loss for the CNN model for the 1D spherical LSWE.

In Figure 7.11, we see that the training and validation loss decrease over the epochs, indicating that the model is learning the dynamics of the solution. We note that the cross between the training and validation loss is quite later, which we also observed for the CNN model in the case with the 1D SWE with Gaussian initial conditions. To see how the model performs, we consider the error plots in Figure 7.12.

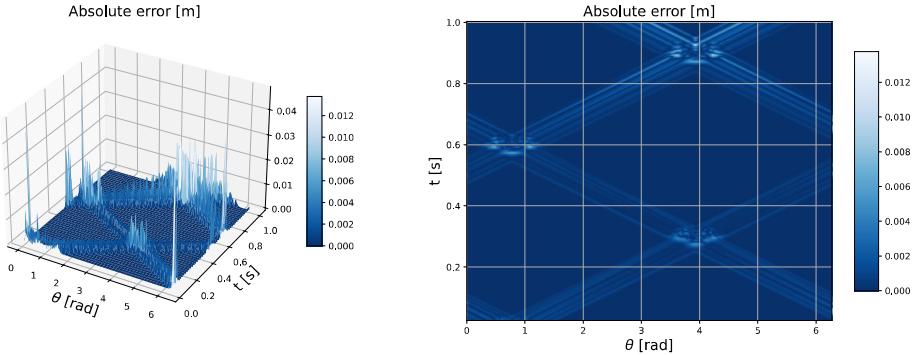


Figure 7.12: Error plots for the predictions of the CNN model for solving the 1D linearized spherical SWE.

In Figure 7.12, we see that errors are largest at the sharp edges of the solution, which is expected as the solution tends to be discontinuous. The predictions for some given time steps are shown in Figure 7.13.

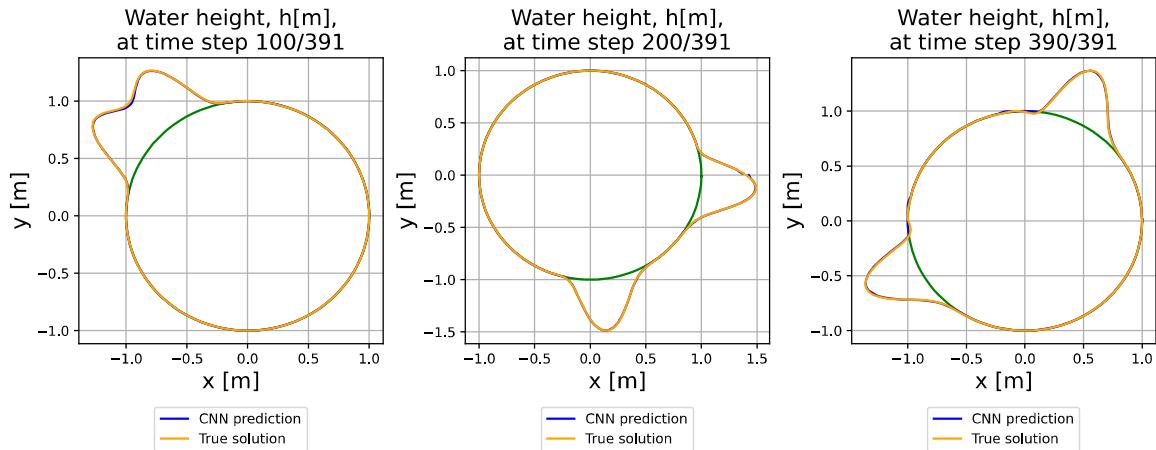


Figure 7.13: Predictions for the spherical 1D LSWE using the CNN model for some given time steps.

From Figure 7.13, we see that the predictions capture the waves.

### FNO model

The FNO model consists of an input channel, 64 hidden channels and an output channel. We use a Fourier basis with 16 modes and a batch size of 32. The model is trained using the Adam optimizer with a learning rate of 0.001 and the criteria is to minimize the mean squared error (MSE). The model is trained on the data from  $t = 0$  s to  $t = 0.6$  s, validated on the data from  $t = 0.6$  s to  $t = 0.8$  s, and tested on the data from  $t = 0.8$  s to  $t = 1.0$  s. The model is trained for 200 epochs, where the current best model is saved throughout the training. The training and validation loss is shown in Figure 7.14.

