Finite Volume Methods for the One-Dimensional Shallow Water Equations

Sudi Mungkasi

October 2008

A thesis submitted in partial fulfilment of the requirements for the degree of Master of Mathematical Sciences of The Australian National University
PLAGIAT MERUPAKAN TINDAKAN TIDAK TERPUJI
I wish to dedicate this work to my parents, Sulasmi and Wagimin, and to my fiancée, Asti, for their unconditional love.
PLAGIAT MERUPAKAN TINDAKAN TIDAK TERPUJI
Declaration

The work in this thesis is my own except where otherwise stated.

Sudi Mungkasi
PLAGIAT MERUPAKAN TINDAKAN TIDAK TERPUJI
Acknowledgements

I would like to thank AusAID for the Australian Partnership Scholarship so that I could do a Master of Mathematical Sciences at The Australian National University in general and complete this thesis in particular.

Special thanks are due to Associate Professor Stephen Roberts for his supervision on this work. He deserves high honors for carefully guiding me in this research.

I am indebted to many friends or colleagues, but it would take a great deal of space to mention all of them. However, at least I must mention some who have offered substantial support in this research. I would like to thank John Jakeman for the discussions on shallow water equations and on Python programming. Thanks must also go to Hendra Gunawan Harno for several suggestions, and to Gail Craswell, Rhys Michie and Lucille Pedersen for proof-reading the draft of this thesis.

I would also like to express gratitude to my parents and my fiancée for their support, encouragement, and unconditional love.
Abstract

This thesis examines the numerical solutions to the one-dimensional shallow water equations. These solutions are obtained by use of well-balanced finite volume methods, while the well-balanced terms added in the numerical flux computations are based on the steady state of the lake at rest. In addition, the finite volume method being applied is the Kurganov’s central-upwind method, which is a Godunov-type method. Here, the simulations are done to test well-balanced central-upwind finite volume methods with two different sets of reconstructions, namely: stage and momentum, and stage and velocity reconstructions.

The well-balanced central-upwind finite volume methods with stage and momentum reconstructions cannot in general solve the unsteady state problems, such as oscillations in a parabolic canal and dam-break problem in some cases, but these methods work much better to solve steady flow problem than those without well-balanced terms. The performance of the methods with these reconstructions are very dependent on the type of slope limiter being used. These methods using the van Leer slope limiter lead to better results than those using other limiters.

On the other hand, the well-balanced central-upwind finite volume methods with stage and velocity reconstructions are able to capture both unsteady and steady states of water flows. It is an advantage that the performance of the methods are not too dependent on the slope limiter, despite the fact that the methods with these reconstructions using superbee limiter yield the smallest error with relatively fast computations. Moreover, the methods with stage and velocity reconstructions result in better performance than those with stage and momentum reconstructions.
PLAGIAT MERUPAKAN TINDAKAN TIDAK TERPUJI
Contents

Acknowledgements vii
Abstract ix
Notations and Terminologies xv

1 Introduction 1
1.1 Motivation ............................................ 1
1.2 Purpose ................................................ 3
1.3 Outline .................................................. 3

2 One-Dimensional SWE 5
2.1 Classification of Fluid Flows ......................... 5
2.2 Derivation of SWE .................................... 7
  2.2.1 Conservation of mass .......................... 8
  2.2.2 Conservation of momentum .................. 11
2.3 Hyperbolic Problems ................................ 14
  2.3.1 Linear Hyperbolic Equations ................. 17
  2.3.2 Domain of Dependence and Range of Influence 19
  2.3.3 Riemann Problem ............................ 20
  2.3.4 Initial-Boundary-Value Problem ............ 22
2.4 Properties of SWE ................................... 23
  2.4.1 Notion of a Simple Wave ................. 24
  2.4.2 Entropy Inequality .......................... 26
2.5 Shock Conditions .................................... 26
2.6 Concluding Remarks ................................ 29
3 Analytical Solutions

3.1 Dam-Break Problems
  3.1.1 Dry-Dam Problem
  3.1.2 Finite Water Depth Problem

3.2 Oscillations in a Parabolic Canal

3.3 Steady Flow

3.4 Concluding Remarks

4 Finite Volume Methods

4.1 Approximation of the Quantities and Fluxes

4.2 Velocity Stabilization

4.3 Slope Limiting

4.4 Temporal Discretisation

4.5 Boundary Conditions

4.6 Properties of Finite Volume Methods
  4.6.1 Convergence
  4.6.2 Stability
  4.6.3 Consistency

4.7 Concluding Remarks

5 Well-Balanced Finite Volume Methods

5.1 Basic Concepts of Well-Balancing

5.2 Semi-Discrete First Order Well-Balanced Schemes

5.3 Fully Discrete First Order Well-Balanced Schemes

5.4 Second Order Well-Balancing

5.5 Higher Order Well-Balancing

5.6 Quantities Reconstruction

5.7 Concluding Remarks

6 Stage and Momentum Reconstructions

6.1 Test 1: Dam-Break Problems
  6.1.1 Accuracy
  6.1.2 Efficiency
CONTENTS

6.2 Test 2: Oscillations in a Parabolic Canal . . . . . . . . . . . . . . 83
6.3 Test 3: Steady Flow . . . . . . . . . . . . . . . . . . . . . . . . 85
6.4 Concluding Remarks . . . . . . . . . . . . . . . . . . . . . . . . 86

7 Stage and Velocity Reconstructions 89
7.1 Test 1: Dam-Break Problems . . . . . . . . . . . . . . . . . . . . 89
  7.1.1 Accuracy . . . . . . . . . . . . . . . . . . . . . . . . . . . . 90
  7.1.2 Efficiency . . . . . . . . . . . . . . . . . . . . . . . . . . . . 92
7.2 Test 2: Oscillations in a Parabolic Canal . . . . . . . . . . . . . . 93
7.3 Test 3: Steady Flow . . . . . . . . . . . . . . . . . . . . . . . . . 96
7.4 Concluding Remarks . . . . . . . . . . . . . . . . . . . . . . . . . 96

8 Conclusions 99

A C Tutorial 101
A.1 Preliminaries for C . . . . . . . . . . . . . . . . . . . . . . . . . . 101
A.2 Types, Operators, and Expressions . . . . . . . . . . . . . . . . . 104
A.3 Control Flow . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 106
A.4 Functions and Program Structures . . . . . . . . . . . . . . . . . 108
A.5 Pointers and Arrays . . . . . . . . . . . . . . . . . . . . . . . . . 110
A.6 Structures . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 112
A.7 Input and Output . . . . . . . . . . . . . . . . . . . . . . . . . . . 113

B C-Python Tutorial 115
B.1 Combining Python with C . . . . . . . . . . . . . . . . . . . . . . 117
B.2 Calling C from Python . . . . . . . . . . . . . . . . . . . . . . . . 117
B.3 Example I: Dot Product of Vectors . . . . . . . . . . . . . . . . . 118
B.4 Example II: Unit Testing . . . . . . . . . . . . . . . . . . . . . . . 118
B.5 Example III: C-Code for Dot-Product of Vectors . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 120

Bibliography 123
PLAGIAT MERUPAKAN TINDAKAN TIDAK TERPUJI
Notations and Terminologies

Notations and terminologies which are mostly used are briefly described as follows. The others will be described within the corresponding chapters in this thesis.

**Notation**

\( x \) coordinate in one-dimensional space in the horizontal direction with a unit of metre (m)

\( t \) time variable with a unit of second (s)

\( \rho \) water density which is assumed to be constant

\( u(x,t) \) water velocity at position \( x \) and at time \( t \) with a unit of m/s²

\( h(x,t) \) water depth which is assumed to vary with the position \( x \), and time \( t \) with a unit of m

\( z(x) \) water bed which is vary only with \( x \) with a unit of m

\( w(x,t) \) water stage defined as \( w(x,t) = z(x) + h(x,t) \) with a unit of m

\( g \) gravitational acceleration which is constant and is taken to be 9.81 m/s²

\( p(x,t) \) pressure at point \((x,t)\) with a unit of kg/s²

\( \text{cst} \) an abbreviation for constant
quantity vector where for one-dimensional shallow water equations $q = (h, hu)^T$

flux function where for one-dimensional shallow water equations $F(q) = (hu, hu^2 + gh^2/2)^T$

numerical approximation to the solution $q$, where the subscript on $Q$ denotes spatial locations, such as the $i$th grid cell, and the superscript $n$ denotes time level $t_n$. When $Q$ lacks a temporal superscript, the current time level $t_n$ is intended.

Terminology

Lagrangian Description of motion where individual particles are observed as a function of time.

Eulerian Description of motion where the flow properties are function of both space and time.

Flow field The region of flow of interest.

Steady flow Flow where quantities do not depend on time, that is $u_t = p_t = \rho_t = 0$.

Bed elevation Bottom topography of the water. It is also known in several references as water bed, bed, topography, elevation, or bathymetry.
List of Figures

2.1 Shallow water flows in one-dimension .................................. 9
2.2 The inflow and outflow of the control volume .......................... 10
2.3 Pressure in a slope area .................................................. 11
2.4 Alternative illustration of shallow water flows in one-dimension ... 13
2.5 A typical hyperbolic system of two equations with $\lambda_1 < 0 < \lambda_2$, (a) shows the domain of dependence of the point $(X, T)$, and (b) shows the range of influence of the point $x_0$ ................... 19
2.6 Solution to the Riemann problem for system of two equations in the $x,t$-plane ......................................................... 22
2.7 Region containing straight characteristics .............................. 25
2.8 Discontinuity conditions .................................................... 27
3.1 Initial water profile of a dam ............................................. 32
3.2 Water profile of a dry-dam problem at some time $t_1 > 0$ ........ 33
3.3 Typical solution of dam-break problem with a finite water depth downstream at $t = t_1 > 0$ .......................... 36
3.4 A typical representation of a planar free surface in a parabolic canal 40
3.5 A typical representation of a steady flow with an obstruction .... 44
4.1 Updating the cell averages $Q_i^n$ by fluxes at the cell edges ....... 47
6.1 Well-balanced central-upwind with $w$-$uh$ reconstructions using 100 cells, first-order spatial discretisation, and first-order time-stepping 75
6.2 Well-balanced central-upwind with $w$-$uh$ reconstructions using 400 cells, first-order spatial discretisation, and first-order time-stepping 75
LIST OF FIGURES

6.3 Well-balanced central-upwind with \( w_{uh} \) reconstructions using 400 cells, second-order spatial discretisation, first-order time-stepping, and minmod limiter ......................................................... 76

6.4 Well-balanced central-upwind with \( w_{uh} \) reconstructions using 400 cells, second-order spatial discretisation, second-order time-stepping, and minmod limiter ......................................................... 76

6.5 Well-balanced central-upwind with \( w_{uh} \) reconstructions using 400 cells, second-order spatial discretisation, first-order time-stepping, and van Leer limiter ................................................................. 78

6.6 Well-balanced central-upwind with \( w_{uh} \) reconstructions using 400 cells, second-order spatial discretisation, second-order time-stepping, and van Leer limiter ................................................................. 78

6.7 Well-balanced central-upwind with \( w_{uh} \) reconstructions using 400 cells, second-order spatial discretisation, second-order time-stepping, and superbee limiter ............................................................... 79

6.8 Well-balanced central-upwind with \( w_{uh} \) reconstructions using 800 cells, second-order spatial discretisation, second-order time-stepping, van Leer limiter and 1 micrometre allowed minimum water depth 79

6.9 Initial water profile in a parabolic canal ................................................................. 84

6.10 Well-balanced central-upwind with \( w_{uh} \) reconstructions using 400 cells, second-order spatial discretisation, second-order time-stepping, and van Leer limiter at \( t = T/4 \) ................................................................. 84

6.11 Central-upwind without well-balanced terms using \( w_{uh} \) reconstructions, 400 cells, second-order spatial discretisation, second-order time-stepping, and van Leer limiter at \( t = T/4 \) ................................................................. 85

6.12 Steady flow using well-balanced central-upwind with \( w_{uh} \) reconstructions, and 400 cells ................................................................. 87

6.13 Steady flow using well-balanced central-upwind with \( w_{uh} \) reconstructions, and 1600 cells ................................................................. 87

7.1 Well-balanced central-upwind with \( w-u \) reconstructions using 400 cells and van Leer limiter ................................................................. 90

7.2 Well-balanced central-upwind with \( w-u \) reconstructions using 3200 cells and superbee limiter ................................................................. 92
LIST OF FIGURES

7.3 Well-balanced central-upwind with $w-u$ reconstructions using 400 cells and van Leer limiter for $t = T/4$ ........................................ 94
7.4 Well-balanced central-upwind with $w-u$ reconstructions using 400 cells and van Leer limiter for $t = T/2$ ........................................ 94
7.5 Well-balanced central-upwind with $w-u$ reconstructions using 400 cells and van Leer limiter for $t = 3T/4$ ........................................ 95
7.6 Well-balanced central-upwind with $w-u$ reconstructions using 400 cells and van Leer limiter for $t = T$ .......................................... 95
7.7 Well-balanced central-upwind with $w-u$ reconstructions using 400 cells and superbee limiter ...................................................... 97
7.8 Well-balanced central-upwind with $w-u$ reconstructions using 400 cells and minmod-Kurganov limiter ........................................ 97
List of Tables

6.1 Absolute errors in water depth, momentum and velocity using 100 cells, second order spatial, second order temporal discretisations and six different limiters ........................................... 81
6.2 Absolute errors in water depth, momentum and velocity using 400 cells, second order spatial, second order temporal discretisations and six different limiters ........................................... 81
6.3 Absolute errors in water depth, momentum and velocity using 1,600 cells, second order spatial, second order temporal discretisations and six different limiters ........................................... 81
6.4 Average time of 10 runs taken to simulate the dam-break problem ............................................................................. 82

7.1 Absolute errors in water depth, momentum and velocity using well-balanced central-upwind with \( w-u \) reconstructions using 100 cells and six different slope limiters ........................................... 91
7.2 Absolute errors in water depth, momentum and velocity using well-balanced central-upwind with \( w-u \) reconstructions using 400 cells and six different slope limiters ........................................... 91
7.3 Absolute errors in water depth, momentum and velocity using well-balanced central-upwind with \( w-u \) reconstructions using 1600 cells and six different slope limiters ........................................... 91
7.4 Average time of 10 runs taken to simulate the dam-break problem with \( w-u \) reconstructions ................................................................. 93
7.5 Absolute errors in water depth, momentum and velocity using well-balanced central-upwind with \( w-u \) reconstructions using 400 cells and six different final time ........................................... 93
PLAGIAT MERUPAKAN TINDAKAN TIDAK TERPUJI
Chapter 1

Introduction

This chapter describes the motivation, purpose and outline of this thesis working on Computational Fluid Dynamics. These three aspects are considered as a chronology of why, what, and how this thesis is done.

1.1 Motivation

Computational Fluid Dynamics (CFD) is concerned with the numerical solutions of differential equations which model fluid movement. CFD activity emerged and gained prominence with availability of computers in the early 1960s [15]. Since approximately 1970s there have been commercial computer software packages available, and since that time CFD has become a combination of physics, numerical mathematics, and computer sciences [4]. The combination is employed to simulate fluid flows such as gas and water movement.

Shallow Water Equations (SWE) model shallow water flow as the name suggests. In general, these equations are according to transport of mass and momentum. These equations form a nonlinear hyperbolic system, and often admit discontinuous solutions even when the initial condition is smooth. They have been widely used for many applications, for example: dam-breaks, tsunami propagation, storm surges, solute transport, river flows, and ecological models [27]. Exact solutions to these equations may be investigated by use of characteristics methods, but it is often difficult to get those solutions analytically. To deal with this difficulty, numerical methods, such as finite volume methods, have been developed to get the solutions to the SWE.
Finite volume methods are based on the integral form of conservation laws. These methods break the spatial domain into cells and approximate the average of quantity over each cell. In each time step, these values are updated by approximations to the flux through the boundaries of the cells. The main problem of these methods is to determine numerical flux functions which are able to approximate the correct fluxes as close as possible.

Finite volume methods are closely related to finite difference methods, and a finite volume method can often be well viewed as a finite difference approximation to the differential equation. However, since finite volume methods are derived on the basis of the integral form of conservation laws, they admit discontinuities unlike finite difference methods which are derived on the basis of the differential equations. This is why finite volume methods are able to solve the Riemann problem that is a hyperbolic equation together with piecewise constant initial data with a single jump discontinuity.

Many finite volume methods, such as Lax-Friedrichs, Lax-Wendroff, Richtmyer (as also known as Two-Step Lax-Wendroff), and Upwind Godunov Methods, have been developed to get an approximate flux function and have been tested [42]. Jakeman [27] has applied Kurganov’s central-upwind method, which is a finite volume Godunov-type method, with stage and momentum reconstructions for solving the one-dimensional SWE. The scheme in this method was developed by Kurganov, et al. [34, 36], and provide an approximate flux function, which leads to an approximate Riemann solver. It is common that a difficulty arises in the simulation of free surface flows due to the appearance of dry areas which are caused by the initial conditions or as a result of the fluid motion. For example, there exists oscillations around the interface of wet and dry-regions when Kurganov’s central-upwind method with stage and momentum reconstructions is applied, as can be seen in [27]. Because standard numerical schemes often fail in the presence of wet/dry situations by producing spurious result, modifications in the schemes need to be made [19].

It is a challenge to obtain and/or specify a method which is able to capture shock discontinuities, to compute smooth solutions and to compute stationary solutions properly. In order to capture stationary solutions, a balance between the flux and the source terms must be satisfied up to a discrete level when a source term is present [19, 50]. Methods which satisfy this property is said to be
1.2 Purpose

The purpose of this thesis is to study well-balanced central-upwind finite volume methods for solving one-dimensional SWE. The main idea is to properly combine the central-upwind schemes developed in [36] with the treatment of well-balanced terms in the flux function introduced in [50]. By adding well-balanced terms in the flux function of central-upwind schemes, it is hoped that the resulting methods which are well-balanced can solve the shallow water problems with a better performance.

Combining two tools stated above is by no means an easy task, as many difficulties appear. Values of parameters need to be chosen appropriately during the simulation, and several reconstructions are attempted to know if the methods work. This means that the quantities to be reconstructed have to be properly chosen so as to maintain the well-balanced property, and at the same time to preserve the nonnegativity of the water depth. In addition, schemes satisfying those requirements with inexpensive computation should be taken into account.

1.3 Outline

This thesis is divided into eight chapters. Most of the chapters finish with several concluding remarks of the corresponding topic presented therein.

Chapter 2 presents the basic knowledge related to the SWE. This includes the classification of fluid flow, derivation of the one-dimensional SWE, and properties of the SWE. Hyperbolic problems are also presented since the SWE form a system which admits hyperbolicity.

Chapter 3 shows how exact solutions to the SWE are derived analytically. Three types of problems consisting of dam-break problems, oscillations of planar surface in a parabolic canal, and steady flow with an obstruction are presented. These three types of analytical solutions will be used to test the performance of the numerical methods. By knowing these solutions, the error of each numerical test can be easily measured.
Chapter 4 describes how finite volume methods are developed. It is started by construction of the approximation of the quantities and fluxes, and followed by presentation on slope limiters to deal with second order spatial discretisation. First and second order temporal discretisation by use of Runge-Kutta methods are also viewed. To complete the presentation, three properties (consisting of convergence, stability and consistency) of finite volume methods as numerical methods are briefly addressed.

The well-balanced finite volume methods are developed in Chapter 5. Even though the well-balanced terms can be constructed from any steady state, the well-balanced methods being discussed in this thesis is limited to those based on the steady state of the lake at rest. It can be seen that the first order well-balanced scheme can be expanded into second order, and then any desired order of well-balanced terms can be obtained by extrapolation.

Simulations of well-balanced finite volume methods are done in Chapter 6 and 7. The finite volume methods being tested are Kurganov's central-upwind methods. Chapter 6 addresses the tests of well-balanced central-upwind methods with water stage and momentum reconstructions, while Chapter 7 presents the test results of the well-balanced central-upwind methods with stage and velocity reconstructions. All of the numerical methods are tested against the analytical solutions derived in Chapter 3.

Finally, conclusions are presented in Chapter 8.
Chapter 2

One-Dimensional Shallow Water Equations

This chapter presents the classification of fluid flows and the derivation of one-dimensional SWE. In addition, topic on hyperbolic equations are viewed as the SWE admit the hyperbolicity property, and properties of the SWE and shock conditions are also presented since it will be considered for the next discussion.

2.1 Classification of Fluid Flows

This section introduces general classification of fluid flows. The classification can be based on the dimension, fluid viscosity, random variation of the particles, and fluid density as described in [57].

a. One-, Two-, and Three-Dimensional Flows

In the Eulerian description, the velocity vector $\mathbf{V} = (u, v, w)^T$ depends on three spatial variables and time in general, that is, $\mathbf{V} = \mathbf{V}(x, y, z, t)$, where the velocity components $u$, $v$, and $w$ depend on $x$, $y$, $z$, and $t$; this means $u = u(x, y, z, t)$, $v = v(x, y, z, t)$, and $w = w(x, y, z, t)$. Because the velocity vector depends on three space coordinates, such a flow is a three-dimensional flow. The solutions to three dimensional flows remain three-dimensional even if the flow could be assumed to be steady, that is $\mathbf{V} = \mathbf{V}(x, y, z)$.

A three-dimensional flow can often be approximated as a two dimensional
flow, that is a flow in which the velocity vector depends only on two spatial variables. An example is a plane flow (consult [37, 57]) in which the velocity vector depends on two spatial coordinates \( x \) and \( y \). Another example is the flow over a wide dam far away from the ends; for this flow, the vertical velocity can be assumed to be zero everywhere, and the horizontal and lateral flows do not depend on \( z \)-coordinate but depend only on \( x \) and \( y \).

Similarly, a one-dimensional flow is a flow in which the velocity vector depends only on one spatial variable. For example, fluid flows along straight pipes or between parallel plates. For the flow along a straight pipe, the velocity at a point depends only on the the distance between that point to the surface of the pipe. Even if the flow is unsteady, the flow is always one-dimensional.

b. Viscous and Inviscid Flows

A viscous flow is a flow which includes the effects of viscosity, while an inviscid flow is one in which viscous effects do not significantly influence the flow and thus the viscosity of the fluid is neglected. To model an inviscid flow analytically, the viscosity is simply set to be zero so that all viscous effects are zero. To do so, it has to be ensured that the viscous effects are negligibly small.

It has been found from experience that the primary class of flows which can be modeled as inviscid flows is external flows, that is, flows which exist exterior to a body. Any viscous effects that may exist are confined to a thin layer, known as a boundary layer which is attached to the boundary. The velocity in a boundary layer is always zero at a fixed wall as a result of viscosity. When the boundary layers are very thin, then they can be ignored so that the flows are assumed to be inviscid.

Viscous flows include a class of internal flows, for example flows in pipes. A substantial amount of energy that must be used to transport oil and gas in pipelines may be lost because of the viscous effects. Such losses are directly caused by the zero velocity at the wall.

c. Laminar and Turbulent Flows

Fluid flow can be said to be either a laminar flow or turbulent flow. A flow with no significant mixing of particles but with significant viscous shear stresses is called
2.2. DERIVATION OF SWE

a laminar flow. For a water flow, if it is laminar then a dye injected into the flow will not mix with the neighbouring fluid except by molecular activity. It will retain its identity for a relatively long period of time. If the flow varies irregularly so that flow quantities show random variation then it is called a turbulent flow. When a water flow is turbulent and a dye is injected into it, then the dye will mix immediately. This is caused by random moving of the fluid particles.

\textbf{d. Incompressible and Compressible Flows}

Another classification of fluid flows is that the flows are said to be incompressible or compressible. An incompressible flow exists if the density of each fluid particle remains relatively constant as it moves through the flow field. Otherwise, it is a compressible flow which means that density variations influence the flow.