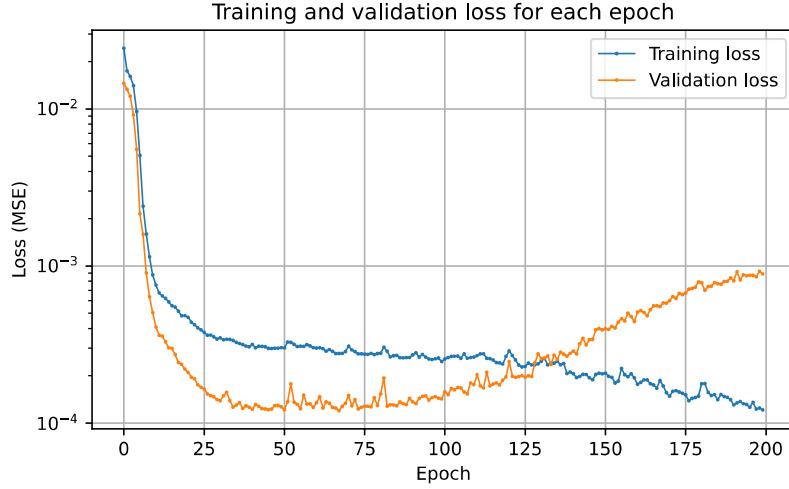


Figure 7.14: Training and validation loss for the FNO model for the spherical shallow water equations in 1D.

Figure 7.14 shows that the training and validation loss decrease over the epochs, indicating that the model is learning the dynamics of the solution. We see that after some time the validation loss increases, indicating that the model is overfitting the training data. This also shows why it can be beneficial to use early stopping. The error plots are shown in Figure 7.15.

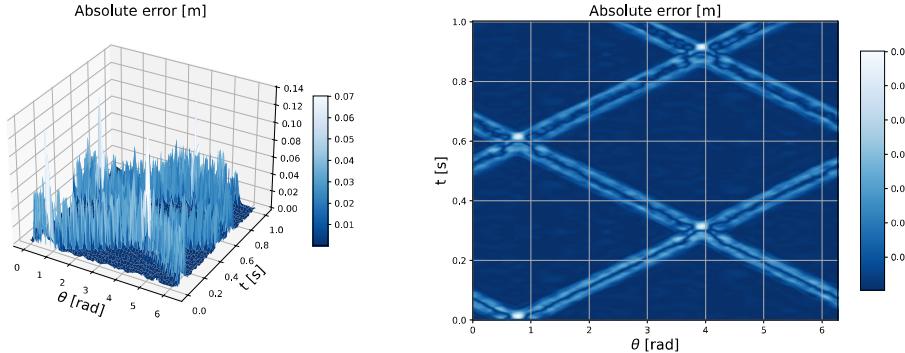


Figure 7.15: Error plots for the predictions of the 1D linearized spherical SWE.

Similar to the CNN model, we observe from Figure 7.15 that the error is largest at the sharp edges of the solution. The predictions for some given time steps are shown in Figure 7.16.

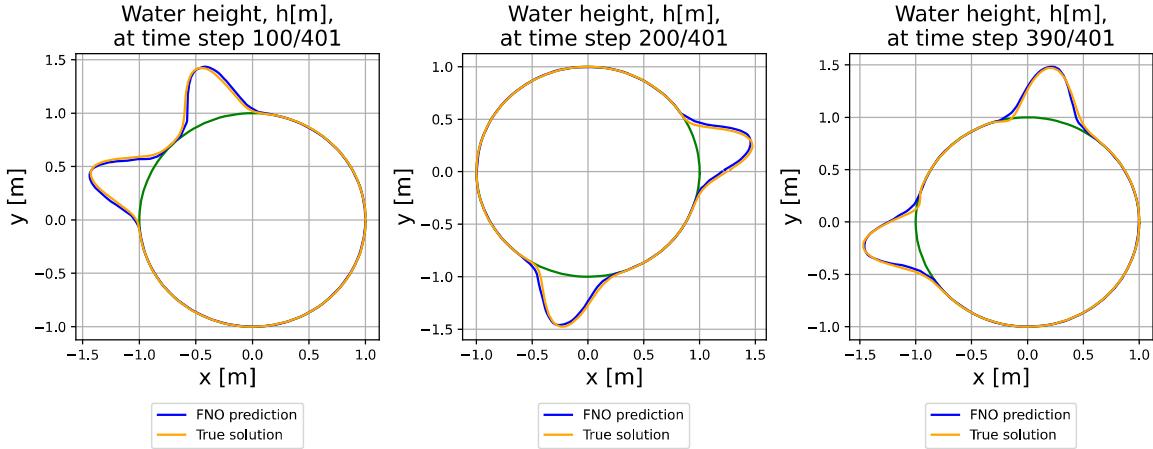


Figure 7.16: Predictions for the spherical shallow water equations in 1D using the FNO model for some given time steps.

Figure 7.16 demonstrates that the FNO model captures the waves, but has some minor delays.

### Comparison

To compare the performance of the CNN and FNO models, we consider the MSE and MAE for the predictions of the 1D spherical LSWE, as well as the training time for the models. We are also interested in measuring how the models handle sharp edges in the solution, tending to discontinuities. Therefore, we introduce two new initial conditions for the water height  $h$ , as described in (5.1.2), with  $\sigma = \frac{\pi}{8}$  and  $\sigma = \frac{\pi}{32}$ . This is done to see how the models perform for different types of initial conditions, depending on how smooth or discontinuous the solution is. The results are summarized in Table 7.4.

Model	$\sigma = \pi/8$			$\sigma = \pi/16$			$\sigma = \pi/32$		
	MSE	MAE	Time [s]	MSE	MAE	Time [s]	MSE	MAE	Time [s]
CNN	2.66e-06	8.27e-04	134.43	1.34e-05	1.60e-03	159.14	8.26e-04	1.11e-02	142.33
FNO	1.63e-04	7.51e-03	229.75	2.49e-04	7.80e-03	186.91	8.90e-04	1.46e-02	201.29

Table 7.4: Test loss in terms of MSE and MAE, and time for training the models for the 1D spherical SWE.

From Table 7.4 we observe that the CNN model is slightly faster and achieves a lower MSE and MAE than the FNO model across all three initial conditions. However, the difference for  $\sigma = \pi/32$  is minimal, suggesting that the FNO model performs better at capturing discontinuities in the solution. Additionally, the FNO model exhibits more stable performance with new initial conditions, while the CNN model shows more variation.

It is also notable that the MAE is generally higher than the MSE and increases as  $\sigma$  decreases. This difference highlights that the MAE places greater emphasis on small errors compared to the MSE.

### 7.3 2D Shallow Water Equations with Gaussian initial conditions

In this section we present the results for the 2D SWE using data-driven models. The initial condition, a Gaussian function, is the same as in the 1D case but extended to two dimensions. We the 2D SWE using both a CNN and

and FNO model, comparing their performance in terms of run time and accuracy. Additionally, we compare their run time to the FVM to evaluate whether the data-driven models can serve as a faster alternative. We also assess the models' ability to generalize to a finer grid by training them on a coarse grid and making predictions of a fine grid. Finally, we evaluate the models' capability to generalize further in time, testing their long-term predictive performance.

The initial condition for the 2D problem is a Gaussian function as given in (5.1.3). The initial condition is illustrated in Figure 7.17.