In general, incompressible flow does not require that the density is constant everywhere. However, if the density of the fluid is known to be constant, then it is clear that the flow is incompressible. It should be noted that constant density is more restrictive than incompressibility. Flows that involve adjacent layers of fresh and salt water, as happens when rivers enter the ocean, are incompressible flows with varying density.

\section*{2.2 The derivation of Shallow Water Equations}

The SWE are derived based on the fluid motion description. There are two types of fluid motion description, namely Langrangian and Eulerian.

Lagrangian description focuses on individual particles, and its motion is observed as a function of time. The position, velocity and acceleration of each particle are listed as $s(x_0, t)$, $u(x_0, t)$, and $a(x_0, t)$, then quantities* of interest can be calculated. Here, the point $x_0$ locates the starting point or the name of each particle. In the Lagrangian description many particles can be followed. However, this becomes difficult as the number of particles in a fluid flow becomes extremely large.

Eulerian description is an alternative to following each fluid particle separately. This identifies points in space then observes the velocity of particles pass-

*The usual quantities of interest are mass, momentum, and velocity.
CHAPTER 2. ONE-DIMENSIONAL SWE

ing each point. The rate of change of velocity as the particle pass each point can be observed by \( \partial u(x,t)/\partial x \), and the change of velocity with respect to time at each particular point can be observed by \( \partial u(x,t)/\partial t \). In this Eulerian description of motion, the flow properties (such as velocity) are functions of both space and time.

In the next presentation, equations describing the laws of conservation of mass and momentum are derived for one-dimensional shallow water dynamics problem. Jakeman [27] has adopted Eulerian approach to describe both laws. This thesis takes a different approach to derive the shallow water equations; the conservation of mass is described by Eulerian, but the conservation of momentum is described by Lagrangian.

2.2.1 Conservation of mass

Conservation of mass means that the mass is neither created nor destroyed. This implies that the total mass in the whole system is always the same at any time.

There are several assumptions involved in the derivation of conservation of mass equations. First, the water flow is assumed to be laminar that is the turbulent velocity is neglected. Second, the density, \( \rho \), of the water at each point is constant so that the water is incompressible. In addition, it is assumed that the water bed is impermeable because the mass is conserved. Therefore, the mass in any control volume (that is a particular volume or column of water being observed) can change only due to the flow acrossing the boundaries of the control volume.

In general, the water flow can be illustrated in Figure 2.1. The notations used in the figure are described as follows:

- \( x \) represents the distance variable along the flow,
- \( t \) represents the time variable,
- \( z(x) \) is the fixed water bed,
- \( h(x,t) \) is the depth of the water at point \( x \) and at time \( t \),
- \( w(x,t) = z(x) + h(x,t) \) is the absolute water level called stage,
- \( u(x,t) \) denotes the velocity of the water flow at point \( x \) and at time \( t \).

By following those notations, the total mass \( m \) of water in any control volume
2.2. DERIVATION OF SWE

![Diagram of shallow water flows in one-dimension](image)

Figure 2.1: Shallow water flows in one-dimension

$[x_1, x_2]$ is given by

$$m = \int_{x_1}^{x_2} \rho h(x, t) \, dx$$  \hspace{1cm} (2.1)

This expression can be obtained as follows. The mass density over the depth $\bar{\rho}$ at an arbitrary point $(x, t)$ is $\rho h(x, t)$ which can be calculated by integration of $\rho$ from $z(x)$ to $w(x, t)$ as

$$\bar{\rho}(x, t) = \int_{z(x)}^{w(x, t)} \rho \, dy = \rho h(x, t).$$

As a result, integrating $\rho h(x, t)$ from $x_1$ to $x_2$ leads to the total mass in the control volume as expressed in (2.1).

The rate of flow of water past any point $(x, t)$ over the depth is called mass flux, $f_1$ which is given by

$$f_1 = \bar{\rho}(x, t)u(x, t)$$  \hspace{1cm} (2.2)

By using (2.2) and the assumption that the mass can change only due to the flow
acrossing the boundaries of the control volume, it can be found that

\[
\int_{x_1}^{x_2} \rho h(x, t + \Delta t)dx = \int_{x_1}^{x_2} \rho h(x, t)dx + \int_{t}^{t+\Delta t} \rho h(x_1, s)u(x_1, s)ds - \int_{t}^{t+\Delta t} \rho h(x_2, s)u(x_2, s)ds
\]  

(2.3)

holds for any control volume. This means that the mass at time step \(t + \Delta t\) is equal to the mass at time \(t\) added by the flux moving into and substracted by the flux going out of the control volume during \(\Delta t\)-period. The description of this mass continuity as given by formulation (2.3) is illustrated in Figure 2.2.

Now, let \(\Delta x\) and \(\Delta t\) be small quantities, where \(\Delta x := x_2 - x_1\). By use of Taylor expansion, equation (2.3) can be written as

\[
\rho h(x, t + \Delta t)\Delta x = \rho h(x, t)\Delta x + \rho h(x - \Delta x/2, t) u(x - \Delta x/2, t) \Delta t \\
- \rho h(x + \Delta x/2, t) u(x + \Delta x/2, t) \Delta t + O((\Delta t)^3) + O((\Delta x)^3)
\]

which is equivalent to

\[
\frac{\rho h(x, t + \Delta t) - \rho h(x, t)}{\Delta t} = \frac{(\rho hu)|_{(x + \frac{\Delta x}{2}, t)} - (\rho hu)|_{(x - \frac{\Delta x}{2}, t)}}{\Delta x}
\]

(2.4)

by neglecting \(O(\Delta t^3)\) and \(O(\Delta x^3)\) terms. Dividing each term by \(\rho\), and as \(\Delta x\) and \(\Delta t\) approach zero, equation (2.4) becomes

\[
h_t + (uh)_x = 0
\]

(2.5)

The resulting equation (2.5) is then called the conservation of mass equation.
2.2. DERIVATION OF SWE

2.2.2 Conservation of momentum

In this subsection, the conservation of momentum equation using Newton’s second law of motion is presented. The law is that

\[ F = \frac{dp}{dt} \]

where the force \( F \) is defined as the rate of change of the momentum \( p \) with respect to \( t \).

The total momentum of water movement in any control volume from \( x_1 \) to \( x_2 \) at time \( t \) is denoted by \( p(t) \) where

\[ p(t) = \int_{x_1(t)}^{x_2(t)} \rho h(x,t) u(x,t) \, dx \]  

(2.6)

By assuming the hydrostatic pressure, the forces at points \( x_1 \) and \( x_2 \) over the depth at time \( t \) are

\[ F_1(t) = \frac{1}{2} \rho gh^2(x_1(t), t) \]

\[ F_2(t) = -\frac{1}{2} \rho gh^2(x_2(t), t) \]

respectively, where \( g > 0 \) is constant denoting the gravity acceleration. Furthermore,

![Diagram of pressure in a slope area](image)

Figure 2.3: Pressure in a slope area

the force over \( \Delta z \) as shown in Figure 2.3 is

\[ \Delta F_3 = -\rho gh(x,t) \Delta z \]
or can be written as
\[ \Delta F_3 = -\rho gh(x,t) \frac{\Delta z}{\Delta x} \Delta x \]
and therefore the force over the bottom of the control volume is
\[ F_3 = \int_{x_1}^{x_2} -\rho gh(x,t)z_x dx \]
Hence, the total force over the control volume denoted by \( F \) is the sum of \( F_1, F_2 \) and \( F_3 \), that is
\[ F = \frac{1}{2} \rho gh^2(x_1(t), t) - \frac{1}{2} \rho gh^2(x_2(t), t) - \int_{x_1(t)}^{x_2(t)} \rho gh(x,t) \frac{dz}{dx} dx. \] (2.7)

The first derivative of \( p \) with respect to \( t \) is
\[ \frac{dp}{dt} = \frac{d}{dt} \int_{x_1(t)}^{x_2(t)} \rho h(x,t)u(x,t) dx \]
Applying Leibniz’ rule to differentiate this integral yields the relation
\[ \frac{dp}{dt} = \int_{x_1(t)}^{x_2(t)} \frac{\partial}{\partial t} \rho h(x,t)u(x,t) dx + \rho h(x_2(t), t)u^2(x_2(t), t) - \rho h(x_1(t), t)u^2(x_1(t), t) \] (2.8)
According to Newton’s second law of motion, the result in equation (2.8) is equal to that in equation (2.7). Hence, for \( \Delta t \)-period it follows that
\[
\int_t^{t+\Delta t} \int_{x_1(t)}^{x_2(t)} (\rho hu)_t dx dt + \int_t^{t+\Delta t} \rho h(x_2(t), t)u^2(x_2(t), t) dt
- \int_t^{t+\Delta t} \rho h(x_1(t), t)u^2(x_1(t), t) dt = \int_t^{t+\Delta t} \frac{1}{2} \rho gh^2(x_1(t), t) dt
\int_t^{t+\Delta t} \frac{1}{2} \rho gh^2(x_2(t), t) dt - \int_t^{t+\Delta t} \int_{x_1(t)}^{x_2(t)} \rho gh(x,t)z_x dx dt \] (2.9)
Similar to (2.3), formulation (2.9) can be written as
\[ (hu)_t + (hu^2 + \frac{1}{2} gh^2)_x = -ghz_x \]
which is called the conservation of momentum equation.

Therefore, the SWE, also known as the Saint-Venant system, can be written as two simultaneous equations
\[
\text{SWE } \begin{cases} 
  h_t + (hu)_x = 0 \\
  (hu)_t + (hu^2 + \frac{1}{2} gh^2)_x = -ghz_x 
\end{cases} \] (2.10)
2.2. DERIVATION OF SWE

These two equations are the differential equations of the nonlinear shallow water theory. Once the initial state of the fluid is prescribed, that is once the values of $u$ and $h$ at the time $t = 0$ are given, the equations (2.10) yield the subsequent motion.

The SWE given by (2.10) above is not the only way to model the shallow water flows. Stoker [66], for instance, models equivalently the one dimensional shallow water flows as two simultaneous equations

$$\eta_t + [u(D + \eta)]_x = 0$$
$$u_t + uu_x = -g\eta_x$$

(2.11)

The first equation in (2.11) is the continuity equation, while the second equation describes the motion that is the evolution of the velocity $u$ corresponding to the $x$-direction. In these alternative SWE, $\eta$ is the surface elevation which is positive if it is above the equilibrium level, whereas $D$ is the depth function which is positive if it is below the equilibrium level. The total depth of the fluid is $D + \eta$. This model is illustrated in Figure 2.4.

![Figure 2.4: Alternative illustration of shallow water flows in one-dimension](image)

By observing Figures 2.1 and 2.4, $D = -z$ and $\eta = h + z$ are obtained. By
use of these relations, it can be easily shown that (2.10) is equivalent to (2.11) as given in the following theorem.

**Theorem 2.1.** The systems of equations,

\[ h_t + (hu)_x = 0 \]  
\[ (hu)_t + (hu^2 + \frac{1}{2}gh^2)_x = -gz_x \]  

and

\[ \eta_t + [u(D + \eta)]_x = 0 \]  
\[ u_t + uu_x = -g\eta_x \]  

are equivalent.

**Proof.** Substituting \( D = -z \) and \( \eta = h + z \) into \( \eta_t + [u(D + \eta)]_x = 0 \) yield \((h + z)_t + (uh)_x = 0\). Since \( z_t = 0 \), the continuity equation,

\[ h_t + (uh)_x = 0, \]

of the SWE is obtained. Now, substituting \( \eta = h + z \) into \( u_t + uu_x = -g\eta_x \) and expanding the result leads to

\[ hu_t + huu_x + hh_x = -gz_x, \]

which can also be written as

\[ [h_tu] + hu_t + [(hu)_xu] + huu_x + gh_x = -gz_x \]  

(2.14)

by applying the continuity equation. Furthermore, equation (2.14) can be simplified to

\[ (hu)_t + \left( hu^2 + \frac{1}{2}gh^2 \right)_x = -gz_x. \]

Hence, the system of equations (2.13) implies (2.12).

By reversing the order of steps above, the system of equations (2.12) implies (2.13). Therefore, both systems of equations are equivalent.

\[ \square \]

### 2.3 Hyperbolic Problems

The SWE considered in this thesis are a first order hyperbolic system of partial differential equations. Hyperbolicity of a linear system is characterised as follows.
2.3. HYPERBOLIC PROBLEMS

**Definition 2.2.** A linear system of the form

\[ q_t + Aq_x = 0, \]

where \( A \) is a constant coefficient matrix, is called *hyperbolic* if the \( m \times m \) matrix \( A \) is diagonalizable with real eigenvalues.

There are several special classes of hyperbolic systems according to the properties of matrices \( A \). If \( A \) is a *symmetric* matrix, then \( A \) is diagonalizable with real eigenvalues and the system is called *symmetric hyperbolic*. If \( A \) has *distinct* real eigenvalues, then \( A \) is diagonalizable and the system is said to be *strictly hyperbolic*. If \( A \) has real eigenvalues but is not diagonalizable, then the system is called *weakly hyperbolic*.

There are also other forms of hyperbolic systems. A variable-coefficient linear system of the form

\[ q_t + A(x)q_x = 0 \]

is hyperbolic at any point \( x \) where the coefficient matrix \( A(x) \) satisfies the hyperbolicity condition stated in Definition 2.2. If source term \( S(x) \) appears in the system written as

\[ q_t + A(x)q_x = S(x) \]

the hyperbolicity still depends on the coefficient matrix, which means that it is hyperbolic at any point where \( A(x) \) is diagonalizable with real eigenvalues.

A *quasilinear* system

\[ q_t + A(q, x, t)q_x = S(x) \]

is said to be hyperbolic at a point \((q, x, t)\) if the coefficient matrix \( A(q, x, t) \) satisfies the hyperbolicity condition as in Definition 2.2. Hence, the nonlinear conservation law

\[ q_t(x, t) + [f(q(x, t))]_x = S(x), \quad (2.15) \]

which can also be written in quasilinear form

\[ q_t + f'(q)q_x = S(x) \quad (2.16) \]

is hyperbolic if the Jacobian matrix \( f'(q) \) satisfies the hyperbolicity condition for each relevant value of \( q \).
Example 2.3. A simple example of hyperbolic is the scalar, linear, constant coefficient advection equation

\[ q_t + \bar{u}q_x = 0 \]  \hspace{1cm} (2.17)

This equation models the advection of a tracer \( q \) along with the fluid, where the fluid velocity \( \bar{u} \) is constant. A tracer here means a substance that is present in very small concentrations within the fluid, so that the magnitude of the concentration has essentially no effect on the fluid dynamics.

It is easy to verify that the general solution of (2.17) is any smooth function of the form

\[ q(x, t) = \tilde{q}(x - \bar{u}t). \]

Along any ray in space-time for which \( x - \bar{u}t = \text{cst} \) (cst is an abbreviation for constant), the quantity \( q(x, t) \) is constant. For example, along the ray \( X(t) = x_0 + \bar{u}t \), the value of \( q(X(t), t) \) is equal to \( \tilde{q}(x_0) \). The rays \( X(t) \) are called the characteristic of the equation. Given the initial data \( q(x, t_0) = \hat{q}(x) \), then the particular solution has the form

\[ q(x, t) = \hat{q}(x - \bar{u}(t - t_0)) \]

for \( t \geq t_0 \).

If the space is bounded, \( a < x < b \), then the density of tracer entering the space at the inflow end must also be specified. For example, given that \( \bar{u} > 0 \), then two conditions are needed to find the form of the solution. The first condition is a boundary condition at \( x = a \), say

\[ q(a, t) = g_0(t) \quad \text{for} \quad t \geq t_0, \]

and the second condition is the initial condition

\[ q(x, t_0) = \hat{q}(x) \quad \text{for} \quad a < x < b. \]

Considering those conditions, the particular solution is then

\[ q(x, t) = \begin{cases} 
  g_0(t - (x - a)/\bar{u}) & \text{if} \quad a < x < a + \bar{u}(t - t_0) \\
  \hat{q}(x - \bar{u}(t - t_0)) & \text{if} \quad a + \bar{u}(t - t_0) < x < b
\end{cases} \]

Note that when \( u \) is not constant and the advection equation is of the form \( q_t + [u(x)q]_x = 0 \), the characteristic curves are not straight lines and the solution \( q \) is no longer constant along the curves.
Example 2.4. Another example of hyperbolic system, which is also the focus of this thesis, is the SWE (2.10). This system can be expressed by (2.15), where

\[
q = \begin{bmatrix} h \\ uh \end{bmatrix} = \begin{bmatrix} q_1 \\ q_2 \end{bmatrix},
\]

(2.18)

\[
f = \begin{bmatrix} uh \\ uh^2 + gh^2/2 \end{bmatrix} = \begin{bmatrix} q_2 \\ (q_2)^2/q_1 + gq_1^2/2 \end{bmatrix},
\]

(2.19)

and

\[
S = \begin{bmatrix} 0 \\ -ghz_x \end{bmatrix} = \begin{bmatrix} 0 \\ -gq_1z_x \end{bmatrix}.
\]

(2.20)

From the explanation above, then the SWE can be written into a quasilinear form (2.16), where the Jacobian matrix \(f'(q)\) of \(f\) is

\[
f'(q) = \begin{bmatrix} 0 & 1 \\ -u^2 + gh & 2u \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial q_1} & \frac{\partial f_1}{\partial q_2} \\ \frac{\partial f_2}{\partial q_1} & \frac{\partial f_2}{\partial q_2} \end{bmatrix}.
\]

(2.21)

The Jacobian matrix \(f'(q)\) has eigenvalues

\[
\lambda_1 = u - \sqrt{gh} \quad \text{and} \quad \lambda_2 = u + \sqrt{gh}
\]

with the corresponding eigenvectors

\[
r_1 = \begin{bmatrix} 1 \\ u - \sqrt{gh} \end{bmatrix} \quad \text{and} \quad r_2 = \begin{bmatrix} 1 \\ u + \sqrt{gh} \end{bmatrix},
\]

where \(g\) is positive. It is clear that the two eigenvalues are distinct and real if \(h\) is positive. This proves that the SWE (2.10) is strictly hyperbolic whenever \(h\) is positive.

2.3.1 Linear Hyperbolic Equations

A linear hyperbolic system of the form

\[
q_t + Aq_x = 0,
\]

where \(A\) is an \(m \times m\) constant matrix, can be reduced to a set of \(m\) decoupled advection equations

\[
\omega_t + \Lambda \omega_x = 0,
\]

(2.22)
where \( \omega = R^{-1}q \), \( A = R \Lambda R^{-1} \), and \( R \) is the matrix of the right eigenvectors. This is because \( A \) is diagonalisable with real eigenvalues.

Suppose that the initial data are given by

\[
q(x, 0) = \hat{q}(x) \quad \text{for} \quad -\infty < x < \infty,
\]

then \( \hat{\omega}(x) = R^{-1}\hat{q}(x) \) satisfies (2.22). The \( p \)th equation of (2.22) is the advection equation

\[
\omega_{p,t} + \lambda_p \omega_{p,x} = 0,
\]

which has the solution \( \omega_p(x, t) = \omega_p(x - \lambda_p t, 0) = \hat{\omega}_p(x - \lambda_p t) \). All \( \omega_p(x, t) \), \( p = 1, \ldots, m \) form \( \omega(x, t) \). Therefore, the solution to the original problem is

\[
q(x, t) = R\omega(x, t) = \sum_{p=1}^{m} \omega_p(x, t)r_p, \tag{2.23}
\]

where \( r_1, \ldots, r_m \) are the column vectors of the matrix \( R \), as well as the right eigenvectors of the matrix \( A \).

The solution (2.23) can also be written as

\[
q(x, t) = \sum_{p=1}^{m} [l_p \hat{q}(x - \lambda_p t)]r_p, \tag{2.24}
\]

where \( l_1, l_2, \ldots, l_m \) are the row vectors of \( R^{-1} \) or the left eigenvectors of the matrix \( A \). This is because \( \omega_p(x, t) = l_p q(x, t) \).

The formula (2.24) can be used even if the initial data \( \hat{q}(x) \) are not smooth, or are even discontinuous, at some points. If the data have a singularity, that is a discontinuity in some derivative at some point \( x_0 \), then one or more of the characteristic variables \( \omega_p(x, 0) \) will also have a singularity at this point. The singularities in the initial data will propagate along the characteristics and lead to singularities in the solution \( q(x, t) \) at some or all of the points \( x_0 + \lambda_p t \). On the other hand, if the initial data are smooth in a neighborhood of all the points \( \bar{x} - \lambda_p \bar{t} \), then the solution \( q(x, t) \) is smooth in a neighborhood of the point \( (\bar{x}, \bar{t}) \). Overall, for a linear system the singularities can only propagate along characteristics.

According to (2.23), the solution \( q(x, t) \) can be viewed as the superposition of \( m \) waves, each of which is advected independently with no change in shape. This means that the \( p \)th wave has shape \( \hat{\omega}_p(x)r_p \) propagating with speed \( \lambda_p \). If \( \omega_p(x, 0) \)}
is constant in $x$ for all but one value of $p$, then the solution to the problem is called *simple waves*. For example, if $\tilde{\omega}_p(x) = \tilde{\omega}_p$ for $p \neq i$, then the initial data simply propagate with speed $\lambda_i$, that is

$$q(x, t) = \tilde{\omega}_i(x - \lambda_i t)r_i + \sum_{p \neq i} \tilde{\omega}_p r_p = \hat{q}(x - \lambda_i t).$$

### 2.3.2 Domain of Dependence and Range of Influence

For some fixed point $(X, T)$ in space-time, the solution $q(X, T)$ depends only on the data $\hat{q}$ at $m$ particular points $X - \lambda_p T$, $p = 1, \ldots, m$. The set of points

$$D(X, T) = \{X - \lambda_p T : p = 1, \ldots, m\},$$

is said to be the *domain of dependence* of point $(X, T)$. The value of the initial data at other points has no influence on the value of $q(X, T)$. For hyperbolic equations, the domain of dependence is always a bounded set. This is due to the eigenvalues of the matrix coefficient being real, meaning that information propagates at finite speed. The domain of dependence for a system of two equations is illustrated in Figure 2.5 (a).

![Figure 2.5](image)

Figure 2.5: A typical hyperbolic system of two equations with $\lambda_1 < 0 < \lambda_2$, (a) shows the domain of dependence of the point $(X, T)$, and (b) shows the range of influence of the point $x_0$.

On the other hand, the data $\hat{q}(x_0)$ will affect the solution along the characteristic rays $x_0 + \lambda_p t$. The set of points

$$I(x_0, t) = \{X + \lambda_p t : p = 1, \ldots, m\}$$

is called the *range of influence* of the point $x_0$. The range of influence is illustrated in Figure 2.5 (b).
2.3.3 Riemann Problem

The SWE as in (2.10) can be written in vector form

\[ q_t + [f(q)]_x = S \]  

(2.25)

where \( q = q(x, t) \) is the quantity vector that is the vector of conserved variables, \( f \) represents the flux vector in the \( x \) direction and \( S \) denotes the source vector.