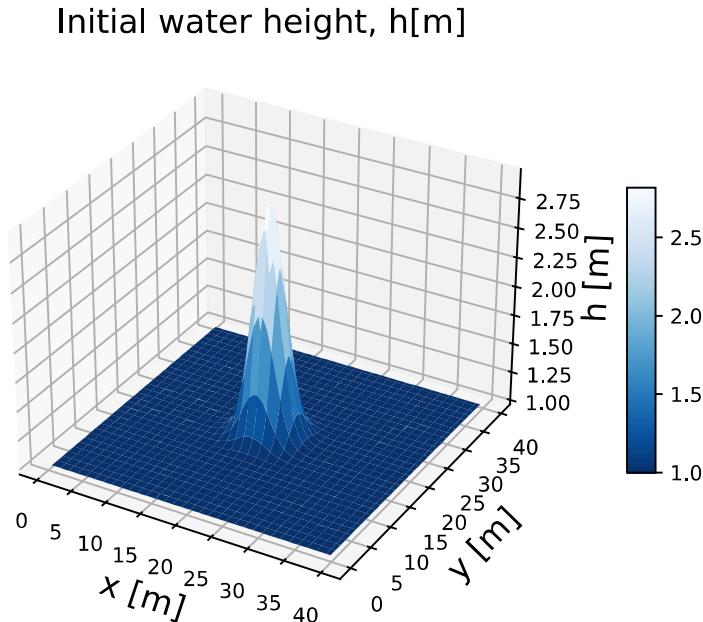


Figure 7.17: Initial condition for the 2D SWE.

In Figure 7.17 we see the initial condition for the 2D SWE problem, which is close to the 2D dam break problem, only here with a Gaussian function. The solution is generated from  $t = 0$  to  $t = 5$  s. Some information of the data used for the 2D SWE generated by the FVM can be found in Table 7.5.

Case	n_train	n_val	n_test	$\Delta x$	$\Delta t$
2D SWE, N = 64	48	16	17	0.625 m	[0.050 s, 0.073 s]
2D SWE, N = 128	99	33	33	0.3125 m	[0.025 s, 0.035 s]

Table 7.5: Details of the used data for the case with the 2D SWE for both a number of grid points  $N = 64$  and  $N = 128$ .

In Table 7.5 we see that the train/validation/test data split is 60%/20%/20% for both  $N = 64$  and  $N = 128$ . In the FVM we use a variable time step size  $\Delta t$  to ensure stability, which is why the time step size varies. We see that for  $N = 64$  the time step size varies between 0.050 s and 0.073 s, while for  $N = 128$  the time step size varies between 0.025 s and 0.035 s. The time step size for the higher  $N$  is in general smaller than for the lower  $N$  to ensure stability. This is also what gives more data points for  $N = 128$  compared to  $N = 64$ .

### CNN Model

We train a CNN model with several convolutional layers and ReLU activation functions. We use the Adam optimizer with a learning rate of 0.001 and a batch size of 32. The model is trained on the data for 60% of the time steps, validated on 20% of the time steps, and tested on the last 20% of the time steps. We make predictions from  $t = 0$  to  $t = 5$ . The model is trained for 500 epochs. The training and validation loss for the 2D CNN model can be seen in Figure 7.18.

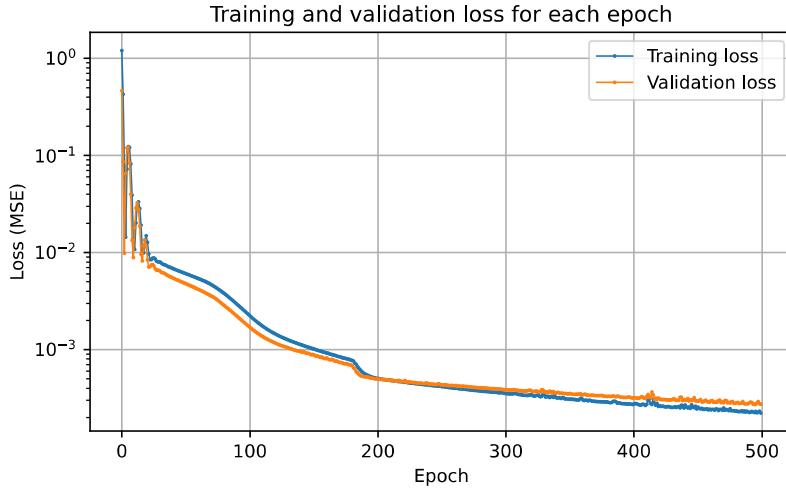


Figure 7.18: Training and validation loss for the 2D CNN model.