The vectors \( q, f, \) and \( S \) are given by

\[ q = \begin{bmatrix} h \\ hu \end{bmatrix}, \quad f = \begin{bmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \end{bmatrix}, \quad \text{and} \quad S = \begin{bmatrix} 0 \\ -ghz_x \end{bmatrix} \]

There is a special type of hyperbolic problem which is called the Riemann problem. This problem is a hyperbolic system together with piecewise constant initial data containing a single jump discontinuity at some point. The initial data is expressed as

\[ q(x, t_0) = \begin{cases} q_l & \text{if } x < x_0 \\ q_r & \text{if } x > x_0 \end{cases} \]

To simplify the notation for the Riemann problem, \( q_l \) and \( q_r \) can be decomposed as

\[ q_l = \sum_{p=1}^{m} \omega_{p,l} r_p \quad \text{and} \quad q_r = \sum_{p=1}^{m} \omega_{p,r} r_p \]

Then Riemann data for the \( p \)th advection equation are

\[ \hat{\omega}_p(x) = \begin{cases} \omega_{p,l} & \text{if } x < x_0 \\ \omega_{p,r} & \text{if } x > x_0 \end{cases} \]

Since the discontinuity propagates with speed \( \lambda_p \), it is obtained that

\[ \omega_p(x, t) = \begin{cases} \omega_{p,l} & \text{if } x - \lambda_p t < x_0 \\ \omega_{p,r} & \text{if } x - \lambda_p t > x_0 \end{cases} \]

By denoting the maximum value of \( p \) for which \( x - \lambda_p t > x_0 \) as \( P(x, t) \), the solution \( q(x, t) \) can be expressed as

\[ q(x, t) = \sum_{p=1}^{P(x, t)} \omega_{p,r} r_p + \sum_{p=P(x, t)+1}^{m} \omega_{p,l} r_p \]
or can be written as

\[
q(x, t) = \sum_{p: \lambda_p < (x-x_0)/t} \omega_{p,r} r_p + \sum_{p: \lambda_p > (x-x_0)/t} \omega_{p,l} r_p.
\] (2.26)

Across the \( p \)th characteristic the solution jumps with the jump in \( q \) given by

\[
(\omega_{p,r} - \omega_{p,l}) r_p = \alpha_p r_p,
\]

where \( \omega_{p,r} - \omega_{p,l} = \alpha_p \) is a constant. Since the jump is a multiplication of scalar and \( r_p \), this means that the jump in \( q \) is an eigenvector of the matrix \( A \). This condition is called the Rankine-Hugoniot jump condition, derived from the integral form of the conservation law, and seen to hold across any propagating discontinuity.

Solving the Riemann problem for a linear system consists of taking the initial data \((q_l, q_r)\) and decomposing the jump \( q_r - q_l \) into eigenvectors of \( A \)

\[
q_r - q_l = \alpha_1 r_1 + \cdots + \alpha_m r_m.
\]

This can be written as a linear system of equations \( R\alpha = q_r - q_l \). Therefore, \( \alpha = R^{-1}(q_r - q_l) \). The vector \( \alpha \) has components \( \alpha_p = l_p(q_r - q_l) \), where \( l_p \)'s are the left eigenvectors of \( A \), so \( \alpha_p = \omega_{p,r} - \omega_{p,l} \).

Since \( \alpha_p r_p \) is the jump in \( q \) across the \( p \)th wave in the solution to the Riemann problem, this wave can be denoted by

\[
\mathcal{W} = \alpha_p r_p.
\]

By use of (2.26), the solution \( q(x, t) \) can be written in terms of the waves in two different forms, either

\[
q(x, t) = q_l + \sum_{p: \lambda_p < (x-x_0)/t} \mathcal{W}_p,
\]

or

\[
q(x, t) = q_r - \sum_{p: \lambda_p \geq (x-x_0)/t} \mathcal{W}_p.
\]

As illustrated by Figure 2.6 in the \( x,t \)-plane, for a general Riemann problem for a system of two equations with arbitrary \( q_l \) and \( q_r \), the solution consists of two discontinuities travelling with speed \( \lambda_1 \) and \( \lambda_2 \), with a constant state inbetween denoted by \( q_m \). Here,

\[
q_m = \omega_{1,r} r_1 + \omega_{2,r} r_2,
\]

and since \( q_l = \omega_{1,l} r_1 + \omega_{2,l} r_2 \) and \( q_r = \omega_{1,r} r_1 + \omega_{2,r} r_2 \), it is obtained that

\[
q_m - q_l = (\omega_{1,r} - \omega_{1,l}) r_1 \quad \text{and} \quad q_r - q_m = (\omega_{2,r} - \omega_{2,l}) r_2.
\]
2.3.4 Initial-Boundary-Value Problem

A hyperbolic system on a bounded interval $a \leq x \leq b$ is an Initial-Boundary-Value Problem (IBVP) since it is a time-dependent problem for which the initial data and the boundary data are needed. A system of $m$ equations needs a total of $m$ boundary conditions, where some of them are prescribed at the left boundary $x = a$ and some at the right boundary $x = b$. The number of the needed data at each boundary depends on the number of eigenvalues of $A$ which are positive and negative.

For a simple advection equation $q_t + \bar{u} q_x = 0$, a boundary condition at $x = a$ is needed if $\bar{u} > 0$ and at $x = b$ if $\bar{u} < 0$. It is clear that a general system which is diagonalisable can be transformed into a decoupled set of advection equations

$$\omega_p, t + \lambda_p \omega_p, x = 0.$$

For this system if all eigenvalues are assumed to be nonzero, the boundary data on $\omega_p(x, t)$ at $x = a$ needs to be specified if $\lambda_p > 0$. Similarly, the boundary data at $x = b$ needs to be specified if $\lambda_p < 0$.

Consider the system of $m$ equations which has $n$ negative eigenvalues which are $\lambda_1, \lambda_2, \ldots, \lambda_n$ and $m - n$ positive eigenvalues which are $\lambda_{n+1}, \lambda_{n+2}, \ldots, \lambda_m$, where

$$\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n < 0 < \lambda_{n+1}, \lambda_{n+2}, \ldots, \lambda_m.$$

This system needs $m - n$ boundary conditions at $x = a$ and $n$ boundary conditions at $x = b$.
2.4. PROPERTIES OF SWE

at $x = b$ to be specified. To impose this type of boundary data, partition the vector $\omega$ as

$$
\omega = \begin{bmatrix}
\omega_I \\
\omega_{II}
\end{bmatrix},
$$

where $\omega_I \in \mathbb{R}^n$ and $\omega_{II} \in \mathbb{R}^{m-n}$. Then at the left boundary, the components of $\omega_{II}$ must be specified while $\omega_I$ are outflow variables. This can be done by writing $\omega_{II}$ in terms of $\omega_I$, such as a linear boundary condition of the form

$$
\omega_{II}(a,t) = B_1 \omega_I(a,t) + g_1(t),
$$

where $B_1 \in \mathbb{R}^{(m-n)xn}$ and $g_1 \in \mathbb{R}^{m-n}$. If $B_1 = 0$, then the inflow variables are given without effect of the outflow. However, there is often a reflection of outgoing waves through a physical boundary, and this requires a nonzero matrix $B_1$.

2.4 Properties of the Shallow Water Equations

To solve the SWE, understanding its properties will be useful. This section explores several of the SWE properties and assumes that the water bed topography is flat, unless stated explicitly. Since the topography is flat, the SWE is considered homogeneous.

By using a gas dynamics analogy, new quantities

$$
\bar{\rho} = \rho h \quad \text{and} \quad \bar{p} = \int_z^w p \, dy
$$

can be introduced where $\bar{\rho}$ and $\bar{p}$ are the density-like and pressure-like respectively. In view of the hydrostatic pressure law, the last equation can be expressed as

$$
\bar{p} = \frac{g \bar{\rho}^2}{2\rho}.
$$

Here, the depth of water is in place of density in a gas. By recalling the sound speed or the propagation speed which is given by $c = \sqrt{\bar{p}'(\bar{\rho})}$ as in acoustics, this quantity is then given by

$$
c = \sqrt{\frac{g \bar{\rho}}{\rho}} = \sqrt{gh}.
$$

The sound speed terms can be seen, for example, in Example 2.4.
CHAPTER 2. ONE-DIMENSIONAL SWE

By applying the method of characteristic there are two sets of characteristic curves, \( C_1 \) and \( C_2 \), which are the solution curves of the ordinary differential equations

\[
\begin{align*}
C_1 : \frac{dx}{dt} &= u + c, \\
C_2 : \frac{dx}{dt} &= u - c .
\end{align*}
\] (2.27)

The functions \( u + c \) and \( u - c \) are the eigenvalues of the coefficient matrix as shown in Example 2.4. The relations

\[
\begin{align*}
u + 2c &= k_1 = \text{cst along a curve } C_1 \text{ and,} \\
u - 2c &= k_2 = \text{cst along a curve } C_2.
\end{align*}
\] (2.28)

hold, and are known as Riemann invariants. In addition, the constants \( k_1 \) and \( k_2 \) are different on different curves and the two families of characteristics given by (2.27) are distinct since \( c \neq 0 \) for \( h \neq 0 \). Furthermore, the system of equations (2.27) together with (2.28) are equivalent to the homogeneous SWE.

2.4.1 Notion of a Simple Wave

Suppose that the initial water state is uniform, that is at time \( t = 0 \) the particle velocity and the sound speed are \( u = u_0 = \text{cst} \) and \( c = c_0 = \sqrt{gh} = \text{cst} \) respectively. Now suppose that a disturbance is initiated at the origin \( x = 0 \) so that either \( u \) or \( c \) changes with the time. This means a disturbance at one point in the water propagates into water of constant depth and uniform velocity. Under these conditions, it can be proved that one of the two families of characteristics given by ordinary differential equations (2.27) consists entirely of straight lines along each of which \( u \) and \( c \) are constant, and then the corresponding motion is called simple wave. To make it clear, this statement is written in the following theorem.

Theorem 2.5. One of the two families of characteristics given by ordinary differential equations

\[
\begin{align*}
\frac{dx}{dt} &= u + c, \quad \text{and} \\
\frac{dx}{dt} &= u - c
\end{align*}
\] (2.29)

consists entirely of straight lines along each of which \( u \) and \( c \) are constant.
2.4. PROPERTIES OF SWE

Proof. First, observe that if the values of \( u \) and \( c \) on any characteristic curve, say a solution curve \( C_1^0 \) of the first of (2.29), are constant, then \( C_1^0 \) is a straight line along each of which \( u \) and \( c \) are constant. This is at least in a region of the \( x,t \)-plane where \( u(x,t) \) and \( c(x,t) \) are without singularities and which is covered by the two distinct families of characteristics. It is clear that the curve \( C_1^0 \) is a straight line if \( u \) and \( c \) are constant along it, because the slope of the curve is constant in this case according to (2.29). Next, let \( C_1 \) be another characteristic near \( C_1^0 \). Then consider any two points \( A_0 \) and \( B_0 \) on \( C_1^0 \) together with the characteristics of the family \( C_2 \) through \( A_0 \) and \( B_0 \) and suppose that \( C_2 \) characteristics intersect \( C_1 \) at points \( A \) and \( B \) as shown in Figure 2.7. To prove the statement is then enough by showing that \( u(A) = u(B) \) and \( c(A) = c(B) \) since then \( u \) and \( c \) would be constant on \( C_1 \) (this is because of the fact that \( A \) and \( B \) are arbitrary on \( C_1 \)) and therefore the slope of the curve \( C_1 \) would be constant. Since \( u(A_0) = u(B_0) \) and \( c(A_0) = c(B_0) \), by using the second relation of (2.28) it is obtained that

\[
\begin{align*}
\left\{ \begin{array}{l}
u_A - 2c_A &= u_{A_0} - 2c_{A_0}, \\
u_B - 2c_B &= u_{B_0} - 2c_{B_0} = u_{A_0} - 2c_{A_0}
\end{array} \right. \\
\text{(2.30)}
\end{align*}
\]

which are equivalent to

\[
u_A - 2c_A = u_B - 2c_B \quad \text{(2.31)}
\]

Making use of the first relation of (2.28) for \( C_1 \) leads to

\[
u_A + 2c_A = u_B + 2c_B. \quad \text{(2.32)}
\]
It is clear that (2.31) and (2.32) can only be satisfied if \( u_A = u_B \) and \( c_A = c_B \). This completes the proof.

**2.4.2 Entropy Inequality**

The SWE model (2.10) above has a hyperbolic type and admits an entropy inequality related to the physical energy. The derivation of the entropy inequality can be found in [6], and is given by

\[
\tilde{\eta}_t(q, z) + \tilde{G}_x(q, z) \leq 0
\]

where

\[
\tilde{\eta}(q, z) = \eta(q) + ghz, \quad \tilde{G}(q, z) = G(q) + ghzu,\]

and

\[
\eta(q) = h\frac{u^2}{2} + \frac{g}{2}h^2, \quad G(q) = (h\frac{u^2}{2} + gh^2)u.
\]

**2.5 Shock Conditions**

By expressing the SWE into quasilinear form and assuming the solution is smooth, it can be shown that there are two families of characteristics producing a curvilinear coordinate system over \( x, t \)-plane. However, that is often not the case because the solution may not be smooth in some problems.

In theory, when two or more characteristics of the same family intersect, a discontinuity known as shock will occur. In this case, to investigate the solutions, the integral form of the conservation law must be applied. Given \( s \) is the shock propagating speed, it can be shown that across any shock the Rankine-Hugoniot condition

\[
s(q_r - q_l) = f(q_r) - f(q_l)
\]

must be satisfied. However, the Rankine-Hugoniot condition does not guarantee a unique solution. To determine whether a weak solution is indeed the physically correct solution, applying the Lax entropy condition is needed. This condition is satisfied if the characteristics converge such that

\[
\lambda_1(q_l) > s > \lambda_1(q_r) \quad \text{and} \quad \lambda_2(q_l) > s > \lambda_2(q_r)
\]
2.5. SHOCK CONDITIONS

for the 1-shock and the 2-shock respectively. Here $\lambda_1 = u - \sqrt{gh}$ and $\lambda_2 = u + \sqrt{gh}$ are the eigenvalues of the Jacobian matrix $f'(q)$ which are often called the characteristic speeds.

In practice, shock wave results from discontinuity of the quantities such as depth, velocity, and pressure. Suppose it is given that there is discontinuity (on the water depth, velocity and pressure) at point $\xi(t)$ between $x = a_0(t)$ and $x = a_1(t)$ where $a_1 > a_0$ as illustrated in Figure 2.8.

![Figure 2.8: Discontinuity conditions](image)

Applying the laws of conservation of mass and of momentum to that column of water lead to relations

$$\frac{d}{dt} \int_{a_0(t)}^{a_1(t)} \rho h \, dx = 0 \quad (2.33)$$

and

$$\frac{d}{dt} \int_{a_0(t)}^{a_1(t)} \rho h u \, dx = \int_{z_0}^{z_0} p_0 \, dy - \int_{z_0}^{z_1} p_1 \, dy$$

$$= \frac{1}{2} \rho gh^2 - \frac{1}{2} \rho gh_1^2 \quad (2.34)$$

where the formula for the water pressure is $p = \rho gh$. The first relation states that the water mass in the column remains constant, while the second relation states that the change in momentum of the water column is equal to the difference of the resultant forces over the end sections of the column.
The integrals in these relations have the form

\[ I = \int_{a_0(t)}^{a_1(t)} \psi(x, t) \, dx \]

where \( \psi(x, t) \) is discontinuous at \( x = \xi(t) \). Applying Leibniz’ rule to differentiate this integral yields the relation

\[
\frac{dI}{dt} = \frac{d}{dt} \int_{a_0(t)}^{\xi(t)} \psi \, dx + \frac{d}{dt} \int_{\xi(t)}^{a_1(t)} \psi \, dx
\]

\[
= \int_{a_0(t)}^{a_1(t)} \frac{\partial \psi}{\partial t} \, dx + \psi(\xi_, t) \dot{\xi}(t) - \psi(a_0(t), t)u_0
\]

\[
+ \psi(a_1(t), t)u_1 - \psi(\xi_+, t)\dot{\xi}t
\]

Here, \( u_0 = \dot{a}_0(t) = \frac{1}{\pi}a_0(t) \) and \( u_1 = \dot{a}_1(t) = \frac{1}{\pi}a_1(t) \) are the velocities at the ends of the column, \( \dot{\xi} \) is the velocity of the discontinuity, and \( \psi(\xi, t) \) and \( \psi(\xi_+, t) \) are the limit value of \( \psi \) to the left and to the right of \( x = \xi \). By taking the limit in which the length of the column tends to zero in such away that the discontinuity remains inside the column, the integral on the right-hand side of (2.35) tends to zero, and it is obtained that

\[
\lim_{l \to 0} \frac{dI}{dt} = \psi_1v_1 - \psi_0v_0.
\]

Here, \( l = a_1 - a_0 \) is the length of the column, \( v_1 = u_1 - \dot{\xi} \) and \( v_0 = u_0 - \dot{\xi} \) are the flow velocities relative to the moving discontinuity, and \( \psi_1 \) and \( \psi_0 \) refer to the limit values of \( \psi \) to the right and to the left of the discontinuity respectively.

According to the presentation on the previous paragraph, using (2.36) for the limit cases which arise from (2.33) and (2.34), it is obtained that

\[
\rho h_1v_1 - \rho h_0v_0 = 0
\]

(2.37)

and

\[
\rho h_1u_1v_1 - \rho h_0u_0v_0 = \frac{1}{2} \rho gh_0^2 - \frac{1}{2} \rho gh_1^2.
\]

(2.38)

Equations (2.37) and (2.38) can also be written as

\[
\bar{\rho}_1v_1 = \bar{\rho}_0v_0
\]

(2.39)

and

\[
\bar{\rho}_1u_1v_1 - \bar{\rho}_0u_0v_0 = \bar{p}_0 - \bar{p}_1
\]

(2.40)
2.6. CONCLUDING REMARKS

where $\bar{\rho} = \rho h$ and $\bar{p} = \frac{1}{2} \rho gh^2 = \frac{1}{2} \bar{\rho}\bar{p}^2$. A discontinuity satisfying (2.39) and (2.40) is referred to as a shock wave or simply as a shock or as a bore, or if it is stationary as a hydraulic jump.

Another way of expressing the shock conditions (2.39) and (2.40) is

$$\begin{cases}
\bar{\rho}_1 v_1! = \bar{\rho}_0 v_0 = m, \\
m(v_1 - v_0) = \bar{p}_0 - \bar{p}_1,
\end{cases}$$

(2.41)

where $m$ represents the mass flux across the shock front.

Consider a special case where $u_0 = 0$ which means the water is at rest on one side of the shock. Since $mv_1 = \bar{\rho}_0 v_0 v_1$ and $mv_0 = \bar{\rho}_1 v_1 v_0$, the second of the shock conditions (2.41) can be expressed as

$$v_1 v_0 = \frac{\bar{p}_0 - \bar{p}_1}{\bar{\rho}_0 - \bar{\rho}_1},$$

(2.42)

Since $u_0 = 0$, it is clear that $v_0 = -\dot{\xi}$ and $v_1 = u_1 - \dot{\xi}$ so that (2.42) takes the form

$$-\dot{\xi}(u_1 - \dot{\xi}) = \frac{g}{2\rho}(\bar{\rho}_0 + \bar{\rho}_1)$$

(2.43)

by using the relation $\bar{p} = \frac{1}{2} \bar{\rho}\bar{p}^2$ and $\bar{p} = \rho_0 - \rho_1$. Similarly, since $u_0 = 0$, the first shock condition now takes the form

$$\bar{\rho}_1(u_1 - \dot{\xi}) = -\bar{\rho}_0 \dot{\xi},$$

(2.44)

Furthermore, if $u_1$ is eliminated, the second shock condition (2.43) can be written as

$$\dot{\xi}^2 = \frac{g\bar{\rho}_1}{2\rho} \left( 1 + \frac{\bar{\rho}_1}{\bar{\rho}_0} \right);$$

(2.45)

or if $\bar{\rho}_0$ is eliminated, it can also be written as

$$-\dot{\xi}(u_1 - \dot{\xi}) = \frac{g\bar{\rho}_1}{2\rho} \left( 1 - \frac{u_1 - \dot{\xi}}{\dot{\xi}} \right)$$

(2.46)

Therefore, (2.44) together with either (2.43), (2.45), or (2.46) are alternative ways of expressing the shock conditions when $u_0 = 0$.

2.6 Concluding Remarks

This chapter has presented three main topics, namely: the derivation of the SWE, hyperbolicity property, and shock conditions. The derivation of the SWE
is based on Eulerian description for the conservation of mass, and on Langrangian for the conservation of momentum. The hyperbolicity property being used is the property for first order partial differential equations. In addition, the shock conditions are derived due to discontinuity on the depth, velocity and pressure.
Chapter 3

Analytical Solutions to the Shallow Water Equations

This chapter considers three specific types of problems relating to the SWE and presents their analytical solutions. These three problems are: dam-break problems, oscillations in a parabolic canal, and steady flow over a parabolic obstruction.

3.1 Dam-Break Problems

This section presents the determination of the flow which results from a sudden destruction of a dam. The considered problem is the SWE (2.10) with a flat-bottom topography (flat water bed), that is $z_x = 0$, such that

$$
\begin{align*}
    h_t + (hu)_x &= 0, \\
    (hu)_t + (hu^2 + \frac{1}{2}gh^2)_x &= 0
\end{align*}
$$

(3.1)

The construction of analytical expressions of this problem can be found in [66]. The solutions to dam-break problems have been presented by several authors (see [27, 75]).

The classification of dam-break problems can be based on the initial position and the initial depth of the downstream side. If the downstream of a dam is on the right side then the problem is called a rightward dam-break because the water will move in a rightward direction after the dam is broken, otherwise it is a leftward dam-break. If the initial depth of the downstream side is zero then it
is known as dry-dam, otherwise it is known as a finite water depth problem.

Here, only the rightward dam-break problem is presented because the leftward dam-break problem can be interpreted in the same way by changing the direction of the water movement. It is assumed that there is water of constant depth on the downstream as well as the upstream side of the dam, and the water is assumed to be at rest on both sides of the dam initially as illustrated in Figure 3.1. Therefore, the rightward dam-break problem is set by a system of equations (3.1) with the initial conditions

\[ u(x, 0) = 0 \quad \text{and} \quad h(x, 0) = \begin{cases} h_1 & \text{if } x < x_0 \\ h_0 & \text{if } x > x_0 \end{cases} \]  
(3.2)

where both \( h_0 \) and \( h_1 \) are nonnegative and \( h_1 > h_0 \).

At time \( t = 0 \), the dam wall is immediately removed and the water upstream flows downstream at the subsequent time \( t \). Solving this problem means finding the profile of this water flow. The flow profile here consists of the depth and the momentum at any point \( x \) and at any time \( t \).

### 3.1.1 Dry-Dam Problem

Consider a dry-dam problem where the initial conditions are given by (3.2) with \( x_0 = 0 \) and \( h_0 = 0 \).

The typical solution of a dam-break problem on to a dry bed at some time \( t_1 > 0 \), where the dam is broken at \( t = 0 \), is illustrated by Figure 3.2. It is seen
that three different regions in the fluid at \( t = t_1 \) are considered. The zone (0) is the zone of dry-bed downstream which is terminated by the tip of a parabolic surface or rarefaction zone (R); the rarefaction zone (R) connects the dry-bed zone (0) with the zone of undisturbed water upstream (1).

The point on the surface which connects zone (1) and (R) at \( t = t_1 \) satisfies

\[
\frac{dx}{dt} = u(t_1) - c(t_1). 
\]

Since the water in zone (1) is still (that is \( u = 0 \) at any time) and \( c(t_1) = \sqrt{gh_1} \)

\[
\frac{dx}{dt} = -\sqrt{gh_1}. 
\]

Figure 3.2: Water profile of a dry-dam problem at some time \( t_1 > 0 \)
Therefore, the motion of that point satisfies $x = -t\sqrt{gh_1}$.

Consider the rarefaction fan in zone (R). Each of the characteristics satisfies

$$\frac{dx}{dt} = u(t_1) - c(t_1). \quad (3.3)$$

Along each of the characteristics in zone (R), according to (2.28) the Riemann invariant $u - 2c$ is constant. Therefore,

$$u(t_1) - 2c(t_1) = u_1 - 2c_1 \quad (3.4)$$

in which $u_1$ and $c_1$ are the known values of $u$ and $c$ in the zone (1). Hence, as one sees from (3.3) and (3.4) the slope of any straight characteristics can be given by either of the two forms:

$$\frac{dx}{dt} = \frac{3}{2} u(t_1) - c_1, \quad \text{or} \quad \frac{dx}{dt} = 2c_1 - 3c. \quad (3.5)$$

The slope in this zone cannot be negative, otherwise the water depth will be negative. Preserving the water depth to be nonnegative requires that $u(t_1) \leq 2c_1$.

In other words, the zone (R) is terminated by a point which moves on the line $x = 2t\sqrt{gh_1}$. In zone (R), the slopes of the characteristics are

$$\frac{dx}{dt} = \frac{x}{t}$$

According to (3.5), $dx/dt = \frac{3}{2} u(t_1) - c_1$, so that

$$\frac{x}{t} = \frac{3}{2} u - c_1,$$

which is equivalent to

$$u = \frac{2}{3} \left( \sqrt{gh_1} + \frac{x}{t} \right).$$

In addition, from the second relation of (3.5), it can be obtained the value of $c$ within zone (R) that

$$c = \frac{2}{3} c_1 - \frac{x}{3t}.$$

This last equation leads to $h = \frac{4}{9g} \left( \sqrt{gh_1} - \frac{x}{2t} \right)^2$.

Therefore, the free surface water profile and velocity at any time $t_1 > 0$ for the dry-dam problem with zero initial velocity are given by

$$h(x) = \begin{cases} 
  h_1 & \text{if } x \leq -t\sqrt{gh_1} \\
  h_R = \frac{4}{9g} \left( \sqrt{gh_1} - \frac{x}{2t} \right)^2 & \text{if } -t\sqrt{gh_1} < x \leq 2t\sqrt{gh_1} \\
  0 & \text{if } x \geq 2t\sqrt{gh_1} 
\end{cases} \quad (3.6)$$
3.1. DAM-BREAK PROBLEMS

and

\[ u(x) = \begin{cases} 
0 & \text{if } x \leq -t\sqrt{gh_1} \\
\frac{2}{3}(\sqrt{gh_1} + \frac{x}{t}) & \text{if } -t\sqrt{gh_1} < x \leq 2t\sqrt{gh_1} \\
0 & \text{if } x \geq 2t\sqrt{gh_1} 
\end{cases} \] (3.7)

The quantities at the position of the dam are interesting. At \( x = 0 \) it is obvious that the depth of the water at the site of the dam is always \( \frac{4}{5}h_1 \) which is constant. The velocity of the water at this point is also constant and has the value \( u = \frac{2}{3}\sqrt{gh_1} \). Thus, the volume rate of discharge of water at the original location of the dam is constant and has the value of \( hu = \frac{8h}{27}\sqrt{gh_1} \). It can also be observed that the water depth, velocity and discharge at every \( x \) will converge to these values as \( t \to \infty \).

Note that the solutions (3.6) and (3.7) are only for a dry-bed problem with horizontal bottom topography. For an ideal fluid flow, these solution may be extended using a technique called the method of superposition [12, 11] so that for a mild, constant slope, the solutions of the SWE are given by

\[ h(x) = \begin{cases} 
h_1 & \text{if } 1 > \frac{x}{t\sqrt{gh_1}} \\
h_R = \frac{b_0}{9}\left(2 + \frac{b_0\sqrt{\frac{g}{h_1}}}{2}\sqrt{\frac{a}{h_1}} - \frac{a}{t\sqrt{gh_1}}\right)^2 & \text{if } 1 \leq \frac{x}{t\sqrt{gh_1}} \leq 2 + \frac{b_0\sqrt{\frac{g}{h_1}}}{2}\sqrt{\frac{a}{h_1}} \\
0 & \text{if } \frac{x}{t\sqrt{gh_1}} > 2 + \frac{b_0\sqrt{\frac{g}{h_1}}}{2}\sqrt{\frac{a}{h_1}} 
\end{cases} \]

and

\[ u(x) = \begin{cases} 
0 & \text{if } 1 > \frac{x}{t\sqrt{gh_1}} \\
u_R = \frac{2\sqrt{\frac{g}{h_1}}}{3}\left(1 + b_0t\sqrt{\frac{a}{h_1}} + \frac{x}{t\sqrt{gh_1}}\right) & \text{if } 1 \leq \frac{x}{t\sqrt{gh_1}} \leq 2 + \frac{b_0\sqrt{\frac{g}{h_1}}}{2}\sqrt{\frac{a}{h_1}} \\
0 & \text{if } \frac{x}{t\sqrt{gh_1}} > 2 + \frac{b_0\sqrt{\frac{g}{h_1}}}{2}\sqrt{\frac{a}{h_1}} 
\end{cases} \]

where \( z_x = b_0 \) is the bed slope which is positive for downwardslope, and \( x \)-axes is the bed (note that \( x \)-axes is not the horizontal axes).

### 3.1.2 Finite Water Depth Problem

Consider the rightward finite water depth problem where \( x_0 = 0 \) and \( h_1 > h_0 > 0 \). The solutions can be constructed by analysing the characteristic curves, as has been done for the previous dry-dam problem, but for this case with additional shock conditions.
When the dam is immediately removed, as shown in Figure 3.3, the spatial domain is separated into four regions which are from left to right: first still region or region (1) where the depth is the same as initial water depth $h_1$, parabolic surface region or region (3) with depth $h_3$, constant region or region (2) with depth $h_2$, and second still region or region (0) where the depth is the same as initial water depth $h_0$. Region (2) is connected to the undisturbed water upstream (region (1)) by a rarefaction fan with a parabolic surface. This parabolic surface expands, while a shock moving to the right forms between the constant region (2) and the second still region (0).

The point at which region (1) and region (3) meet is determined by the characteristic

$$x = -c_1 t,$$
3.1. DAM-BREAK PROBLEMS

while the point separating region (3) and region (2) is determined by

\[ x = (u_2 - c_2)t, \]

and the position of the shock moving at speed \( \dot{\xi} \) is given by \( x = \dot{\xi}t \). Here, \( c_i = \sqrt{gh_i} \), where \( i = 1, 2, 3 \), are the wave propagation speed in water of depth \( h_i \).

In region (3), the characteristic curves are determined by

\[ \frac{dx}{dt} = \frac{x}{t} = u_3 - c_3. \]

Along each of the characteristic in this region, the Riemann invariants \( u + 2c \) is constant. Therefore, on the left extreme of the rarefaction fan it is true that \( u + 2c = 2c_1 \), and along the rarefaction fan it is also true that \( u + 2c = u_3 + 2c_3 \). Following this,

\[ 2c_1 = u_3 + 2c_3. \] (3.8)

Substituting \( c_3 = u_3 - x/t \) to (3.8) results in \( u_3 = \frac{2}{3}(c_1 + x/t) \) or

\[ u_3 = \frac{2}{3}(c_1 + \frac{x}{t}). \] (3.9)

Furthermore, substituting (3.9) to (3.8) leads to \( c_3 = \frac{1}{3}(2c_1 - x/t) \) or

\[ h_3 = \frac{4}{9g}(\sqrt{gh_1} - \frac{x}{2t})^2. \]

At the interface between region (2) and (0), it is convenient to write the shock conditions for the passage from state (0) to the state (2) in the form

\[ -\dot{\xi}(u_2 - \dot{\xi}) = \frac{1}{2}(c_0^2 + c_2^2), \] (3.10)

and

\[ c_2^2(u_2 - \dot{\xi}) = -c_0^2\dot{\xi}, \] (3.11)

which are the same as (2.43) and (2.44) with \( c_i^2 = gh_i \) in place of \( g\bar{\rho}_i/\rho \). To get the expression of \( u_2 \), the variable \( c_2^2 \) needs to be eliminated from (3.10) by use of (3.11), and then the resulting quadratic needs to be solved for \( u_2 \) and to be simplified so that it can be written

\[ u_2 = \dot{\xi} - \frac{c_0^2}{4\dot{\xi}}[1 + \sqrt{1 + 8(\dot{\xi}/c_0)^2}]. \] (3.12)
This yields
\[ u_2 = \dot{\xi} - \frac{gh_0}{4\xi} \left[ 1 + \sqrt{1 + \frac{8\dot{\xi}^2}{gh_0}} \right]. \] (3.13)

In the expression (3.12), the plus sign before the radical was taken in order that \( u_2 - \dot{\xi} \) and \(-\dot{\xi}\) have the same sign; and it can be observed that only positive values of \(\dot{\xi}\) and \(u_2\) need to be determined throughout the entire discussion since the side of (0) is the front side of the shock and the positive \(x\)-direction is taken to the right. Now, to get the expression of \(c_2\), the variable \(u_2\) needs to be eliminate from (3.10) by use of (3.11), and then the result can be expressed in the form
\[ \frac{c_2}{c_0} = \left[ \frac{1}{2} \sqrt{1 + 8(\dot{\xi}/c_0)^2} - \frac{1}{2} \right]^{1/2}. \] (3.14)

This leads to
\[ h_2 = \frac{h_0}{2} \left[ \sqrt{1 + \frac{8\dot{\xi}^2}{gh_0}} - 1 \right]. \]

Equations (3.13) and (3.14) provide expression of the velocity \(u_2\) and the wave speed \(c_2\) in the constant region (2) behind the shock as functions of the shock speed \(\dot{\xi}\) and the wave propagation speed \(c_0\) in the undisturbed region (0). Consider the rarefaction fan in region (3). Along each of the characteristics \(x/t = u - c\) in this region, the Riemann invariant \(u + 2c\) is constant. Therefore, on the left extreme and on the right extreme of the rarefaction fan
\[ u + 2c = 2c_1 \quad \text{and} \quad u + 2c = u_2 + 2c_2, \]
respectively. According to these two relations, it is clear that \(2c_1 = u_2 + 2c_2\). Since the form of \(u_2\) and \(c_2\) have been determined in (3.13) and (3.14), it is obtained that
\[ c_1 = \frac{\dot{\xi}}{2} - \frac{c_0^2}{8\xi} \left[ 1 + \sqrt{1 + \frac{8\dot{\xi}^2}{c_0}} \right] + \left[ \frac{c_0^2}{2} \sqrt{1 + \frac{8\dot{\xi}^2}{c_0}} - \frac{c_0^2}{2} \right], \]
which can be rewritten as
\[ \dot{\xi} = 2c_1 + \frac{c_0^2}{4\xi} \left[ 1 + \sqrt{1 + \frac{8\dot{\xi}^2}{c_0}} \right] - \left[ 2c_0^2 \sqrt{1 + \frac{8\dot{\xi}^2}{c_0}} - 2c_0^2 \right]^{1/2}. \] (3.15)

In summary, the solution to the finite water depth problem can be written in
3.2. OSCILLATIONS IN A PARABOLIC CANAL

This section presents the analytical solutions to time-dependent motions in parabolic canals without friction and without Coriolis force. These solutions provide a valuable test for numerical models of shallow water flows. The solutions presented here are based on the work of Thacker [69]. An interesting feature of the solutions is that no shock forms as the water flows up the sloping sides of the canal.

The motion of water in shallow canals is governed by the SWE (2.10) or (2.11). For investigating the properties of the motion, the equations (2.11) is considered as what Thacker does. The equations read

\[
\eta_t + [u(D + \eta)]_x = 0
\]
\[
u_t + uu_x = -g\eta_x
\]

According to these equations, the instantaneous shoreline is determined by \( D + \eta = 0 \). The moving shoreline separates a region in which the total depth is positive from another region in which it is negative. It follows from the continuity
equation, which is the first equation in (3.17), that the volume of water within the region for which the total depth is positive remains constant in time as the shoreline moves about. The illustration of this problem is given by Figure 3.4.

Figure 3.4: A typical representation of a planar free surface in a parabolic canal.

It is assumed that there exists a solution for $u$ of the form

$$u = u_0 + u_x x$$  \hspace{1cm} (3.18)

where $u_0$ and $u_x$ are functions only of time. Substituting (3.18) into (3.17) leads to an idea that the solution for $\eta$ have the form

$$\eta = \eta_0 + \eta_x x + \frac{1}{2} \eta_{xx} x^2$$  \hspace{1cm} (3.19)

where

$$\eta_x = -\frac{1}{g} \left[ \frac{du_0}{dt} + u_0 u_x \right],$$  \hspace{1cm} (3.20)

$$\eta_{xx} = -\frac{1}{g} \left[ \frac{du_x}{dt} + u_x^2 \right].$$  \hspace{1cm} (3.21)

and where $\eta_0$ is a function only of $t$.

In order to satisfy the continuity equation, $D$ must be a polynomial similar to $\eta$. In particular, it can be assumed that the canal is a parabola of the form

$$D = D_0 \left( 1 - \frac{x^2}{L^2} \right).$$
The equilibrium shoreline is determined by the condition \( D = 0 \) which means \( x^2/L^2 = 1 \) or \( x = \pm L \).

Substituting (3.18) and (3.19) into the continuity equation leads to

\[
\left[ \eta_0 + \eta_x x + \frac{1}{2} \eta_{xx} x^2 \right]_t \cdot \left[ \left( u_0 + u_x x \right) (D_0 - \frac{x^2}{L^2} + \eta_0 + \eta_x x + \frac{1}{2} \eta_{xx} x^2) \right]_x = 0,
\]

which is equivalent to

\[
\left[ \eta_0 + \eta_x x + \frac{1}{2} \eta_{xx} x^2 \right]_t \cdot \left[ u_x (D_0 + \eta_0) + u_0 \eta_x \right] + \left[ u_0 (\eta_{xx} - \frac{2D_0}{L^2}) + 2u_x \eta_x \right] x + \frac{3}{2} u_x (\eta_{xx} - \frac{2D_0}{L^2}) x^2 = 0.
\]

Since the last equation holds for all points, then the time-varying coefficients of the linearly independent terms must separately equal to zero. This means that

\[
\frac{d\eta_0}{dt} + u_x (D_0 + \eta_0) + u_0 \eta_x = 0,
\]

(3.22)

\[
\frac{d\eta_x}{dt} + u_0 (\eta_{xx} - \frac{2D_0}{L^2}) + 2u_x \eta_x = 0,
\]

(3.23)

and

\[
\frac{d\eta_{xx}}{dt} + 3u_x (\eta_{xx} - \frac{2D_0}{L^2}) = 0.
\]

(3.24)

These three equations can determine the corresponding three unknown functions \( (\eta_0, \eta_x, \text{ and } \eta_{xx}) \) of time.

Now, assume that \( u_x = 0 \), so that according to (3.21) it is clear that \( \eta_{xx} = 0 \). Then only two functions, \( u_0 \) and \( h_0 \), must be determined. Equation (3.24) is identically satisfied, and equation (3.22) and (3.23) can be written respectively as

\[
\frac{d\eta_0}{dt} + \eta_x u_0 = 0
\]

(3.25)

and

\[
\frac{d\eta_x}{dt} - \frac{2D_0}{L^2} u_0 = 0.
\]

(3.26)

Substituting \( \eta_x \) given by (3.20) into (3.25) and (3.26) yield, respectively,

\[
\frac{d\eta_0}{dt} - \frac{u_0}{g} \frac{du_0}{dt} = 0,
\]

(3.27)

\[
\frac{d^2 u_0}{dt^2} + \frac{2gD_0}{L^2} u_0 = 0.
\]

(3.28)
CHAPTER 3. ANALYTICAL SOLUTIONS

The general solution of (3.28) is \( u_0 = c_1 \sin(\omega t) + c_2 \cos(\omega t) \) where \( \omega = \sqrt{\frac{2gD_0}{L^2}} \), and \( c_1 \) and \( c_2 \) are constant. Using the initial condition that \( u_0 = 0 \) for \( t = 0 \), it is obtained that \( c_2 = 0 \). Therefore, \( u_0 = c_1 \sin(\omega t) \). This implies that the horizontal displacement \( \delta \), which is the integral of the velocity, has the form of

\[
\delta = -\frac{c_1}{\omega} \cos(\omega t) + c_3
\]

where \( c_3 \) is another constant. Applying the conditions \( \delta(0) = A \) and \( \delta(\frac{\pi}{2\omega}) = 0 \), where the constant \( A \) determines the amplitude of the motion, result \( c_1 = -A\omega \) and \( c_3 = 0 \). Hence, the horizontal displacement is

\[
\delta = A \cos(\omega t),
\]

the shorelines has the formula

\[
x = \delta \pm L = A \cos(\omega t) \pm L,
\]

and the velocity has the form

\[
u = u_0 = -A\omega \sin(\omega t).
\]

(3.29)

To determine the form of the surface elevation \( \eta \), the quantities \( \eta_x \) and \( \eta_0 \) need to be specified. Based on equation (3.26) and (3.29), it is true that

\[
\frac{d\eta_x}{dt} = \frac{2D_0 u_0}{L^2} = \frac{2D_0(-A\omega \sin(\omega t))}{L^2} = -\frac{2A\omega D_0}{L^2} \sin(\omega t)
\]

and this yields

\[
\eta_x = \frac{2AD_0}{L^2} \cos(\omega t) + c_4
\]

for some constant \( c_4 \). Based on (3.27) and (3.29),

\[
\frac{d\eta_0}{dt} = \frac{u_0 du_0}{g \ dt} = \frac{A^2\omega^3}{2g} \sin(2\omega t)
\]

and this yields

\[
\eta_0 = -\frac{A^2\omega^2}{4g} \cos(2\omega t) + c_5
\]
3.3. **STEADY FLOW**

for some constant $c_5$. As a result, it can be evaluated easily that

$$
\eta = \eta_0 + \eta_x x
= \frac{A^2 D_0}{2L^2} [1 - 2 \cos^2(\omega t)] + c_5 + \left[ \frac{2AD_0}{L^2} \cos(\omega t) + c_4 \right] x
$$

For $x = 0$ and $t = \frac{\pi}{2\omega}$, it is clear that $\eta = 0$; this implies $c_5 = -\frac{A^2 D_0}{2L^2}$. In addition, for $x = A \cos(\omega t) + L$ and $t = \frac{\pi}{2\omega}$, the water surface is horizontal which means $\eta = 0$; this implies $c_4 = 0$. Therefore, the closed form of the water surface is given by

$$
\eta = \frac{2AD_0}{L^2} \cos(\omega t) \left[ x - \frac{A}{2} \cos(\omega t) \right].
$$

The solutions obtained above are under the assumption that the origin point is $O_1$ as given in Figure 3.4. If the origin is $O_2$, then a linear transformation is needed. Considering $O_2$ as the origin, the canal profile is then given by

$$
z = \frac{D_0}{L^2} x^2,
$$

the velocity and the shorelines are, still the same as before, given by

$$
u = -A \omega \sin(\omega t), \quad \text{and} \quad x = A \cos(\omega t) \pm L,
$$

while the water stage $w$ has the form

$$
w = D_0 + \frac{2AD_0}{L^2} \cos(\omega t) \left[ x - \frac{A}{2} \cos(\omega t) \right].
$$

Here the angular frequency of oscillation and the period are given by $\omega = \sqrt{\frac{2gD_0}{L}}$ and $T = \frac{2\pi}{\omega}$ respectively.

### 3.3 Steady Flow

This section presents analytical solutions of steady flow in a wide, rectangular, frictionless channel with a single obstruction somewhere along the channel. Houghton and Kasahara [25] have presented this problem and have considered four different cases of asymptotic solutions rather than merely a steady state solution. The derivation here follows from [25] and [27].

Consider a steady flow in a horizontal frictionless channel with bottom elevation $z(x)$ with an obstruction somewhere along the channel as illustrated in...
Figure 3.5. The initial water depth and initial velocity at a point upstream unaffected by the obstruction are given by \( h_0 \) and \( u_0 \) respectively. The SWE

\[
\begin{align*}
    h_t + (hu)_x &= 0, \\
    (hu)_t + (hu^2 + \frac{1}{2}gh^2)_x &= -ghz_x
\end{align*}
\]

(3.30)
govern this flow.

Figure 3.5: A typical representation of a steady flow with an obstruction.

It is assumed that the derivatives of \( u \) and \( h \) are smooth, then expanding the second equation in (3.30) leads to

\[
u_t h + uh_t + uz + u(uh)_x + ghh_x + ghz_x = 0.
\]

Substituting \( h_t \) given by the first equation in (3.30) into the last equation, and simplifying the result yield

\[
u_t + uu_x + g(h + z)_x = 0.
\]

Since the flow is assumed to be steady, the water depth and the velocity do not change with time which means that \( u_t = 0 \) and \( h_t = 0 \). Therefore, the SWE can be written

\[
\begin{align*}
    (uh)_x &= 0, \\
    uu_x + g(h + z)_x &= 0,
\end{align*}
\]

and by integration, these can be rewritten

\[
\begin{align*}
    uh &= q, \\
    \frac{1}{2}u^2 + h + z &= c,
\end{align*}
\]

(3.31)
for some constants \( q \) and \( c \). Equations (3.31) are valid for the whole domain.

Far from the obstacle, the bottom elevation, the water depth and the velocity are \( z(x) = 0 \), \( h(x,t) = h_0 \) and \( u(x,t) = u_0 \) respectively. Thus for the whole domain, it is clear that

\[
 uh = u_0 h_0
\]  

(3.32)

and

\[
 \frac{1}{2} u^2 + h + z = \frac{u_0^2}{2} + gh_0.
\]  

(3.33)

If \( u \) is eliminated from (3.33) by use of (3.32), it is obtained that

\[
 \frac{u_0^2 h_0^2}{2h^2} + g(h + z) = \frac{u_0^2}{2} + gh_0.
\]

Using the Froude number \( F_0 = u_0 / \sqrt{gh_0} \), it is obtained that

\[
 \frac{F_0^2 h_0^2}{2 h_0} + \frac{h}{h_0} + \frac{z}{h_0} = \frac{F_0^2}{2} + 1.
\]

Introducing dimensionless quantities \( y = h/h_0 \) and \( C = z/h_0 \) into the last equation and simplifying the result yield

\[
 y^3 + (C - \frac{1}{2} F_0^2 - 1) y^2 + \frac{1}{2} F_0^2 = 0.
\]  

(3.34)

Solving the cubic equation (3.34) for all points lead to the water surface profile, that is \( h = y h_0 \).

### 3.4 Concluding Remarks

In this chapter, analytical solutions of three different problems have been derived. Those solutions are based on an ideal fluid flow which means that any friction and Coriolis force are neglected. The results obtained in this chapter will be used to test numerical methods in Chapter 6 and 7.
CHAPTER 3. ANALYTICAL SOLUTIONS
Chapter 4

Finite Volume Methods

This chapter presents the Finite Volume Methods to solve the SWE. The general formulation for conservation laws is introduced and then several schemes of the methods are presented. Note that the methods can be applied to solve conservation laws and hyperbolic systems in general.

4.1 Approximation of the Quantities and Fluxes

A finite volume method is based on subdividing the spatial domain into cells and keeping track of an approximation to the integral of the quantity, \( q \), over each of these cells. In this paper, the \( i \)-th cell is denoted by

\[
C_i = (x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}})
\]

as shown in Figure 4.1.