Figure 7.18 shows that the training and validation loss are decreasing, as a function of the number of epochs. The error plot for the last prediction for the 2D CNN can be seen in Figure 7.19.

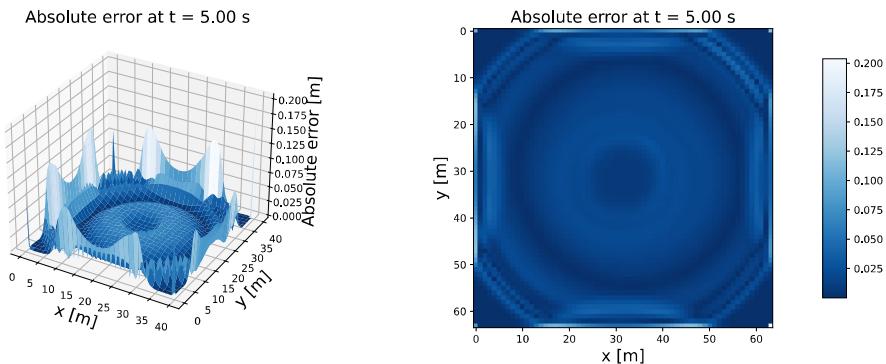


Figure 7.19: Error plot for the last prediction for the 2D CNN.

From Figure 7.19, we see that overall the absolute error is small, meaning that the CNN model is able to learn the dynamics of solution. The error is largest at the boundaries. This may be due to the fact that in this case we are working with boundary conditions simulating a wall. Hence, when the wave hits the wall, the wave is reflected, and the model has to learn the reflection of the wave. This may also lead to discontinuities in the solution, which can be difficult for the model to handle.

### FNO Model

We train a FNO model with the same training/validation/testing split as for the CNN model. The FNO model uses 8 Fourier modes, the Adam optimizer with a learning rate of 0.001, and a batch size of 10. The model is trained for 500 epochs. The training and validation loss for the 2D FNO model can be seen in Figure 7.20.

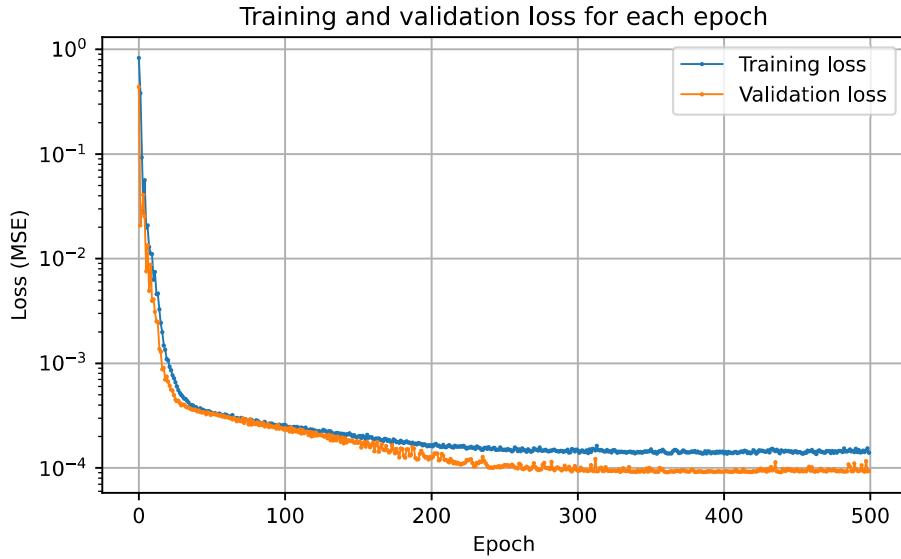


Figure 7.20: Training and validation loss for the 2D FNO model.