Figure 4.1: Updating the cell averages \( Q^n_i \) by fluxes at the cell edges

\[ Q^n_{i-1} \quad Q^n_i \quad Q^n_{i+1} \]
In general, the integral form of the conservation law states

\[
\frac{d}{dt} \int_{C_i} q(x,t) \, dx = f[q(x_{i-\frac{1}{2}},t)] - f[q(x_{i+\frac{1}{2}},t)]
\] (4.1)

which can be written as

\[
\int_{C_i} q(x,t_{n+1}) \, dx = \int_{C_i} q(x,t_n) \, dx + \int_{t_n}^{t_{n+1}} f[q(x_{i-\frac{1}{2}})] \, dt - \int_{t_n}^{t_{n+1}} f[q(x_{i+\frac{1}{2}})] \, dt
\] (4.2)

Dividing equation (4.2) by \(\Delta x\) yields

\[
\frac{1}{\Delta x} \int_{C_i} q(x,t_{n+1}) \, dx = \frac{1}{\Delta x} \int_{C_i} q(x,t_n) \, dx - \frac{1}{\Delta x} \left( \int_{t_n}^{t_{n+1}} f[q(x_{i+\frac{1}{2}})] \, dt - \int_{t_n}^{t_{n+1}} f[q(x_{i-\frac{1}{2}})] \, dt \right)
\] (4.3)

Equation (4.3) can be stated as a numerical scheme of the form

\[
Q_{i+1}^n = Q_i^n - \frac{\Delta t}{\Delta x} \left( F_{i+\frac{1}{2}}^n - F_{i-\frac{1}{2}}^n \right)
\] (4.4)

Here, \(Q_i^n\) approximates the average value of the quantity \(q\) at time \(t^n := n\Delta t\), that is

\[
Q_i^n \approx \frac{1}{\Delta x} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} q(x,t_n) \, dx \equiv \frac{1}{\Delta x} \int_{C_i} q(x,t_n) \, dx
\] (4.5)

where \(\Delta x := x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}}\) is the length of the cell; and \(F_{i+\frac{1}{2}}^n\) is some approximation to the average flux across the cell interface at \(x = x_{i+\frac{1}{2}}\), that is

\[
F_{i+\frac{1}{2}}^n \approx \frac{1}{\Delta t} \int_{t_n}^{t_{n+1}} f[q(x_{i+\frac{1}{2}},t)] \, dt
\] (4.6)

**Remark 4.1.** The finite volume scheme (4.4) can also be interpreted as a finite difference scheme to the conservation law \(q_t(x,t) + f(q(x,t))_x = 0\). To illustrate this, we can rearrange equation (4.4) as

\[
\frac{Q_{i+1}^n - Q_i^n}{\Delta t} + \frac{F_{i+\frac{1}{2}}^n - F_{i-\frac{1}{2}}^n}{\Delta x} = 0
\]

Even though finite volume schemes can be equally well viewed as finite difference approximation, it should be noted that finite volume schemes are derived on the basis of the integral form of the conservation law.
So far, the general finite volume method for conservation laws, especially the scheme of average quantity in each cell, has been formulated. To apply the formulation, we need to define the form of the flux scheme, $F_{i+\frac{1}{2}}^n$. There are available several schemes which can be applied, all have their strengths and weaknesses depending on the specific problem to solve.

One simple scheme for calculating the flux is

$$F_{i+\frac{1}{2}}^n = \frac{1}{2} [f(Q_{i-1}^n) + f(Q_i^n)]$$

and so, the finite volume scheme (4.4) becomes

$$Q_i^{n+1} = Q_i^n - \frac{\Delta t}{2\Delta x} [f(Q_{i+1}^n) - f(Q_{i-1}^n)]$$

This scheme is very simple, but the downside is that it is generally unstable for hyperbolic problems and cannot be used even if the CFL condition is satisfied ([42], p.71).

An attempt to get a stable method is the Lax-Friedrichs scheme. This is similar to the unstable scheme (4.8), but the value $Q_i^n$ is replaced by the average of $Q_{i-1}^n$ and $Q_{i+1}^n$. Therefore, the scheme is

$$Q_i^{n+1} = \frac{1}{2}(Q_{i-1}^n + Q_{i+1}^n) - \frac{\Delta t}{2\Delta x} [f(Q_{i+1}^n) - f(Q_{i-1}^n)]$$

For a linear hyperbolic equation this method is stable if the Courant number is less than or equal to 1 ([42], p.71). The numerical flux used in this method looks like the unstable centered flux (4.7) with an additional term. The numerical flux reads

$$F_{i+\frac{1}{2}}^n = \frac{1}{2} [f(Q_{i-1}^n) + f(Q_i^n)] - \frac{\Delta x}{2\Delta t} (Q_i^n - Q_{i-1}^n)$$

Even though this method is stable, it is much more diffusive and gives results that are generally badly smeared unless a very fine grid is applied. In addition, it gives first order accuracy ([42], p.72).

There has been developed a scheme by implementing the Taylor series for the quantities, called Lax-Wendroff method (see [39, 42]). To simplify the presentation, consider a linear system $q_t + Aq_x = 0$. The Taylor series for $q(x, t_{n+1})$ can be written as

$$q(x, t_{n+1}) = q(x, t_n) + \Delta t q_t(x, t_n) + \frac{1}{2}(\Delta t)^2 q_{tt}(x, t_n) + \ldots.$$
From the given linear system, we have \( q_{tt} = A^2 q_{xx} \). Then, the Taylor series can be rewritten as

\[
q(x, t_{n+1}) = q(x, t_n) - \Delta t A q_x(x, t_n) + \frac{1}{2} (\Delta t)^2 A^2 q_{xx}(x, t_n) + \ldots.
\]

Keeping only the first three terms of the Taylor series above and replacing the spatial derivatives by central finite difference approximation leads to a scheme for the Lax-Wendroff method

\[
Q_{i}^{n+1} = Q_{i}^n - \frac{\Delta t}{2\Delta x} A (Q_{i+1}^n - Q_{i-1}^n) + \frac{1}{2} (\frac{\Delta t}{\Delta x})^2 A^2 (Q_{i-1}^n - 2Q_{i}^n + Q_{i+1}^n). \tag{4.9}
\]

Although the derivation of this method is based on a finite difference interpretation, we can also interpret it as a finite volume method of the form (4.4) with the flux function

\[
F_{i-\frac{1}{2}}^n = \frac{1}{2} A (Q_{i-1}^n + Q_{i}^n) - \frac{1}{2} \frac{\Delta t}{\Delta x} A^2 (Q_{i}^n - Q_{i-1}^n).
\]

One scheme proposed by Richtmyer is the Richtmyer method which is also known as the two-step Lax-Wendroff method. The method first approximates \( q \) at the midpoint in time \( t_{n+\frac{1}{2}} := t_n + \frac{1}{2} \Delta t \), then evaluates the flux at this point. After that, \( Q_{i}^{n+1} \) can be evaluated using formula (4.4). This method has the form

\[
F_{i-\frac{1}{2}}^n = f(Q_{i-1}^{n+\frac{1}{2}})
\]

where \( Q_{i-\frac{1}{2}}^{n+\frac{1}{2}} \) is obtained from the Lax-Friedrichs method at the cell interface \( \frac{1}{2} \Delta x \) and \( \frac{1}{2} \Delta t \) instead of \( \Delta x \) and \( \Delta t \) such that

\[
Q_{i-\frac{1}{2}}^{n+\frac{1}{2}} = \frac{1}{2} (Q_{i-1}^n + Q_{i+1}^n) - \frac{\Delta t}{2\Delta x} [f(Q_{i+1}^n) - f(Q_{i-1}^n)].
\]

The Richtmyer method may reduce the standard Lax-Wendroff method, but this method often leads to spurious oscillation in solutions when solving problems with discontinuous solutions ([42], p.72).

Still another method is the so called upwind method. Consider the equation \( q_t + \bar{u} q_x = 0 \). The upwind method has the form

\[
Q_{i}^{n+1} = Q_{i}^n - \frac{\bar{u} \Delta t}{\Delta x} (Q_{i+1}^n - Q_{i}^n)
\]

and the numerical flux is given by

\[
F_{i-\frac{1}{2}}^n = \bar{u}^- Q_{i}^n + \bar{u}^+ Q_{i-1}^n
\]
where \( \bar{u}^+ = \max(\bar{u}, 0) \), and \( \bar{u}^- = \min(\bar{u}, 0) \)

Methods that could also be considered are the Godunov-type Methods which can be applied to systems of equations. One of this type is the upwind Godunov method which works under the REA (Reconstruct-Evolve-Average) algorithm. This means: first, reconstruct a piecewise polynomial function \( \tilde{q}^n(x, t_n) \) defined for all \( x \) from the cell averages \( Q^n_i \); second, evolve the hyperbolic equation with this initial data to obtain \( \tilde{q}^n(x, t_{n+1}) \) a time \( \Delta t \) later; third, average this function over each grid cell to obtain new cell averages. This process is repeated in the next time step.

There is a finite volume Godunov-type method, that does not require exact or approximate Riemann solvers (which upwind Godunov type method requires in the second step of REA algorithm) developed by Kurganov, et al. [36], which is called central-upwind Godunov method. They have derived the conservative semi-discrete central-upwind scheme

\[
\frac{d}{dt}Q_i(t) = \frac{F_{i+\frac{1}{2}}(t) - F_{i-\frac{1}{2}}(t)}{\Delta x}
\]

and the numerical flux is given by

\[
F_{i+\frac{1}{2}}(t) = \frac{a^+_{i+\frac{1}{2}} f(q_{i+\frac{1}{2}}^-) - a^-_{i+\frac{1}{2}} f(q_{i+\frac{1}{2}}^+)}{a^+_{i+\frac{1}{2}} - a^-_{i+\frac{1}{2}}} + \frac{a^+_{i+\frac{1}{2}} a^-_{i+\frac{1}{2}}}{a^+_{i+\frac{1}{2}} - a^-_{i+\frac{1}{2}}} \left[ q_{i+\frac{1}{2}}^+ - q_{i+\frac{1}{2}}^- \right]
\]

where

\[
a^+_{i+\frac{1}{2}} = \max \left\{ \lambda_N \left( \frac{\partial f}{\partial q} (q_{i+\frac{1}{2}}^-) \right), \lambda_N \left( \frac{\partial f}{\partial q} (q_{i+\frac{1}{2}}^+) \right), 0 \right\},
\]

\[
a^-_{i+\frac{1}{2}} = \min \left\{ \lambda_1 \left( \frac{\partial f}{\partial q} (q_{i+\frac{1}{2}}^-) \right), \lambda_1 \left( \frac{\partial f}{\partial q} (q_{i+\frac{1}{2}}^+) \right), 0 \right\}.
\]

Here \( \lambda_1 < \ldots < \lambda_N \) are the eigenvalues of the Jacobian \( \frac{\partial f}{\partial q} \). In addition, \( q^+_{i+\frac{1}{2}} = p_{i+1}(x_{i+\frac{1}{2}}, t) \) and \( q^-_{i+\frac{1}{2}} = p_i(x_{i+\frac{1}{2}}, t) \) are the right and the left values at \( x = x_{i+\frac{1}{2}} \) of a conservative non-oscillatory piecewise polynomial interpolant

\[
\tilde{q}(x, t) = \sum_i p_i(x, t) \chi_i
\]

which is reconstructed at each time step from the previously computed cell averages, \( \{Q^n_i(t)\} \), where

\[
Q^n_i(t) = \frac{1}{\Delta x} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} q(x, t^n) dx.
\]
The pieces \( \{p_i(\cdot, t)\} \) are polynomials of a given degree, and \( \mathcal{X}_i \) is the characteristic function which is defined by

\[
\mathcal{X}_i = \begin{cases} 
1, & \text{if } x_{i-1/2} < x < x_{i+1/2} \\
0, & \text{otherwise}
\end{cases}
\]

The order of the piecewise-polynomial reconstruction determines the order of the numerical fluxes (4.11).

### 4.2 Velocity Stabilization

When the momentum \( p = uh \) and the water stage \( w \) are maintained to be conserved, the flux calculation requires the value of velocity \( u \) which can be obtained as \( u = p/h \), where the water depth \( h \) can be calculated as \( h = w - z \). If the water depth approaches zero, these calculations become numerically unreliable and will typically cause unphysical speeds.

The problems associated with these numerical instabilities due to small water depths near a wet/dry boundary can be alleviated by employing a flux limiter that ensures that unphysical fluxes are never encountered (see [59]). The flux limiter, which replaces the calculations above with limited approximation, is applied as

\[
u = \frac{p}{h + \epsilon/h}
\]

where \( \epsilon \) is a regularisation parameter that controls the minimal magnitude of the denominator. This new calculation can be done since

\[
\lim_{h \to 0} u = \lim_{h \to 0} \frac{p}{h + \epsilon/h} = 0
\]

and

\[
\lim_{h \to \infty} u = \lim_{h \to \infty} \frac{p}{h + \epsilon/h} = \frac{p}{h} = u.
\]

### 4.3 Slope Limiting

This subsection reviews several slope limiters to reconstruct the piecewise polynomials involved in calculating the numerical approximation of the quantities to reduce the artificial oscillation in the numerical result. The review is limited to present first and second order method (as also presented in [27]).
4.3. SLOPE LIMITING

In the construction of central-upwind Godunov method, a piecewise polynomial initial condition is used to get a non-oscillatory piecewise polynomial \( \tilde{q}(x,t^n) \) reconstruction. At each cell interface, the left and right states are estimated by that reconstruction polynomial. The simplest reconstruction is a piecewise constant reconstruction which produces a first order method. Note that to have a second order method, a piecewise linear reconstruction is required; to get a third order method, a piecewise quadratic reconstruction is required; and so on.

For second order methods, a piecewise linear reconstruction in the \( i \)th cell is given by

\[
\tilde{q}(x,t^n) = Q_i^n + \sigma_i^n (x - x_i)
\] (4.12)

where \( \sigma_i \) is the slope of the cell. If each \( \sigma_i^n \) in (4.12) is zero, the reconstruction is piecewise constant (but the accuracy may be second order still). Particularly, the left and the right states of a cell interface are approximated by

\[
q_{i+\frac{1}{2}} = Q_i^n + \sigma_i^n \frac{\Delta x}{2} \quad \text{and} \quad q_{i+\frac{1}{2}}^+ = Q_{i+1}^n + \sigma_{i+1}^n \frac{\Delta x}{2}
\]

Second order or higher reconstructions often introduce artificial oscillation near discontinuities. Therefore a technique to avoid oscillations is needed. One way to overcome the oscillation in a second order method is by introducing a slope-limiter which gives the value of the slope, \( \sigma_i \), in each cell.

There are several slope-limiter functions which are common to use. One of them is the minmod slope-limiter

\[
\sigma_i^n = \minmod\left(\frac{Q_i^n - Q_{i-1}^n}{\Delta x}, \frac{Q_{i+1}^n - Q_i^n}{\Delta x}\right)
\]

where the \( \minmod \) of two arguments is defined by

\[
\minmod(a,b) = \begin{cases} 
    a & \text{if } |a| < |b| \text{ and } ab > 0 \\
    b & \text{if } |a| > |b| \text{ and } ab > 0 \\
    0 & \text{if } ab \leq 0
\end{cases}
\]

Another possibility is the minmod-Kurganov [36] which is given by

\[
\sigma_i^n = \minmod\left(\beta \frac{Q_i^n - Q_{i-1}^n}{\Delta x}, \frac{Q_{i+1}^n - Q_i^n}{2\Delta x}, \beta \frac{Q_{i+1}^n - Q_i^n}{\Delta x}\right)
\]

where \( 1 \leq \beta \leq 2 \), and when \( \beta = 2 \) this limiter is known as Monotonized Central-difference (MC) limiter. Another limiter that could be chosen is superbee which is defined by

\[
\sigma_i^n = \maxmod(\sigma^{(1)}, \sigma^{(2)})
\]
CHAPTER 4. FINITE VOLUME METHODS

where

\[ \sigma^{(1)} = \text{minmod} \left( \frac{Q^n_{i+1} - Q^n_i}{\Delta x}, 2 \frac{Q^n_i - Q^n_{i-1}}{\Delta x} \right) \]

\[ \sigma^{(2)} = \text{minmod} \left( 2 \frac{Q^n_{i+1} - Q^n_i}{\Delta x}, \frac{Q^n_i - Q^n_{i-1}}{\Delta x} \right) \]

and

\[ \text{maxmod}(a, b) = \begin{cases} 
  a & \text{if } |a| > |b| \text{ and } ab > 0 \\
  b & \text{if } |a| < |b| \text{ and } ab > 0 \\
  0 & \text{if } ab \leq 0
\end{cases} \]

Still two other limiters are the van Leer limiter and van Albada limiter defined by

\[ \sigma^n_i = \frac{|a|a + |b|b}{|a| + |b|} \quad \text{and} \quad \sigma^n_i = \frac{a^2b + ab^2}{a^2 + b^2} \]

respectively, where \( a \) and \( b \) are given by

\[ a = \frac{Q^n_{i+1} - Q^n_i}{\Delta x} \quad \text{and} \quad b = \frac{Q^n_i - Q^n_{i-1}}{\Delta x} \]

Finally, there is a limiter called pyvolution which is available in ANUGA software package (consult \[27\]).

The performance of these limiters will be investigated in the simulation presented in Chapter 6 and 7.

4.4 Temporal Discretisation

The central-upwind Godunov-type method requires not only spatial discretisation, but also a discretisation of time. In other words, the semi-discrete conservative scheme (4.10),

\[ \frac{d}{dt}Q_i(t) = -\frac{F_{i+\frac{1}{2}}(t, Q_i) - F_{i-\frac{1}{2}}(t, Q_i)}{\Delta x} = H_i(t, Q_i) \]

which is a system of time-dependent ordinary differential equations need to be discretised with respect to time to be solved numerically. To do so, an explicit Runge-Kutta method such as first and second order Runge-Kutta schemes can be applied. As stated in [27] as well, according to C. Shu [63] this type of discretisation maintains strong stability properties (SSP) in any semi-norm of the forward Euler step.
The first order Runge-Kutta scheme is defined by

\[ Q_{i}^{n+1} = Q_{i}^{n} + \Delta t H_{i}(t^{n}, Q_{i}^{n}) \]

where

\[ H_{i}(t^{n}, Q_{i}^{n}) = \frac{F_{i-\frac{1}{2}}^{n} - F_{i+\frac{1}{2}}^{n}}{\Delta x}. \]

This temporal discretisation is also known as first order explicit forward Euler scheme. It has been known that the first order scheme is efficient to implement, but it is inconsistent with the second order accuracy of the spatial discretisation. Therefore, considering second order temporal discretisation is important.

The second order Runge-Kutta scheme is given in three steps by

\[ Q_{i}^{(1)} = Q_{i}^{n} + \Delta t H_{i}(t^{n}, Q_{i}^{n}), \]
\[ Q_{i}^{(2)} = Q_{i}^{(1)} + \Delta t H_{i}(t^{n+1}, Q_{i}^{(1)}), \]
and

\[ Q_{i}^{n+1} = \frac{1}{2}(Q_{i}^{n} + Q_{i}^{(2)}). \]

4.5 Boundary Conditions

To update the cell average \( Q_{i}^{n} \), the neighboring cell values \( Q_{i-1}^{n} \) and \( Q_{i+1}^{n} \) have to be known, and values further away might be needed to compute the fluxes \( F_{i-\frac{1}{2}}^{n} \) and \( F_{i+\frac{1}{2}}^{n} \) in some methods. In practice, some finite set of grid cells covering a bounded domain must always be computed, and the required neighboring information in the first and the last cells will not be available. Instead, some set of physical boundary conditions must be used.

There are two main approaches to update the cells on both ends of the domain. The first approach is using a special formula which is applied on the cells near the boundary. The formula usually depends on the type of boundary conditions and the method which is used. The second approach is setting ghost cells, that is extending the computational domain to include several additional cells on each end. The values of ghost cells are set at the beginning of each time step, and usually depend on the type of boundary conditions and the interior values.

The values of ghost cells can be defined using either absorbing boundary conditions or reflective boundary conditions. Absorbing boundary conditions
are supposed to completely absorb any wave which hits the boundaries. A set of these boundary conditions can be generated using extrapolation, such as zero-order extrapolation. On the contrary, reflective boundary conditions introduce reflections of the outgoing waves. For these boundary conditions, if a cell on the boundary has a quantity \( q = [h, uh]^T \), then the quantity of the corresponding ghost cell is given by \( Q_b = [h, -uh]^T \).

### 4.6 Properties of Finite Volume Methods

#### 4.6.1 Convergence

One essential requirement to judge how good a particular flux function for numerical computation is that the resulting method should be *convergent*. This means that the numerical solution should converge to the true solution of the differential equation as the grid is refined, i.e. as \( \Delta x, \Delta t \to 0 \). According to Lax Equivalence Theorem, convergence requires two conditions:

- The method must be *consistent* with the differential equation. Consistent means that the local truncation error vanishes as \( \Delta t \to 0 \) for all smooth functions satisfying the differential equation. In other words, the method approximates the differential equation well locally and introduces a small error in any single time step.

- The method must be *stable*. This means that the local small errors made in each time step do not grow too fast in later time steps and hence a bound on the global error can be obtained in terms of these local errors.

If a bound on the local error can be found in an appropriate sense, then stability can be used to convert this into a bound on the global error that can be used to prove convergence.

#### 4.6.2 Stability

A necessary condition for stability of any numerical method is as follows. *A numerical method can be convergent only if its numerical domain of dependence contains the true domain of dependence of the partial differential equations, at
least in the limit as $\Delta t$ and $\Delta x$ approach to zero.
This condition is called **CFL condition**, named after Courant, Friedrichs, and Lewy. It is important that the CFL condition is only a necessary condition for stability, and it is not always sufficient to guarantee stability. This means that there exist numerical methods which are unstable even though the CFL condition is satisfied.

Define the Courant number by

$$
\nu = \frac{\Delta t}{\Delta x} \max_p |\lambda_p|
$$

For a three-point stencil the CFL condition requires $\nu \leq 1$, which means that the necessary condition for stability is that the Courant number must be less than or equal to 1. For a centered five-point stencil the CFL condition requires $\nu \leq 2$. For hyperbolic equations, the explicit methods and grids are typically used for which the Courant number is smaller than 1; this is in order to keep $\Delta t / \Delta x$ fixed as the grid is refined.

### 4.6.3 Consistency

A general explicit numerical method can be written as

$$Q^{n+1} = \mathcal{N}(Q^n),$$

where $\mathcal{N}(\cdot)$ denotes the numerical operator mapping the approximate solution at one time step to the approximate solution at the next. The **one-step error** is defined by

$$\text{one-step error} = \mathcal{N}(q^n) - q^{n+1},$$

where

$$q^n = q^n_i = \frac{1}{\Delta x} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} q(x, t^n)dx$$

is the true solution restricted to the grid. The **local truncation error** $\tau^n$ is defined by dividing the one-step error by $\Delta t$, that is

$$\tau^n = \frac{1}{\Delta t} [\mathcal{N}(q^n) - q^{n+1}].$$

By following this, the method is said to be **consistent** with the differential equation if the local truncation error vanishes as $\Delta t \to 0$ for all smooth functions $q(x, t)$ satisfying the differential equation.
Recall that an explicit finite volume method can be written as

\[ N(Q^n) = Q^n + \Delta t H^n, \]

where for first and second order temporal discretisation

\[ H^n = H(t^n, Q^n) \quad \text{and} \quad H^n = \frac{1}{2} [H(t^n, Q^n) + H(t^{n+1}, Q^{(1)})] \]

respectively and \( Q^{(1)} = Q^n + \Delta t H(t^n, Q^n) \) as presented in Section 4.4. Therefore, the local truncation error \( \tau^n \) for finite volume methods is given by

\[ \tau^n = \frac{q^n - q^{n+1}}{\Delta t} + H^n. \]

Furthermore, the \( L^1 \) absolute error \( E \) given by

\[ E = \frac{1}{N} \sum_{i=1}^{N} |q(x_i) - Q_i| \]

can be used to quantify the numerical error, where \( N \) is the number of grids. Absolute error, rather than relative error, is chosen so that the errors occurring at small values of quantities do not dominate the calculation.

### 4.7 Concluding Remarks

In this chapter, the finite volume formulations for conservation laws have been presented. The resulting finite volume schemes can often be interpreted as finite difference scheme to the conservation laws. Furthermore, slope limiters, temporal discretisations, and boundary conditions related to those formulations were also reviewed. The central-upwind schemes viewed in this chapter with additional well-balanced terms given in Chapter 5 will be simulated and the results will be presented in Chapter 6 and 7.
Chapter 5

Well-Balanced Finite Volume Methods

If a numerical scheme does not preserve the hydrostatic balance at the discrete level, it may result in spurious oscillation. This suggests that well-balancing of the numerical schemes needs to be taken into account. In this chapter, the concepts of well-balancing and several types of well-balanced schemes are presented.

5.1 Basic Concepts of Well-Balancing

A simple form of one-dimensional SWE is given by (2.10). In (2.10), two-dimensional effects, bottom friction, Coriolis forces arising in a rotational frame, wind forces, and vertical variations of the velocity fields are neglected. The derivation of (2.10) can be found in some references such as [27] and [48].

In this chapter, the idea of well-balancing and the well-balanced finite volume schemes are presented. To begin the discussion, the following is the definition of equilibrium state.

**Definition 5.1.** A state of shallow water is in *equilibrium* or *stationary* or *steady* if the mass flux over the depth is constant and the mechanical energy at every point on the water surface is also constant.

Recall that the mass flux at a point is \( \rho u \), where \( \rho \) is the mass density and \( u \) is the velocity, then the mass flux over the depth is \( \rho hu \). This can be found by integrating \( \rho u \) over the depth. In addition, the mechanical energy of every
particle on the water surface is the summation of potential energy and kinetic energy, that is \( \frac{1}{2}mu^2 + mgw \) where \( m \) is the particle mass. Therefore, according to Definition 5.1 the equilibrium conditions are

\[
\rho hu = \text{cst, } \quad \frac{1}{2}mu^2 + mgw = \text{cst}
\]

which can be simplified to

\[
hu = \text{cst, } \quad \frac{1}{2}u^2 + gw = \text{cst.} \tag{5.1}
\]

An example of (unperturbed) equilibrium state, which is also the one considered in this paper, is the lake at rest which is given by

\[
u \equiv 0 \text{ and } w \equiv \text{cst}. \tag{5.2}
\]

It is clear that the conditions of the lake at rest satisfy (5.1). Other equilibrium states with nonvanishing velocity can also be considered, but they are beyond the scope of this thesis.

The characterisation (5.2) of lake at rest comes from the fact that the water is still, \( u = 0 \), and the water depth stays the same all the time, \( h_t = 0 \). Substituting these into (2.10), it is found that

\[
0 = \left( \frac{1}{2}gh^2 \right)_x + ghz_x \tag{5.3}
\]

which is called hydrostatic balance. Applying the chain rule of differentiation and dividing the result by \( h \), it is obtained that \( (h + z)_x = 0 \). This means that the characterisation of the lake at rest can be written by

\[
w = h + z = \text{cst}
\]

together with \( u = 0 \) as expected.

Based on the explanation above, the hydrostatic balance (5.3) is equivalent to the "lake at rest" conditions (5.1). The first term in (5.3) is the hydrostatic pressure, modelling the tendency of a column of water to collapse vertically and at the same time expand laterally under the influence of gravity. The second term is the gravitational acceleration down an inclined bed \( z(x) \). The summation of the first and the second term results a balance of momentum flux and momentum source term.
A numerical scheme should be able to preserve the unperturbed equilibrium state at the discrete level. If a numerical scheme does not preserve the hydrostatic balance (5.3) at the discrete level, it may result in spurious oscillations, also known as numerical storm, as shown in [50]. Those oscillations might be reduced by running such a scheme with a much finer grid, but the computation would be very costly. Another way, which is the goal of this thesis, is by using a so called well-balanced scheme defined as follows.

**Definition 5.2.** A finite volume scheme is *well-balanced* if it preserves the steady state of a lake at rest

\[ hu = \text{cst}, \quad \text{and} \quad \frac{1}{2} u^2 + gw = \text{cst} \]

or the hydrostatic balance

\[ 0 = \left( \frac{1}{2} gh^2 \right)_x + ghz_x \]

Several natural properties that should be satisfied by a scheme in general and a well-balanced scheme particularly are conservativity of the water depth \( h \) and nonnegativity of \( h \), in order to handle the dry states \( h = 0 \) and transcritical flows when the Jacobian matrix \( F' \) of the flux function becomes singular, and in order to satisfy a discrete entropy inequality. The difficulty when the Jacobian matrix \( F' \) of the flux function becomes singular is related to resonance, and theoretical studies on resonance can be found such as in [26, 46] (see Chapter 3 in [6] for definition of resonance which is also known as critical point). The discrete entropy inequality property ensures the admisibility of shocks and gives overall the nonlinear stability of the scheme. As stated in [2], the exact Godunov scheme satisfies these all requirements, but in practice it is computationally too expensive, and not easily adaptable to more complex systems. Attempts to derive an approximate solver (such as kinetic solver, relaxation solver, and Roe solver) satisfying all the requirements are presented in [6]. However, these approximate solver methods are still quite heavy in practice.

It is a challenge to explain how it is possible, by a very flexible approach involving a hydrostatic reconstruction, to obtain a well-balanced scheme satisfying all the above requirements and which is computationally inexpensive. In contrast with kinetic solver and relaxation solver, it is generic in the sense that it can be
used in conjunction with any given numerical flux for the homogeneous SWE, that is a Saint-Venant problem with constant or flat topography.

## 5.2 Semi-Discrete First Order Well-Balanced Schemes

The following derivation is based on [2]. Consider the SWE

\[ q_t + [f(q)]_x = S(x) \]  

(5.4)

of the SWE, where \( q, f, \) and \( S \) are given by

\[
q = \begin{bmatrix} h \\ uh \end{bmatrix}, \quad f = \begin{bmatrix} uh \\ uh^2 + gh^2/2 \end{bmatrix}, \quad \text{and} \quad S = \begin{bmatrix} 0 \\ -ghz_x \end{bmatrix}.
\]

In the semidiscrete case, the system (5.4) has the form

\[
\Delta x_i \frac{d}{dt} Q_i(t) + F_{i+\frac{1}{2}} - F_{i-\frac{1}{2}} = S_i \]

(5.5)

where \( \Delta x_i \) denotes a variable mesh size \( \Delta x_i = x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}} \), and the cell-centered vector of discrete unknown quantities is

\[
Q_i(t) = \begin{bmatrix} h_i(t) \\ h_i(t)u_i(t) \end{bmatrix}.
\]

Classically, to produce a first order accurate scheme, the fluxes are computed as \( F_{i+1/2} = \mathcal{F}(Q_i(t), Q_{i+1}(t)) \) where the numerical flux \( \mathcal{F} \) is computed by an approximate \textit{Riemann solver} \(^*\) which is an approximate solution of the Riemann problem. Since cell-centered evaluations of the source term in (5.5) will generally not be able to preserve the steady state of a lake of rest \( (h_i + z_i = \text{cst}, u_i = 0) \), Audusse, et al. in [2] propose a discretisation of the source term using hydrostatic balance reconstruction.

The reconstruction is motivated by a hydrostatic balance equation (5.3). In the associated asymptotic limit for nearly hydrostatic flows (which means \( u \ll \)

\(^*\)Several approximate Riemann solvers have been developed, such as Godunov, Roe, Kinetic, etc.
\( \sqrt{g h} \), the leading term of \( \bar{h} \) expansion with respect to temporal variable adjusts so as to satisfy equation (5.3) such that

\[
0 = \left(\frac{1}{2} g \bar{h}^2 \right)_x + g \bar{h} z_x. \tag{5.6}
\]

Note that taking only the leading term of the water depth expansion here leads to a first order accuracy of the source discretisation. An approximation to the net source term can be found by integrating the equation (5.6) over the \( i \)th grid cell

\[- \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} g \bar{h} z_x \, dx = \frac{g}{2} \bar{h}_{i+\frac{1}{2}}^2 - \frac{g}{2} \bar{h}_{i-\frac{1}{2}}^2. \]

This motivates the source term discretisation as

\[
S_i = \begin{bmatrix}
0 \\
\frac{g}{2} \bar{h}_{i+\frac{1}{2}}^2 - \frac{g}{2} \bar{h}_{i-\frac{1}{2}}^2
\end{bmatrix}.
\]

Any hydrostatic state is maintained exactly if the momentum fluxes and the locally reconstructed depths satisfy

\[
F^{hu}_{i+\frac{1}{2}} = \frac{1}{2} g \bar{h}_{i+\frac{1}{2}}^2 = \frac{1}{2} g \bar{h}_{i+\frac{1}{2}}^2 +.
\]

This motivates the flux functions

\[
F_{i+\frac{1}{2}} = \mathcal{F}(Q_{i+\frac{1}{2}}, Q_{i+\frac{1}{2}+})
\]

which gives this property if \( Q_{i+1/2-} = Q_{i+1/2+} = (h_{i+1/2-}, 0) = (h_{i+1/2+}, 0) \) holds for hydrostatic states.

For the lake at rest, \( h + z = \text{const.} \) This implies that \( h_{i+1/2} + z_{i+1/2} = h_i + z_i = h_{i+1} + z_{i+1} \), and the reconstruction of the leading term depths is

\[
\bar{h}_{i+\frac{1}{2}} = h_i + z_i - z_{i+\frac{1}{2}}, \quad \text{and} \quad \bar{h}_{i+\frac{1}{2}+} = h_{i+1} + z_{i+1} - z_{i+\frac{1}{2}}. \tag{5.7}
\]

To ensure nonnegativity of the water depth even when capturing dry regions, a truncation of the leading term depths in (5.7) is needed such that \( h_{i+1/2-} = \max(0, \bar{h}_{i+1/2-}) \) and \( h_{i+1/2+} = \max(0, \bar{h}_{i+1/2+}) \). In addition, the cell interface depth is evaluated by an upwind type of the form

\[
z_{i+\frac{1}{2}} = \max(z_i, z_{i+1}). \tag{5.8}
\]
These choices ensures that $0 \leq h_{i+1/2}^- \leq h_i$ and $0 \leq h_{i+1/2}^+ \leq h_{i+1}$ to satisfy the nonnegativity property of the water depth.

In summary, the first order well-balanced finite volume scheme is

$$\Delta x_i \frac{d}{dt} Q_i(t) + F_{i+\frac{1}{2}} - F_{i-\frac{1}{2}} = S_i, \quad (5.9)$$

where

$$F_{i+\frac{1}{2}} = F(Q_{i+\frac{1}{2}^-}, Q_{i+\frac{1}{2}^+}),$$

$$Q_{i+\frac{1}{2}^-} = \begin{bmatrix} h_{i+1/2}^- \\ h_{i+1/2}^- - u_i \end{bmatrix}, \quad Q_{i+\frac{1}{2}^+} = \begin{bmatrix} h_{i+1/2}^+ \\ h_{i+1/2}^+ + u_i \end{bmatrix},$$

$$h_{i+\frac{1}{2}^-} = \max(0, h_i + z_i - z_{i+\frac{1}{2}}), \quad h_{i+\frac{1}{2}^+} = \max(0, h_{i+1} + z_{i+1} - z_{i+\frac{1}{2}}), \quad (5.10)$$

and

$$S_i = \begin{bmatrix} 0 \\ \frac{g}{2} h_i^2 \end{bmatrix}, \quad S_{i+\frac{1}{2}^-} = \begin{bmatrix} 0 \\ \frac{g}{2} h_{i+1/2}^- \end{bmatrix}, \quad S_{i+\frac{1}{2}^+} = \begin{bmatrix} 0 \\ \frac{g}{2} h_{i+1/2}^+ \end{bmatrix}.$$

By defining

$$S_{i+\frac{1}{2}^-} = \begin{bmatrix} 0 \\ \frac{g}{2} h_{i+1/2}^- \end{bmatrix}, \quad S_{i+\frac{1}{2}^+} = \begin{bmatrix} 0 \\ \frac{g}{2} h_{i+1/2}^+ \end{bmatrix},$$

it is clear that $S_i = S_{i+1/2}^- + S_{i+1/2}^+$. This means that the source may be considered as being distributed to the cell interface. Furthermore, the scheme (5.9)-(5.11) can be rewritten as

$$\Delta x_i \frac{d}{dt} Q_i + F^r(Q_i, Q_{i+1}, z_i, z_{i+1}) - F^l(Q_{i-1}, Q_i, z_{i-1}, z_i) = 0 \quad (5.12)$$

where the right and the left numerical fluxes are

$$F^r(Q_i, Q_{i+1}, z_i, z_{i+1}) = F_{i+\frac{1}{2}} - S_{i+\frac{1}{2}^-}, \quad (5.13)$$

and

$$F^l(Q_{i-1}, Q_i, z_{i-1}, z_i) = F_{i-\frac{1}{2}} + S_{i-\frac{1}{2}^+}. \quad (5.14)$$

Audusse, et al. in [2] prove that, given a consistent numerical flux $F$ for the homogeneous problem that preserves nonnegativity of $h_i(t)$ and satisfies an in-cell entropy inequality, the finite volume scheme (5.9)-(5.11) preserves the nonnegativity of $h_i(t)$, preserves the steady state of the lake at rest, is consistent with (2.10), and satisfies an in-cell entropy inequality.

The argument that shows how the first order schemes lead to a well-balanced scheme for the lake at rest is given in the proof of the following theorem.
5.3. FULLY DISCRETE FIRST ORDER WELL-BALANCED SCHEMES

Theorem 5.3. The first order semidiscrete scheme

\[ \Delta x_i \frac{d}{dt} Q_i + \mathcal{F}^r(Q_i, Q_{i+1}, z_i, z_{i+1}) - \mathcal{F}^l(Q_{i-1}, Q_i, z_{i-1}, z_i) = 0 \quad (5.15) \]

where the right and the left numerical fluxes are given by

\[ \mathcal{F}^r(Q_i, Q_{i+1}, z_i, z_{i+1}) = F_{i+\frac{1}{2}} - S_{i+\frac{1}{2}} - , \quad (5.16) \]

and

\[ \mathcal{F}^l(Q_{i-1}, Q_i, z_{i-1}, z_i) = F_{i-\frac{1}{2}} + S_{i-\frac{1}{2}} + . \quad (5.17) \]

is well-balanced.

Proof. Consider the steady state of the lake at rest. Recall from (5.8) and (5.10) that in this case the hydrostatic reconstruction \( Q \) is continuous across the interface and \( u = 0 \). This results that any consistent numerical flux \( F_{i+1/2} \) and \( F_{i-1/2} \) will reduce to the hydrostatic pressure term,

\[ F_{i+\frac{1}{2}} = \mathcal{F}(Q_{i+\frac{1}{2}}, Q_{i+\frac{1}{2}+}) = \begin{bmatrix} 0 \\ \frac{g}{2} h_{i+1/2} \end{bmatrix} \quad (5.18) \]

and

\[ F_{i-\frac{1}{2}} = \mathcal{F}(Q_{i-\frac{1}{2}-}, Q_{i-\frac{1}{2}+}) = \begin{bmatrix} 0 \\ \frac{g}{2} h_{i-1/2} \end{bmatrix} . \quad (5.19) \]

Substituting this information in (5.16) and (5.17) to (5.15), and using (5.18) and (5.19), it is obtained that at steady state \( Q \) remains constant, so that the scheme is well-balanced. \( \square \)

5.3 Fully Discrete First Order Well-Balanced Schemes

The fully discrete version of (5.12) can be formed by discretising the time variable. This reads

\[ Q_i^{n+1} - Q_i^n + \frac{\Delta t}{\Delta x_i} \left( \mathcal{F}^r(Q_i^n, Q_{i+1}^n, z_i, z_{i+1}) - \mathcal{F}^l(Q_{i-1}^n, Q_i^n, z_{i-1}, z_i) \right) = 0. \quad (5.20) \]

In order to preserve the nonnegativity of \( h_i \), Audusse, et al. in [2] shows that the CFL condition\(^1\) which needs to be used in the fully discrete scheme is still

---

\(^1\)The CFL condition is applied to guarantee stability.
the same as that of the homogeneous solver. In addition, the consistency and the well-balanced property are still valid, and the data which violates the fully discrete entropy inequality are not preserved by the scheme.

### 5.4 Second Order Well-Balancing

A common way to get a second order extension of a given first order scheme is to compute the fluxes from limited reconstructed values on both sides of each interface rather than cell centered values.

According to Bouchut [6], to preserve consistency not only the reconstructed values $Q_{i,r}$ at $i + 1/2$ and $Q_{i+1,l}$ at $i + 1/2$ need to be defined, but also $z_{i,r}$, $z_{i+1,l}$, and a cell-centered source term $S_{ci}$ must be added. After second order reconstruction of these values are done, the hydrostatic reconstruction at each interface as in Section 5.2 can be applied. As a result, the second order well-balanced scheme reads

$$\Delta t \frac{d}{dt} Q_i(t) + F_{i+\frac{1}{2}} - F_{i-\frac{1}{2}} = S_i + S_{ci},$$

where

$$F_{i+\frac{1}{2}} = F(Q_{i+\frac{1}{2}-}, Q_{i+\frac{1}{2}+}),$$

$$Q_{i+\frac{1}{2}-} = \begin{bmatrix} h_{i+1/2-} \\ h_{i+1/2-} - u_{i,r} \end{bmatrix}, \quad Q_{i+\frac{1}{2}+} = \begin{bmatrix} h_{i+1/2+} \\ h_{i+1/2+} + u_{i+1,l} \end{bmatrix},$$

$$h_{i+\frac{1}{2}-} = \max(0, h_{i,r} + z_{i,r} - z_{i+\frac{1}{2}}), \quad h_{i+\frac{1}{2}+} = \max(0, h_{i+1,l} + z_{i+1,l} - z_{i+\frac{1}{2}}),$$

$$z_{i+\frac{1}{2}} = \max(z_{i,r}, z_{i+1,l}),$$

and a simple well-balanced discretisation for the centered source term $S_{ci}$ can be written as

$$S_{ci} = \begin{bmatrix} 0 \\ g \frac{h_{i+1/2-}}{2} - g \frac{h_{i,r}}{2} (z_{i,l} - z_{i,r}) \end{bmatrix}.$$

(5.22)

Again, similar with Section 5.2, by defining

$$S_{i+\frac{1}{2}-} = \begin{bmatrix} 0 \\ \frac{g}{2} h_{i+1/2-} - \frac{g}{2} h_{i,r}^2 \end{bmatrix}, \quad \text{and} \quad S_{i-\frac{1}{2}+} = \begin{bmatrix} 0 \\ \frac{g}{2} h_{i+1/2+} - \frac{g}{2} h_{i-1/2+} \end{bmatrix},$$

it is clear that

$$S_i = S_{i+\frac{1}{2}-} + S_{i-\frac{1}{2}+}.$$
5.4. SECOND ORDER WELL-BALANCING

Therefore, the scheme (5.21)-(5.22) can be rewritten as
\[
\Delta x_i \frac{d}{dt} + \mathcal{F}^r(Q_{i,r}, Q_{i+1,l}, z_{i,r}, z_{i+1,l}) - \mathcal{F}^l(Q_{i-1,r}, Q_{i,l}, z_{i-1,r}, z_{i,l}) = S_{ci},
\]
(5.23)
where the right and the left numerical fluxes are
\[
\mathcal{F}^r(Q_i, Q_{i+1}, z_{i}, z_{i+1}) = F_{i+\frac{1}{2}} - S_{i+\frac{1}{2}^-},
\]
(5.24)
and
\[
\mathcal{F}^l(Q_{i-1}, Q_{i}, z_{i-1}, z_{i}) = F_{i-\frac{1}{2}} + S_{i-\frac{1}{2}^+}.
\]
(5.25)

**Remark 5.4.** Here \(z_i\) do not depend on time, but the reconstructed values \(z_{i,l}\) and \(z_{i,r}\) could depend on time via a coupling with \(Q_i\) in the reconstruction step.

**Theorem 5.5.** The second order semidiscrete scheme
\[
\Delta x_i \frac{d}{dt} + \mathcal{F}^r(Q_{i,r}, Q_{i+1,l}, z_{i,r}, z_{i+1,l}) - \mathcal{F}^l(Q_{i-1,r}, Q_{i,l}, z_{i-1,r}, z_{i,l}) = S_{ci},
\]
(5.26)
together with
\[
\mathcal{F}^r(Q_i, Q_{i+1}, z_{i}, z_{i+1}) = F_{i+\frac{1}{2}} - S_{i+\frac{1}{2}^-},
\]
(5.27)
\[
\mathcal{F}^l(Q_{i-1}, Q_{i}, z_{i-1}, z_{i}) = F_{i-\frac{1}{2}} + S_{i-\frac{1}{2}^+},
\]
(5.28)
and
\[
S_{ci} = \begin{bmatrix} 0 \\ \frac{g}{2} h_{i,r}^2 (z_{i,l} - z_{i,r}) \end{bmatrix}.
\]
(5.29)
is well-balanced.

**Proof.** Suppose that \(w = h + z\) is constant at time \(t\), and \(u = 0\). Since \(w_{i,r} = w_{i+1,l}\),
\[
h_{i-\frac{1}{2}^+} = \max(0, w_{i,r} - z_{i+\frac{1}{2}}) = \max(0, w_{i+1,l} - z_{i+\frac{1}{2}}) = h_{i+\frac{1}{2}^-}
\]
where \(z_{i+1/2} = \max(z_{i,r}, z_{i+1,l})\), and since \(w_{i,r} = w_{i+1,l} = 0\), it is obvious that
\[
Q_{i,r} = Q_{i+1,l}.
\]

Since the values \(Q_{i,r}\) and \(Q_{i+1,l}\) are now equals and \(u = 0\), the numerical fluxes \(F_{i+1/2}\) and \(F_{i-1/2}\) reduce to the hydrostatic pressure. Substituting this information in (5.27) and (5.25), it is obtained that
\[
\mathcal{F}^r(Q_{i,r}, Q_{i+1,l}, z_{i,r}, z_{i+1,l}) = \begin{bmatrix} 0 \\ \frac{g}{2} h_{i,r}^2 \end{bmatrix},
\]
(5.30)
and
\[
\mathcal{F}^i(Q_{i-1,r}, Q_{i,l}, z_{i-1,r}, z_{i,l}) = \begin{bmatrix} 0 \\
\frac{g}{2} h_{i,l}^2 \end{bmatrix}.
\] (5.31)

Substituting (5.30) and (5.31) in (5.26) implies that
\[
\frac{d}{dt} h_i(t) = 0
\] (5.32)

and
\[
\frac{d}{dt} (h_i u_i) = -\frac{1}{\Delta x} \left( \frac{g}{2} h_{i,r}^2 - \frac{g}{2} h_{i,l}^2 - \frac{g}{2} (h_{i,l} + h_{i,r}) (z_{i,l} - z_{i,r}) \right)
- \frac{1}{\Delta x} \left( \frac{h_{i,r}^2 - h_{i,l}^2}{2} - \frac{h_{i,l} + h_{i,r}}{2} ((w_{i,l} - h_{i,l}) - (w_{i,r} - h_{i,r})) \right)
- \frac{1}{\Delta x} \left( - \frac{h_{i,l} + h_{i,r}}{2} (w_{i,l} - w_{i,r}) \right).
\]

Since \(w_{i,l} = w_{i,r} = w\), it is clear that
\[
\frac{d}{dt} (h_i u_i)(t) = 0.
\] (5.33)

Since (5.32) and (5.33) hold,
\[
\frac{d}{dt} Q_i(t) = 0.
\]

Therefore, \(Q_i(t)\) remains constant, i.e. the second order semidiscrete scheme preserves the stationary of lake at rest. \(\square\)

### 5.5 Higher Order Well-Balancing

Starting from a given first order accurate well-balanced scheme in Section 5.2 and 5.3, an extension to second order has been presented in Section 5.4. If one wants to track small waves over long periods of time, high order accurate well-balanced schemes are needed. In this section, higher order schemes are derived by extending the previous first and second order schemes, in particular by finding a quadrature rule for the source term which is both accurate and well-balanced. The derivation is based on the work of Noelle, et al. in [50].