From Figure 7.20 we see that the training and validation loss are decreasing, and more or less stable with some fluctuations. The plot suggests that the model has converged. The error plot for the last prediction for the 2D FNO can be seen in Figure 7.21.

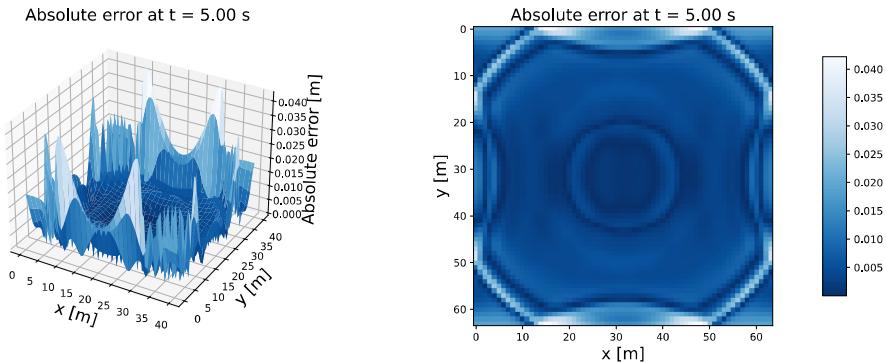


Figure 7.21: Error plot for the last prediction for the 2D FNO.

As for the CNN model, the error is largest at the boundaries. However, we notice that the absolute error is smaller for the FNO model compared to the CNN model.

### Comparison

We compare the performance of the CNN and FNO models in terms of the MSE and MAE for the predictions. We test for both  $N = 64$  and  $N = 128$  grid points in each direction. The errors are calculated for the test data, i.e., the last 20% of the time steps. The results are summarized in Table 7.6, together with the time for training the models.

Model	$N = 64$				$N = 128$			
	Epochs	MSE	MAE	Time (s)	Epochs	MSE	MAE	Time (s)
CNN	500	4.96e-04	1.53e-02	65.18	100	1.34e-03	2.16e-02	114.00
FNO	500	1.56e-04	8.34e-03	549.55	100	4.63e-04	2.63e-02	1411.42

Table 7.6: Test loss in terms of MSE and MAE, and time for training the models for the 2D SWE.

We see that the FNO model in general needs fewer epochs to converge compared to the CNN model, but it also takes longer time to train. From the theory we know that FNO are supposed to work when we are training on a coarse grid and then make predictions on a fine grid. To test this, we will train the models on a coarse grid and then make predictions on a fine grid. The table below shows the results when the models are trained on a grid with  $N = 64$  and subsequently making predictions on a finer grid with  $N = 128$ .

Model	$N = 128$			$N = 256$		
	MSE	MAE	Prediction time [s]	MSE	MAE	Prediction time [s]
CNN	3.97e-04	1.47e-02	0.10	3.80e-04	1.48e-02	1.43
FNO	9.32e-05	6.20e-03	1.34	7.30e-05	5.15e-03	20.43

Table 7.7: Test loss in terms of MSE and MAE, and time for training the FNO model on a grid with  $N = 64$  and then making predictions on a grid with  $N = 128$  and  $N = 256$ .

From the results in Table 7.7 we see that the errors for the FNO model are small, meaning that the FNO model is able to generalize to a finer grid. The CNN model has a higher error and if it is accurate enough depends on the application. This way, it is possible to train the FNO model on a coarse grid and then make predictions on a fine grid, which is a great advantage when solving the SWE numerically, as it is very computationally expensive to solve the SWE on a fine grid using the FVM. By comparing the run time of the FVM, see Table 6.3, we observe that the data-driven models offer an alternative, which may be advantageous depending on the specific application. This approach provides a potential solution to the scalability challenges we are facing when solving the SWE numerically. To test this ability further, we will train the models on a grid with  $N = 64$  and then make predictions on a grid with  $N = 256$ . The results are summarized in Table 7.7.