Consider a polynomial reconstruction of \(Q\) of any desired order of accuracy. Let \(Q_{i,r}\) and \(Q_{i+1,l}\) be the left and the right values of piecewise polynomial reconstruction at interface \(x_{i-1/2}^+\) and \(x_{i+1/2}^-\) respectively. Define the hydrostatic
5.5. **HIGHER ORDER WELL-BALANCING**

reconstruction

\[ h_{i+\frac{1}{2}^-} = \max(0, h_{i,r} + z_{i,r} - z_{i+\frac{1}{2}}), \quad h_{i+\frac{1}{2}^+} = \max(0, h_{i+1,l} + z_{i+1,l} - z_{i+\frac{1}{2}}), \]

where

\[ z_{i+\frac{1}{2}} = \max(z_{i,r}, z_{i+1,l}), \]
as before, then set

\[ Q_{i+\frac{1}{2}^-} = \begin{bmatrix} h_{i+1/2-} \\ h_{i+1/2-} u_{i,r} \end{bmatrix}, \quad Q_{i+\frac{1}{2}^+} = \begin{bmatrix} h_{i+1/2+} \\ h_{i+1/2+} u_{i+1,l} \end{bmatrix}. \]

Next, define the right and the left interface fluxes as in (5.24) and (5.25). The main problem is then to define a high order well-balanced quadrature of the source term \( S^{(j)} \), where the index \( j \) represents the order of the numerical source term. As Noelle, et al. propose, the problem can be solved by numerical extrapolation.

Note that

\[ S := -\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} gh z_x \, dx. \]

Subdividing each cell into \( N \) subcells and applying the cell-centered quadrature to all subcells gives the quadrature \( S^N \),

\[ S := -g \sum_{j=1}^{N} \frac{h_{j-1} + h_j}{2} (z_{j-1} - z_j) \approx S, \]

where \( z_j = z(x_{i-1/2} + j \Delta x/N) \), and \( h_j = h(x_{i-1/2} + j \Delta x/N) \) are local values of the reconstruction at the interfaces of the subcells. For the lake at rest, it is clear that \( z_{j-1} - z_j = h_j - h_{j-1} \). Therefore, the source term reduces to

\[ S^N = -g \sum_{j=1}^{N} (h_{j-1} + h_j)(h_j - h_{j-1}) \]

\[ = -g (h_N^2 - h_0^2) \]

\[ = -g (h_{i,r}^2 - h_{i,l}^2). \]  

(5.34)

Since the cell-centered quadrature of source term (5.22) leads to second order accuracy and is well-balanced, then the source term reconstruction (5.34) is still second order accuracy and is also well-balanced.
As stated above, numerical extrapolation can be applied to get higher order of accuracy. Since the quadrature here is symmetric and second order, there exists an asymptotic expansion of the form

\[ S^N = S + c_1 \left( \frac{\Delta x}{N} \right)^2 + c_2 \left( \frac{\Delta x}{N} \right)^4 + \cdots. \]

To compute \( S \) with any order of accuracy, the \( S^N \) can be combined for different values of \( N \). For example, a source term of order four can be found by

\[ 4S^2 - S^1 = S + c_2 (\Delta x)^4 + \cdots, \]

and according to (5.34)

\[ S^1 = S^2 = -\frac{g}{2} (h_{i,r}^2 - h_{i,l}^2) \]

for the lake at rest. As a result, a source term of order four for the lake at rest can be expressed as

\[ S_{c,i}^{(4)} = \frac{4S^2 - S^1}{3} = -\frac{g}{2} (h_{i,r}^2 - h_{i,l}^2). \]

### 5.6 Quantities Reconstruction

Recall the quantities involved in the numerical methods presented above. There are five quantities of interest, namely water depth \( h \), bed elevation \( z \), velocity \( u \), stage \( w = h + z \), and momentum \( p = uh \).

Central-upwind methods with reconstructions on stage and momentum have been tested in [27], but they may result in oscillations over dry-bed areas, such as for solving parabolic canal and dry-dam problem. To alleviate those oscillations, adding the well-balanced terms in the flux calculation will be attempted, and the simulation results will be presented in Chapter 6. In this case, the values of \( z \) are defined at the centroids and vertices of cells, then the algorithm for computing the fluxes each time step is as follows.

1. Find \( w \) and \( p \) at the centroids.
2. Extrapolate \( w \) and \( p \) at the centroids to the vertices.
3. Calculate \( h = w - z \) and \( u = p/h \) at the vertices.
4. Compute the fluxes with well-balanced terms.
Another attempt, which will be simulated in Chapter 7, is by doing reconstructions on the stage and velocity with the well-balanced terms in the flux calculation. Note that \( z \) is again defined at the centroids and vertices, then the corresponding algorithm is as follows.

1. Find \( w \) and \( p \) at the centroids.
2. Calculate \( h = w - z \) and \( u = p/h \) at the centroids.
3. Extrapolate \( w \) and \( u \) to the vertices.
4. Calculate \( h = w - z \) and \( p = uh \) at the vertices.
5. Compute the fluxes with well-balanced terms.

5.7 Concluding Remarks

In this chapter, the well-balanced technique has been presented. The technique comes from the steady state conditions of a lake at rest. In addition, high order well-balancing can be obtained by numerical extrapolation which is applied to some lower orders of accuracy.
CHAPTER 5. WELL-BALANCED FINITE VOLUME METHODS
Chapter 6

Well-Balanced Methods with Stage and Momentum Reconstructions

This chapter is devoted to the validation of well-balanced finite volume methods with stage and momentum ($w$-$uh$) reconstructions. The finite volume schemes being used are the central-upwind schemes developed by Kurganov, et al. [35, 36] as presented at the end of Section 4.1, and the well-balanced technique is based on the work of Audusse, et al. [2] and Noelle, et al. [50] as presented in Chapter 5. All numerical tests in this chapter use $g = 9.81$ m/s$^2$ and $\epsilon = 10^{-12}$ and $h_{\text{min}} = 10^{-3}$ m, unless otherwise explicitly stated. Here, $\epsilon$ is a parameter of velocity stabilization as described in Section 4.2, while $h_{\text{min}}$ is a parameter that controls the water depth for flux calculations. In the flux calculations, any water depth less than $h_{\text{min}}$ is considered zero. In addition, any numerical error (defined in Section 4.6.3) is presented in two decimal-places, where the errors in water depth $h$, momentum $uh$, and velocity $u$ are denoted by $E_h$, $E_{uh}$, and $E_u$ respectively.

6.1 Test 1: Dam-Break Problems

In this section, simulations for the well-balanced central-upwind methods with $w$-$uh$ reconstructions are presented. While Jakeman [27] has simulated one dam at a time using central-upwind methods without well-balanced terms in the flux calculation, two dams at a time are simulated in this thesis. Two dams here
CHAPTER 6. STAGE AND MOMENTUM RECONSTRUCTIONS

means that there is a dam-wall on the right side of a reservoir and there is another dam-wall on the left as well. The final time of the tests is set to be 20 s.

Consider two-dam-break problem with the following initial conditions

\[ u(x,0) = 0 \quad \text{and} \quad h(x,0) = \begin{cases} 
0 & \text{if } x < 500 \\
10 & \text{if } 500 < x < 1500 \\
5 & \text{if } 1500 < x < 2000 
\end{cases} \] (6.1)

The initial conditions tell that there are two dams, located at \( x = 500 \) m and \( x = 1500 \) m from the point of origin. At time \( t = 0 \), sudden destruction happens on both dam walls. The simulation is then done to illustrate the subsequent motion of the water at any point \( x \) and at any time \( t \).

6.1.1 Accuracy

Usually, the larger number of cells (that is the finer the grid) leads to the closer numerical solution to the exact solution. This is illustrated by Figures 6.1 and 6.2. Figure 6.1 shows the water profile after 20 seconds of the dam destruction using 100 cells with first-order spatial discretisation and first-order time-stepping. In comparison, Figure 6.2 shows the water profile after 20 seconds of the dam destruction using 400 cells with the same order of spatial and temporal discretisation. It is obvious that the numerical solution with 400 cells is closer to the exact solution. However, it should be noted that if the grid is too fine, then the central-upwind methods may result in a large error (consult [27]).

The previous two simulations result in a high diffusion around discontinuities and corners. In addition, the numerical methods being used cannot solve the problems around dry-regions. Methods with second order spatial discretisation can approximate the exact solution better than those with first order in general. They are able to reduce the diffusion for the finite water depth. However, they produce oscillations around dry regions as viewed in Figures 6.3 and 6.4. Figure 6.3 illustrates the water profile after 20 seconds of the dam destruction using 400 cells with second-order spatial discretisation and first-order time-stepping and minmod slope limiting. This results in \( E_h = 2.65 \cdot 10^{-2} \), \( E_{uh} = 0.22 \), and \( E_u = 0.23 \). Figure 6.4 shows the water profile after the same time-period of the dam destruction using the same number of cells with the same order spatial
6.1. TEST 1: DAM-BREAK PROBLEMS

Figure 6.1: Well-balanced central-upwind with \( w-uh \) reconstructions using 100 cells, first-order spatial discretisation, and first-order time-stepping.

Figure 6.2: Well-balanced central-upwind with \( w-uh \) reconstructions using 400 cells, first-order spatial discretisation, and first-order time-stepping.
Figure 6.3: Well-balanced central-upwind with \( w-uh \) reconstructions using 400 cells, second-order spatial discretisation, first-order time-stepping, and minmod limiter.

Figure 6.4: Well-balanced central-upwind with \( w-uh \) reconstructions using 400 cells, second-order spatial discretisation, second-order time-stepping, and minmod limiter.
discretisation and the same slope limiting, but the time-stepping is set to be second-order. This results in slightly larger errors $E_h = 2.83 \cdot 10^{-2}$, $E_{uh} = 0.26$, and $E_u = 0.60$. It can be seen that the error produced by methods using second order temporal discretisation is larger. This is not the case if finer grids are applied, because the errors in depth and momentum produced by methods with second order temporal evolution will be smaller than the errors produced by those with first order time-stepping, but the error in velocity will be larger. For example when the computational domain is divided into 1,600 cells, the absolute error in the water depth is $7.28 \cdot 10^{-3}$ and $6.84 \cdot 10^{-3}$ for first and second order time-stepping respectively, the absolute error in the momentum is $6.63 \cdot 10^{-2}$ and $6.58 \cdot 10^{-2}$, but the absolute error in the velocity is 0.27 and 0.51.

Another slope limiter can be applied to determine whether the methods yield a better result. When the van Leer limiter is used in this method to solve the dam-break problem, it is found that there is not much improvement, as shown by Figures 6.5 and 6.6. Figure 6.5 presents the water profile after 20 seconds of the dam destruction using 400 cells with second-order spatial discretisation, first-order time-stepping and the van Leer limiter. The errors in water depth, momentum and velocity are $E_h = 2.34 \cdot 10^{-2}$, $E_{uh} = 0.21$, and $E_u = 0.94$ respectively. Figure 6.6 shows the water profile after 20 seconds of the dam destruction using 400 cells with second-order spatial discretisation and second-order time-stepping and the van Leer slope limiting. The errors in depth, momentum, and velocity are $E_h = 2.48 \cdot 10^{-2}$, $E_{uh} = 0.22$ and $E_u = 0.17$. The error produced by this method using second order time-stepping is smaller than the method using first order time-stepping, since then According to those Figures, these other two simulations still produce oscillations around dry-regions.

Furthermore, central-upwind methods with $w$-$uh$ reconstructions can only work well with particular types of limiter, such as van Leer, minmod, and van Albada. They generate worse oscillations and can even fail to solve the problem if second order spatial discretisation, second order temporal discretisation, and either minmod-Kurganov, superbee, or pyvolution limiter are applied together. This is shown by Figure 6.7 for example. When the well-balanced central-upwind with $w$-$uh$ reconstructions using 400 cells, second-order spatial discretisation, second-order time-stepping, and superbee limiter are applied with $\epsilon = 10^{-6}$, the errors in water depth, momentum and velocity are $E_h = 0.13$, $E_{uh} = 1.06$, and
Figure 6.5: Well-balanced central-upwind with \( w-uh \) reconstructions using 400 cells, second-order spatial discretisation, first-order time-stepping, and van Leer limiter

Figure 6.6: Well-balanced central-upwind with \( w-uh \) reconstructions using 400 cells, second-order spatial discretisation, second-order time-stepping, and van Leer limiter
6.1. TEST 1: DAM-BREAK PROBLEMS

Figure 6.7: Well-balanced central-upwind with $w$-$uh$ reconstructions using 400 cells, second-order spatial discretisation, second-order time-stepping, and superbee limiter.

Figure 6.8: Well-balanced central-upwind with $w$-$uh$ reconstructions using 800 cells, second-order spatial discretisation, second-order time-stepping, van Leer limiter and 1 micrometre allowed minimum water depth.
The oscillations can be reduced by setting $h_{\text{min}}$ into smaller value, by taking smaller cells, and by using some particular types of limiter (that is one of van Leer, minmod, and van Albada). For example, by setting $h_{\text{min}} = 10^{-6}$ m, Figure 6.8 shows that there is not oscillations in the numerical solution and this produces smoother solution with $E_h = 1.13 \cdot 10^{-2}$, $E_{uh} = 0.10$, and $E_u = 1.20$ for $\epsilon = 10^{-12}$.

However, setting $h_{\text{min}}$ into a very small value will make the computation very slow since this makes the number of computations increase. Once again, the methods do not match with either Kurganov’s minmod, superbbee and pyvolution, and can only work with minmod, van Leer, and van Albada slope limiter.

Table 6.1 and Table 6.2 present error comparisons resulting from well-balanced central-upwind methods for solving dam-break problems including dry-dam and finite water depth problems, using 100 and 400 cells respectively. The simulations are run with $h_{\text{min}} = 10^{-3}$, and $\epsilon = 10^{-6}$. The value of $\epsilon$ is chosen to be $10^{-6}$ in order to get a faster computation and to allow all limiters to be applied. It should be noted that the last columns in the tables show the running time $t$ needed in one simulation only, and it is the running time which covers the total time needed to compute both the analytical and numerical solutions.

Table 6.3 shows the error comparisons using finer grids, 1,600 cells, and the other parameters are set to be the same as those done in the previous simulation. The methods using minmod-Kurganov, pyvolution, and superbbee limiter are not tested in this case, since it is clear from the previous two tables they are not efficient (they take between 30 and 300 times longer than those needed by the methods with minmod, van Albada, and van Leer limiter in the simulations presented in Tables 6.1 and 6.2), and produce much larger error.

According to Table 6.1, 6.2, and 6.3, methods with the van Leer limiter result in the smallest error and need a relatively short running time. It can be inferred that van Leer limiter works better than other limiters for solving the dam-break problems using well-balanced central-upwind methods with $w-u\n$ reconstructions. This is the same inference as when well-balanced terms are not added in the flux computations, as simulated in [27]. In addition, it can be seen that the results presented in this section are better than the results by central-upwind methods.
### 6.1. TEST 1: DAM-BREAK PROBLEMS

<table>
<thead>
<tr>
<th>Slope limiter</th>
<th>$E_h$</th>
<th>$E_{uh}$</th>
<th>$E_u$</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minmod</td>
<td>0.11</td>
<td>0.98</td>
<td>0.80</td>
<td>1.62 s</td>
</tr>
<tr>
<td>Minmod Kurganov</td>
<td>0.96</td>
<td>11.38</td>
<td>3.93</td>
<td>204.84 s</td>
</tr>
<tr>
<td>Pyvolution</td>
<td>0.79</td>
<td>8.77</td>
<td>5.44</td>
<td>46.41 s</td>
</tr>
<tr>
<td>Superbee</td>
<td>0.29</td>
<td>2.80</td>
<td>2.90</td>
<td>250.69 s</td>
</tr>
<tr>
<td>van Albada</td>
<td>0.10</td>
<td>0.88</td>
<td>0.73</td>
<td>1.65 s</td>
</tr>
<tr>
<td>van Leer</td>
<td>0.10</td>
<td>0.84</td>
<td>0.78</td>
<td>1.90 s</td>
</tr>
</tbody>
</table>

Table 6.1: Absolute errors in water depth, momentum and velocity using 100 cells, second order spatial, second order temporal discretisations and six different limiters

<table>
<thead>
<tr>
<th>Slope limiter</th>
<th>$E_h$</th>
<th>$E_{uh}$</th>
<th>$E_u$</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minmod</td>
<td>$2.83 \cdot 10^{-2}$</td>
<td>0.26</td>
<td>0.61</td>
<td>10.84 s</td>
</tr>
<tr>
<td>Minmod Kurganov</td>
<td>$2.59 \cdot 10^{-1}$</td>
<td>2.34</td>
<td>1.59</td>
<td>3184.18 s</td>
</tr>
<tr>
<td>Pyvolution</td>
<td>$2.94 \cdot 10^{-1}$</td>
<td>2.80</td>
<td>1.75</td>
<td>283.17 s</td>
</tr>
<tr>
<td>Superbee</td>
<td>$1.28 \cdot 10^{-3}$</td>
<td>1.06</td>
<td>1.76</td>
<td>450.87 s</td>
</tr>
<tr>
<td>van Albada</td>
<td>$2.57 \cdot 10^{-2}$</td>
<td>0.23</td>
<td>0.56</td>
<td>10.92 s</td>
</tr>
<tr>
<td>van Leer</td>
<td>$2.48 \cdot 10^{-2}$</td>
<td>0.22</td>
<td>0.16</td>
<td>13.85 s</td>
</tr>
</tbody>
</table>

Table 6.2: Absolute errors in water depth, momentum and velocity using 400 cells, second order spatial, second order temporal discretisations and six different limiters

<table>
<thead>
<tr>
<th>Slope limiter</th>
<th>$E_h$</th>
<th>$E_{uh}$</th>
<th>$E_u$</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minmod</td>
<td>$6.84 \cdot 10^{-3}$</td>
<td>6.58 $\cdot 10^{-2}$</td>
<td>0.51</td>
<td>155.27 s</td>
</tr>
<tr>
<td>Minmod Kurganov</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Pyvolution</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Superbee</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>van Albada</td>
<td>$6.29 \cdot 10^{-3}$</td>
<td>5.96 $\cdot 10^{-2}$</td>
<td>0.49</td>
<td>156.20 s</td>
</tr>
<tr>
<td>van Leer</td>
<td>$5.78 \cdot 10^{-3}$</td>
<td>5.32 $\cdot 10^{-2}$</td>
<td>0.39</td>
<td>177.03 s</td>
</tr>
</tbody>
</table>

Table 6.3: Absolute errors in water depth, momentum and velocity using 1,600 cells, second order spatial, second order temporal discretisations and six different limiters
According to the presentation above, the following can be inferred. If first order spatial discretisation is used, it is found that the smaller the size of cells (that is the more number of cells), the closer the numerical solution to the exact one. However, this discretisation leads to diffusion near discontinuities generally. A better approximation can be done by taking second order spatial discretisation together with either first order or second order temporal discretisation. Applying second order spatial discretisation often results in oscillations at some positions near discontinuities and corners, but using well-balanced methods the oscillation can be eliminated by applying a very small value of $\epsilon$ and $h_{\min}$ together with one of van Leer, minmod, and van Albada limiter. The slope limiter has an important role in facing the dry-region, so a particular type of limiter should be carefully chosen.

### 6.1.2 Efficiency

To investigate the speed of the methods in doing the computations, the average time of 10 runs per grid size is taken. Table 6.4 shows the results for three different cases of discretisations, namely first order spatial and temporal discretisations, second order spatial and first order temporal discretisations, and second order spatial and temporal discretisations. For second order spatial discretisation, van Leer limiter is applied.

Based on Table 6.4, it can be conjectured that the algorithm of well-balanced central-upwind schemes is nearly an $O(n^2)$. This is reasonable since the algorithm of central-upwind schemes without well-balanced terms is predicted an $O(n^2)$.

<table>
<thead>
<tr>
<th>Number of cells</th>
<th>First-Order</th>
<th>Second/First Order</th>
<th>Second-Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>0.04</td>
<td>0.16</td>
<td>0.24</td>
</tr>
<tr>
<td>100</td>
<td>0.13</td>
<td>0.63</td>
<td>0.85</td>
</tr>
<tr>
<td>200</td>
<td>0.47</td>
<td>2.63</td>
<td>3.17</td>
</tr>
<tr>
<td>400</td>
<td>1.86</td>
<td>9.29</td>
<td>12.19</td>
</tr>
<tr>
<td>800</td>
<td>7.71</td>
<td>34.48</td>
<td>46.70</td>
</tr>
</tbody>
</table>

Table 6.4: Average time of 10 runs taken to simulate the dam-break problem without well-balanced terms, by excluding the methods with minmod-Kurganov, pyvolution, and superbee limiter.
6.2. **TEST 2: OSCILLATIONS IN A PARABOLIC CANAL**

The computation order of the well-balanced central-upwind scheme is slightly lower because of the well-balanced terms which make more computations in the algorithm. Furthermore, it can be inferred from the table that increasing the accuracy leads to decreasing the efficiency. This kind of trade-off is usual in computations and is not surprising.

### 6.2 Test 2: Oscillations in a Parabolic Canal

Consider the oscillating flow in a canal with a maximum equilibrium water depth of $D_0 = 10$ m and an equilibrium horizontal water surface length of $2L = 5,000$ m. Then suppose that the amplitude $A$ is given by $A = L/2$. The initial water profile can be illustrated by Figures 3.4 and 6.9, where the water bed, the initial velocity, and the initial water stage are

$$z = \frac{D_0}{L^2} x^2, \quad u = 0, \quad \text{and} \quad w = D_0 + \frac{2AD_0}{L^2} [x - A/2] \quad (6.2)$$

respectively. Following the analytical solution presented in Section 3.2, the angular frequency of oscillation and the period are respectively given by

$$\omega = \sqrt{\frac{2gD_0}{L}} \quad \text{and} \quad T = \frac{2\pi}{\omega}. \quad (6.3)$$

It is known that the well-balanced central-upwind methods with $w-uh$ reconstructions cannot solve the oscillating planar flow in a frictionless parabolic canal. Figure 6.10 shows a result of the well-balanced central-upwind method with $w-uh$ reconstructions for solving this problem using second order spatial, second order temporal discretisation, and the van Leer limiter. The resulting errors in depth, momentum, and velocity are $E_h = 0.89$, $E_{uh} = 11.94$, and $E_u = 1.95$ respectively. Obviously, these errors are much larger than the errors results from a central-upwind method without well-balanced terms as can be compared with those presented in Table 6.6 in [27]. In this test, $\epsilon = 10^{-6}$ is taken because when it is too small, such as $\epsilon = 10^{-12}$, the methods may fail.

This phenomenon is very different from that results from the central-upwind methods without well-balanced terms simulated in [27]. This suggests that flux functions with the well-balanced terms are not appropriate to solve oscillation in a parabolic canal. This also suggests that well-balanced central-upwind methods
CHAPTER 6. STAGE AND MOMENTUM RECONSTRUCTIONS

Figure 6.9: Initial water profile in a parabolic canal

Figure 6.10: Well-balanced central-upwind with \(w-uh\) reconstructions using 400 cells, second-order spatial discretisation, second-order time-stepping, and van Leer limiter at \(t = T/4\)
6.3. TEST 3: STEADY FLOW

Consider the steady flow problem presented in Section 3.3. Let a horizontal frictionless channel which has length of 50 m containing a parabolic obstruction

Figure 6.11: Central-upwind without well-balanced terms using \( w-uh \) reconstructions, 400 cells, second-order spatial discretisation, second-order time-stepping, and van Leer limiter at \( t = T/4 \)

with \( w-uh \) reconstructions do not work with unsteady flows with nonzero bottom topography.