The results in Table 7.7 show that the FNO model is able to generalize to a grid with  $N = 256$ . While the errors are a bit higher than for the predictions on a grid with  $N = 128$ , the errors are still small, and for many applications the errors may be acceptable. The results for the CNN model are not as good as for the FNO model, but the CNN model is still, to some extent, able to generalize to a finer grid.

### Long-term predictions

We are particularly interested in evaluating the models' ability to generalize further in time. To do this we train the model on the generated 2D data with a fixed time step size of  $\Delta t = 0.025$ . To test this, we will train the models on a time interval  $[0, 5]$  and make predictions for up to  $n = 200$  time steps in the future, equivalent predicting up to  $t = 10$  s.

Initially, we adopted an iterative approach, where the model made a prediction for the next time step and used that prediction as input for the next time step. While this approach produces acceptable results for the initial time steps, the error increases as we make predictions further in time, probably due to accumulating errors. To address this issue, we implemented an alternative approach by creating sequences of data and training the model to predict the next time step based on the entire sequence. In this method, the most recent prediction becomes part of a sequence, distributing the influence across subsequent predictions, rather than heavily impacting the next prediction alone. This approach aims to stabilize long-term predictions. The error plot for the long-term predictions from the CNN model can be seen in Figure 7.22.

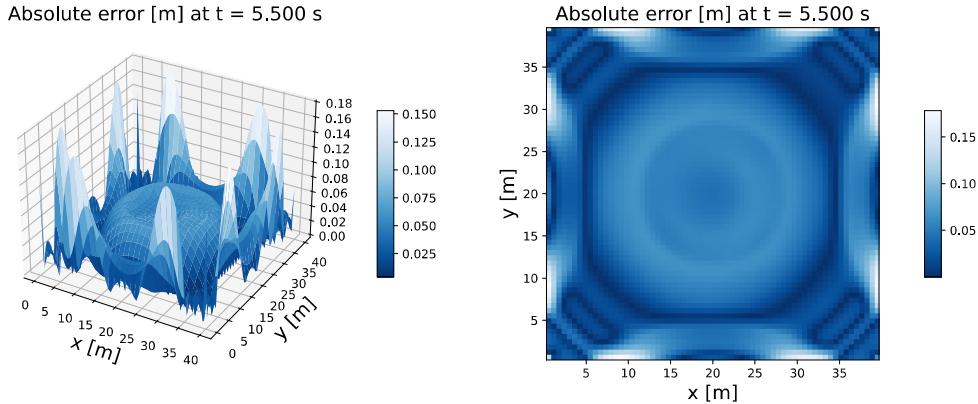


Figure 7.22: Error plot for the long-term prediction for the 2D CNN model.

Figure 7.22 shows that the error is larger for the short-term predictions in Figure 7.19 compared to the long-term predictions. But it has not increased drastically. We run the same test for the FNO model. The error plots of the long-term predictions from the FNO model after 20 time steps, equivalent to  $t = 0.5$  s, can be seen in Figure 7.23.

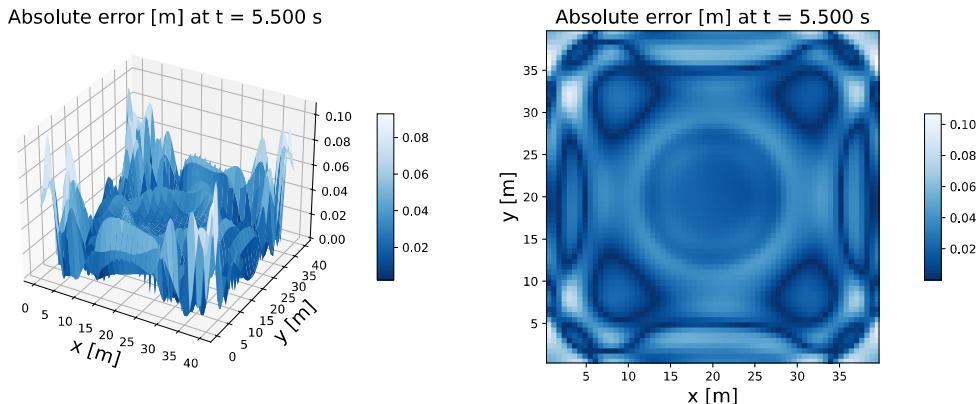


Figure 7.23: Error plot for the long-term prediction for the 2D FNO model.

From the error plot in Figure 7.23, we see that the error is bigger than for the short-term predictions in Figure 7.21, but for many applications still acceptable. We also see that error for the FNO model is smaller than for the CNN model, indicating that the FNO model is able to generalize further in time and make long-term predictions. A plot of the predictions and the ground truth can be found in Figure A.13 in Appendix section A.3.

The corresponding error plot for  $n = 40$  time steps can be seen in Figure A.14 in Appendix section A.3, where we see that the error has increased.

To get an overview of how the model performs over time, we plot the error in each time step for the long-term prediction for up to  $n = 200$  time steps in the future. To compare the error we have plotted the MSE, MAE and the max error for the long-term predictions for the CNN and FNO models in Figure 7.24.

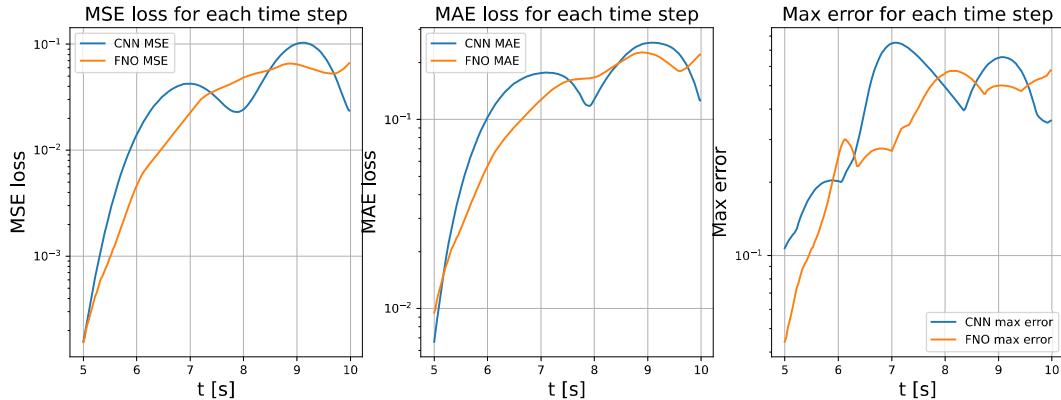


Figure 7.24: MSE, MAE and max error for the long-term predictions for the 2D CNN and FNO models.

When making predictions, it is essential to consider the physical properties of the system, rather than relying solely on the loss functions. The choice of error metric should align with the goals of the predictions. For instance, it may be more important to minimize the MSE or the maximum absolute error, depending on the physical properties of the system. This raises a key question: is it better to have one large error or many small errors?

As shown in Figure 7.24, the error increases as we make predictions further in time. Overall, the FNO model demonstrates lower errors compared to the CNN model, indicating it may offer more accurate long-term predictions.

## **Chapter 8**

## **Discussion**

# **Chapter 9**

## **Conclusion**

- By training the FNO on a coarse grid and evaluate on a fine grid, the run time is XX times faster, while maintaining the same accuracy.
- Long-term prediction: the FNO can predict the solution for a longer time than the training time. It was demonstrated that training on a coarse grid was not suitable to facilitate a long-term prediction for a fine grid.
- Long-term prediction: the FNO model outperforms the CNN model for long-term prediction.
- Grid independence: when training on a coarse grid and evaluating on a fine grid, the predictions by the FNO is much better than the predictions by the CNN. Aligning with the theory that FNOs are grid independent.
- Efficiency: CNN is faster than FNO, but FNO is more accurate. Which we prefer depends on the application.
- CNN has shown potential for many of the same abilities as the FNO - especially in the 1D case.
- Some of the theoretical advantages of the FNO are better realized in the 2D case than in the 1D case.
- Which model we choose is depending on the application.

### **9.1 Further work**