It is interesting that if the well-balanced terms are not applied in the numerical flux function, the errors in water depth and momentum decrease dramatically. However, there exists a very high velocity in the interface between wet and dry region on the right hand side as the water flows to the left. This means that the methods without well-balanced terms cannot solve the problem on drying region in general, even though it appears that they lead to a reasonable result for the water depth and momentum as illustrated by Figure 6.11. The simulation shown by Figure 6.11 yield errors in water depth, momentum and velocity of \( E_h = 2.03 \times 10^{-3} \), \( E_{uh} = 2.03 \times 10^{-2} \), and \( E_u = 0.19 \).

6.3 Test 3: Steady Flow

Consider the steady flow problem presented in Section 3.3. Let a horizontal frictionless channel which has length of 50 m containing a parabolic obstruction
centred at $x_{\text{max}} = 25$ m with a maximum elevation of $z_{\text{max}} = 0.2$ m and width of $2b$, where $b = 4$ m be given. In other words, the water bed is given by

$$z(x) = \begin{cases} 
    z_{\text{max}} \left[ 1 - \left( \frac{x - x_{\text{max}}}{b} \right)^2 \right] & \text{if } x_{\text{max}} - b < x < x_{\text{max}} + b \\
    0 & \text{otherwise}
\end{cases}$$

(6.4)

Suppose that the water has a discharge of $q = 0.3$ m$^2$/s flowing in the channel, and the water depth on the left and on the right boundaries have the same level which is $h_0 = h_1 = 0.5$ m. The value of $q$, $h_0$, and $h_1$ lead to the velocity value of $u_0 = u_1 = 0.6$ m/s at the boundaries. Following Jakeman [27], here the final time is set to be 25 seconds instead of 20 seconds in order to easily compare the errors resulting from the well-balanced methods and those resulting from central-upwind methods without well-balanced terms.

Figure 6.12 shows the result of well-balanced central-upwind method for solving the steady flow. Here the simulation is using 400 cells, second-order spatial discretisation, second-order time-stepping, and van Leer limiter where $\epsilon = 10^{-6}$. It is found that the well-balanced methods lead to smaller errors than the methods without well-balanced terms. Jakeman [27] has simulated this problem using central-upwind methods without well-balanced terms and has got the maximum momentum error which is greater than $1.5 \cdot 10^{-3}$. However, using the well-balanced methods for solving this steady flow problem, the maximum momentum error can be reduced to $5.0 \cdot 10^{-4}$. The $L^1$ errors in depth, momentum, and velocity for the steady flow test shown by Figure 6.12 are $E_h = 5.90 \cdot 10^{-6}$, $E_{uh} = 7.28 \cdot 10^{-6}$, and $5.75 \cdot 10^{-5}$. Obviously, well-balanced central-upwind methods with $w$-$uh$ reconstructions yield a better numerical solution to steady flow problem than those resulting from methods without well-balanced terms.

Figure 6.13 is another result using the same specification as before, but the number of cells is set to be 1,600. This simulation has $E_h = 6.36 \cdot 10^{-7}$, $E_{uh} = 7.94 \cdot 10^{-7}$, and $E_u = 6.94 \cdot 10^{-6}$ for the errors in water depth, momentum, and velocity.

### 6.4 Concluding Remarks

The well-balanced central-upwind methods with $w$-$uh$ reconstructions have been simulated in this chapter. It is found that these methods solve the dam-break
6.4. CONCLUDING REMARKS

Figure 6.12: Steady flow using well-balanced central-upwind with \( w-uh \) reconstructions, and 400 cells

Figure 6.13: Steady flow using well-balanced central-upwind with \( w-uh \) reconstructions, and 1600 cells
problem and steady flow nicely, but fail to solve the oscillations in a parabolic canal. Therefore, it can be inferred that these methods work well on steady flow in general, but may fail to solve a unsteady flow with non-horizontal bottom topography. In addition, the performance of the methods rely much on the slope limiter being used, and it is obtained that minmod, van Albada, and van Leer limiters work better than minmod-Kurganov, pyvolution, and superbee.
Chapter 7

Well-Balanced Methods with Stage and Velocity Reconstructions

Well-balanced central-upwind methods with stage and momentum \((w-uh)\) reconstructions fail to solve appropriately the oscillations in a parabolic canal. They also fail to solve the dam-break problems when minmod-Kurganov, superbee and pyvolution are used to limit the water depth. This chapter presents the simulation results of well-balanced central-upwind methods with stage and velocity \((w-u)\) reconstructions. Here unless otherwise stated explicitly, \(g = 9.81\, \text{m/s}^2\), \(\epsilon = 10^{-12}\), and \(h_{\text{min}} = 10^{-3}\, \text{m}\). The simulations are done with second order spatial and second order temporal discretisations.

7.1 Test 1: Dam-Break Problems

Following the presentation in Chapter 6, dam-break problems will be used to test the accuracy and efficiency of the methods. The standard spatial discretisation which is used here is 400 cells in 2,000 m spatial domain, and the final time is set to be \(t = 20\, \text{s}\).
CHAPTER 7. STAGE AND VELOCITY RECONSTRUCTIONS

Figure 7.1: Well-balanced central-upwind with $w-u$ reconstructions using 400 cells and van Leer limiter

7.1.1 Accuracy

Figure 7.1 shows the water profile using well-balanced central-upwind methods with $w-u$ reconstructions with van Leer limiter and 400 cells. It is obtained that $E_h = 2.44 \times 10^{-2}$, $E_{uh} = 0.21$, and $0.60$.

Table 7.1, 7.2 and 7.3 present error comparisons produced by well-balanced central-upwind methods with $w-u$ reconstructions for solving dam-break problems in general including dry-dam and finite water depth problems. According to those tables, methods using superbee limiter yield small errors in water depth and momentum, and also a small error in velocity in general. Methods with minmod limiter lead to larger errors. It should be noted that the last columns in these two tables present the running time $t$ needed in one simulation only, and the running time covers the total time needed to compute both the analytical and numerical solutions.

To get a better result, the size of the cells can be set into smaller values without resulting small time-steps. Figure 7.2 shows the water profile of dam-break problem with 3200 cells, which means that the size of the cells is $0.625m$ using superbee limiter. The errors resulting are $E_h = 2.87 \times 10^{-3}$, $E_{uh} = 2.50 \times 10^{-2}$,
7.1. TEST 1: DAM-BREAK PROBLEMS

<table>
<thead>
<tr>
<th>Slope limiter</th>
<th>$E_h$</th>
<th>$E_{uh}$</th>
<th>$E_u$</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minmod</td>
<td>$1.11 \cdot 10^{-1}$</td>
<td>1.02</td>
<td>1.00</td>
<td>2.23 s</td>
</tr>
<tr>
<td>Minmod-Kurganov</td>
<td>$9.37 \cdot 10^{-2}$</td>
<td>0.79</td>
<td>1.19</td>
<td>6.57 s</td>
</tr>
<tr>
<td>Pyvolution</td>
<td>$9.37 \cdot 10^{-2}$</td>
<td>0.79</td>
<td>1.19</td>
<td>2.96 s</td>
</tr>
<tr>
<td>Superbee</td>
<td>$9.07 \cdot 10^{-2}$</td>
<td>0.80</td>
<td>1.21</td>
<td>2.39 s</td>
</tr>
<tr>
<td>van Albada</td>
<td>$1.04 \cdot 10^{-1}$</td>
<td>0.92</td>
<td>1.09</td>
<td>2.39 s</td>
</tr>
<tr>
<td>van Leer</td>
<td>$9.65 \cdot 10^{-2}$</td>
<td>0.83</td>
<td>1.05</td>
<td>2.36 s</td>
</tr>
</tbody>
</table>

Table 7.1: Absolute errors in water depth, momentum and velocity using well-balanced central-upwind with $w-u$ reconstructions using 100 cells and six different slope limiters

<table>
<thead>
<tr>
<th>Slope limiter</th>
<th>$E_h$</th>
<th>$E_{uh}$</th>
<th>$E_u$</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minmod</td>
<td>$2.88 \cdot 10^{-2}$</td>
<td>0.26</td>
<td>0.69</td>
<td>17.10 s</td>
</tr>
<tr>
<td>Minmod-Kurganov</td>
<td>$2.37 \cdot 10^{-2}$</td>
<td>0.20</td>
<td>0.56</td>
<td>96.77 s</td>
</tr>
<tr>
<td>Pyvolution</td>
<td>$2.37 \cdot 10^{-2}$</td>
<td>0.20</td>
<td>0.56</td>
<td>27.05 s</td>
</tr>
<tr>
<td>Superbee</td>
<td>$2.31 \cdot 10^{-2}$</td>
<td>0.20</td>
<td>0.53</td>
<td>20.33 s</td>
</tr>
<tr>
<td>van Albada</td>
<td>$2.59 \cdot 10^{-2}$</td>
<td>0.23</td>
<td>0.64</td>
<td>17.39 s</td>
</tr>
<tr>
<td>van Leer</td>
<td>$2.44 \cdot 10^{-2}$</td>
<td>0.21</td>
<td>0.60</td>
<td>17.50 s</td>
</tr>
</tbody>
</table>

Table 7.2: Absolute errors in water depth, momentum and velocity using well-balanced central-upwind with $w-u$ reconstructions using 400 cells and six different slope limiters

<table>
<thead>
<tr>
<th>Slope Limiter</th>
<th>$E_h$</th>
<th>$E_{uh}$</th>
<th>$E_u$</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minmod</td>
<td>$6.89 \cdot 10^{-3}$</td>
<td>$6.49 \cdot 10^{-2}$</td>
<td>0.44</td>
<td>269.52 s</td>
</tr>
<tr>
<td>Minmod-Kurganov</td>
<td>$5.39 \cdot 10^{-3}$</td>
<td>$4.66 \cdot 10^{-2}$</td>
<td>0.34</td>
<td>1655.41 s</td>
</tr>
<tr>
<td>Pyvolution</td>
<td>$5.39 \cdot 10^{-3}$</td>
<td>$4.66 \cdot 10^{-2}$</td>
<td>0.34</td>
<td>433.94 s</td>
</tr>
<tr>
<td>Superbee</td>
<td>$5.15 \cdot 10^{-3}$</td>
<td>$4.48 \cdot 10^{-2}$</td>
<td>0.20</td>
<td>318.21 s</td>
</tr>
<tr>
<td>van Albada</td>
<td>$6.07 \cdot 10^{-3}$</td>
<td>$5.46 \cdot 10^{-2}$</td>
<td>0.41</td>
<td>267.96 s</td>
</tr>
<tr>
<td>van Leer</td>
<td>$5.64 \cdot 10^{-3}$</td>
<td>$4.96 \cdot 10^{-2}$</td>
<td>0.37</td>
<td>272.91 s</td>
</tr>
</tbody>
</table>

Table 7.3: Absolute errors in water depth, momentum and velocity using well-balanced central-upwind with $w-u$ reconstructions using 1600 cells and six different slope limiters
and $E_u = 0.10$. Obviously there is not any oscillation in this case, but there exists a gap in velocity around the dry-region.

### 7.1.2 Efficiency

Similar to the investigation of the speed of the computations in Chapter 6, the average time of 10 runs per grid size is taken. Table 7.4 shows the results for three different cases of discretisations. For second order spatial discretisation, van Leer limiter is applied. It should be noted that the running-time presented in this table is only for the numerical solution, unlike those presented in Table 7.2 and 7.3 which covers both analytical and numerical solutions.

It can be seen from Table 7.4 that when the cell size is halved, the running time of all three models increases by a factor of four. According to this fact, it can be conjectured that the algorithm of a well-balanced central-upwind scheme with $w-u$ reconstructions is an $O(n^2)$ algorithm. To prove whether this conjecture is true, algorithm analysis needs to be carried out, but it is not done here since it is beyond the scope of this thesis.
7.2. **TEST 2: OSCILLATIONS IN A PARABOLIC CANAL**

<table>
<thead>
<tr>
<th>Number of cells</th>
<th>First-Order</th>
<th>Second/First Order</th>
<th>Second-Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>0.04</td>
<td>0.12</td>
<td>0.20</td>
</tr>
<tr>
<td>100</td>
<td>0.09</td>
<td>0.43</td>
<td>0.77</td>
</tr>
<tr>
<td>200</td>
<td>0.32</td>
<td>1.82</td>
<td>3.19</td>
</tr>
<tr>
<td>400</td>
<td>1.20</td>
<td>7.94</td>
<td>13.26</td>
</tr>
<tr>
<td>800</td>
<td>4.82</td>
<td>34.60</td>
<td>54.82</td>
</tr>
</tbody>
</table>

Table 7.4: Average time of 10 runs taken to simulate the dam-break problem with \( w-u \) reconstructions

<table>
<thead>
<tr>
<th>Final time</th>
<th>( E_h )</th>
<th>( E_{uh} )</th>
<th>( E_u )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T/4 )</td>
<td>( 5.35 \cdot 10^{-4} )</td>
<td>( 4.33 \cdot 10^{-3} )</td>
<td>( 4.54 \cdot 10^{-2} )</td>
</tr>
<tr>
<td>( T/2 )</td>
<td>( 7.70 \cdot 10^{-4} )</td>
<td>( 6.92 \cdot 10^{-3} )</td>
<td>( 2.64 \cdot 10^{-3} )</td>
</tr>
<tr>
<td>( 3T/4 )</td>
<td>( 1.08 \cdot 10^{-3} )</td>
<td>( 8.80 \cdot 10^{-3} )</td>
<td>( 4.54 \cdot 10^{-2} )</td>
</tr>
<tr>
<td>( T )</td>
<td>( 1.17 \cdot 10^{-3} )</td>
<td>( 9.10 \cdot 10^{-3} )</td>
<td>( 2.87 \cdot 10^{-3} )</td>
</tr>
<tr>
<td>( 5T/4 )</td>
<td>( 1.37 \cdot 10^{-3} )</td>
<td>( 1.07 \cdot 10^{-2} )</td>
<td>( 4.55 \cdot 10^{-2} )</td>
</tr>
<tr>
<td>( 3T/2 )</td>
<td>( 1.44 \cdot 10^{-3} )</td>
<td>( 9.92 \cdot 10^{-3} )</td>
<td>( 2.98 \cdot 10^{-3} )</td>
</tr>
</tbody>
</table>

Table 7.5: Absolute errors in water depth, momentum and velocity using well-balanced central-upwind with \( w-u \) reconstructions using 400 cells and six different final time

**7.2 Test 2: Oscillations in a Parabolic Canal**

Recall the initial conditions given in Section 6.2 where \( D_0 = 10 \) m, \( 2L = 5,000 \) m, and \( A = L/2 \). In addition, the angular frequency of oscillation and the period of the analytical solution are respectively given by \( \omega = \sqrt{2gD_0/L} \) and \( T = 2\pi/\omega \approx 1121.43 \). Figures 7.3, 7.4, 7.5, and 7.6 show the results of the well-balanced central-upwind method with \( w-u \) reconstructions for solving this problem using the van Leer limiter, where the final time is \( t = T/4, t = T/2, t = 3T/4, \) and \( t = T \) respectively.

According to those figures, well-balanced central-upwind methods with \( w-u \) reconstructions are able to solve oscillations in a parabolic canal, and spurious oscillations do not appear in the water surface. In addition, well-balanced central-upwind methods with \( w-u \) reconstructions results in smaller error than those without well-balanced terms (consult [27]), and obviously than those resulting...
CHAPTER 7. STAGE AND VELOCITY RECONSTRUCTIONS

Figure 7.3: Well-balanced central-upwind with $w-u$ reconstructions using 400 cells and van Leer limiter for $t = T/4$

Figure 7.4: Well-balanced central-upwind with $w-u$ reconstructions using 400 cells and van Leer limiter for $t = T/2$
7.2. TEST 2: OSCILLATIONS IN A PARABOLIC CANAL

Analytical Solution
Numerical Solution
Canal Bed

Figure 7.5: Well-balanced central-upwind with $w-u$ reconstructions using 400 cells and van Leer limiter for $t = 3T/4$

Figure 7.6: Well-balanced central-upwind with $w-u$ reconstructions using 400 cells and van Leer limiter for $t = T$
from well-balanced methods with $w$-$uh$ reconstructions simulated in Chapter 6. From these facts, it can be inferred that well-balanced central-upwind methods with $w$-$u$ reconstructions solve unsteady flows better than other methods being compared.

## 7.3 Test 3: Steady Flow

This section considers the same problem solved in Section 6.3, but the problems are now solved by use of well-balanced central-upwind methods with $w$-$u$ reconstructions.

When the problem is solved using this method, it is found that the errors in three quantities of interest are larger than those resulting from well-balanced central-upwind with $w$-$uh$ reconstructions. Figure 7.7 shows the water profile after 25 seconds of running-time for both analytical and numerical solutions using 400 cells. (Note that the analytical solution does not change from one time instant to another time instant, as can also be seen in Chapter 6). The errors in water depth, momentum, and velocity are $E_h = 1.90 \cdot 10^{-5}$, $E_{uh} = 3.40 \cdot 10^{-5}$, and $E_u = 8.26 \cdot 10^{-5}$. Another test using minmod-Kurganov limiter has also been done as illustrated in Figure 7.8. This test leads to a slight diffusion in the momentum profile on the left boundary. The errors in this case are $E_h = 1.96 \cdot 10^{-5}$, $E_{uh} = 3.06 \cdot 10^{-5}$, and $E_u = 1.01 \cdot 10^{-4}$.

## 7.4 Concluding Remarks

In this chapter, well-balanced central-upwind methods with $w$-$u$ reconstructions have been tested. According to the simulations, these methods lead to a better performance than those with $w$-$uh$ reconstructions presented in Chapter 6. Furthermore, it is found that the methods using superbee limiter lead to the smallest error with a reasonable running time. It should be noted that the methods with minmod-Kurganov run very slowly and result in the same accuracy as those with pyvolution limiter.
7.4. CONCLUDING REMARKS

Figure 7.7: Well-balanced central-upwind with $w-u$ reconstructions using 400 cells and superbee limiter

Figure 7.8: Well-balanced central-upwind with $w-u$ reconstructions using 400 cells and minmod-Kurganov limiter
CHAPTER 7. STAGE AND VELOCITY RECONSTRUCTIONS
Chapter 8

Conclusions

This thesis has constructed and simulated well-balanced finite volume methods based on the steady state conditions of a lake at rest. The well-balanced terms are added in the flux calculation so as to alleviate the unphysical oscillations in the numerical result. Higher order well-balanced schemes can be obtained by numerical extrapolation applied to some lower orders of accuracy.

The methods being applied to calculate the fluxes in the numerical simulations are central-upwind finite volume methods. The simulations show that the well-balanced central-upwind finite volume methods with stage and momentum reconstructions can solve the steady flow nicely, but fail to solve the oscillations in a parabolic canal and dam-break problem in some cases. It can be inferred that these methods work well on steady flow in general, but may fail to solve unsteady flow with non-horizontal bottom topography. In addition, the performance of the methods rely much on the slope limiter being used, where the methods with minmod, van Albada, and van Leer limiters work better than those with minmod-Kurganov, pyvolution, and superbee.

The well-balanced central-upwind finite volume methods with stage and velocity reconstructions have also been tested. These methods lead to a better performance than those with stage and momentum reconstructions. Furthermore, the methods using superbee limiter lead to the smallest error with a reasonable running time, while the methods with minmod-Kurganov run very slow but result in the same accuracy as those with pyvolution limiter.
Appendix A

C Tutorial

The simulations presented in this thesis are mostly done in Python, where the expensive computations such as flux calculations are allocated in C. In order to understand how those two programming languages work, short tutorials are attached in this thesis.

Here we present short tutorial on C first, and integration of C with Python will be shown later. The basic knowledge on Python programming is assumed. Although we present more on C tutorial (since it is useful when we want to combine it with other language), but the main objection is on the application of mixed C and Python programming. The reasons of the mixed language programming are typically for migration of slow code in C, which is a compiled language, and/or accessing to the existing numerical code from Python, which is an interpreted language.

A.1 Preliminaries for C

In this section, some basic tutorials on C language are presented. Brief programming concept, and some examples are also discussed. More explanation on C programming can be found in some references, such as in [29].

A C program is a set of definitions of variables and functions. This means that the program is made up of functions and variables. A function contains statements enclosed in braces {} specifying the computing operation, whereas the values used during the computation are stored in variables. Communicating data between functions can be done, for instance for the calling function to provide a
list of values (i.e. arguments) to the function it calls. The list of arguments is surrounded by the parentheses following the function name.

A program to write the word "Hello!", for example, can be created as follows.

```c
#include <stdio.h>
main() {
  printf("Hello!\n")
}
```

The program above can be written in some text editor, named, and saved in a file with extension `.c`, such as `greet.c`. In Windows operating system if the gcc compiler is available then the file can be compiled with the command `gcc greet.c`. If it is done correctly, the compilation will make an executable file named `a.exe` automatically in the same directory where the `greet.c` is placed. Running `a.exe` by typing `a.exe` in the command prompt will print `Hello!`.

It should be noted that:

1. `#include <stdio.h>` at the first line asks the compiler to include information in the standard input-output library. In general, any source line of the form `#include "filename"` or `#include <filename>` is replaced by the contents of the file `filename`.

2. `main()` defines a function named `main`, which expects no arguments indicated by the empty list `()`.  

3. A sequence of characters in double quotes, such as "Hello!\n", is called a character string or string constant or just string.

4. `\n` asks to include a newline character in the `printf` argument.

Suppose a program consisting of three source files, viz `main.c`, `file1.c`, and `file2.c`, is given. To compile those three files we use a command of the form `gcc main.c file1.c file2.c` which results in three object files, namely `main.o`, `file1.o`, and `file2.o`, then loads them all into `a.exe`.

A declaration is an announcement stating the properties of variables and must be done before the variables are used. It lists the variables involved in the program, and states their type. Declaration refers to the place where the nature of
the variables is stated, but usually no storage is allocated. However, variables' initial values can also be set in the declaration and so the storages are allocated for the corresponding variables. For example,

```c
double fahr, cel;
int lower, upper, step = 1;
```

means that there are five variables involved. They are `fahr` and `cel` which are double-precision floating points, and `lower`, `upper`, and `step` which are integers where the `step`'s initial value is 1. After declaring those variables, it can be followed by initializing their values, such as `lower=0; upper=100`.

A `#define` name replacement-text is a definition which says that the subsequent occurrences of the token `name` will be replaced by anything in the `replacement text`. (Note: The difference between definition and declaration is that definition refers to the place where the name is created or assigned storage, but declaration refers to the place where the nature of the variables is stated. In addition, storage is always allocated in definition.) The name in a `#define` has the same form as a usual variable name, and the replacement text can be arbitrary such as a sequence of characters, a number, a loop, etc. For example,

```c
#define STEP 1
#define max(A, B) ((A) > (B) ? (A) : (B))
```

The first definition sets the symbolic constant `STEP` to 1. (Note: In C it is a convention that a symbolic name or symbolic constant is written in upper case and name of a variable is written in lower case.) The second defines `max(A,B)` to the maximum between `A` and `B`.

There are two kinds of arguments, namely formal argument (parameter) and actual argument (argument, for short). Parameter is a variable named in the parenthesized list in a function definition. Argument is the value used in a call of the function.

In C all function arguments are passed by value, not by reference. The called function is given the values of its arguments in temporary variables (i.e. local temporary copies) rather than the originals. This makes differences from several languages, such as Fortran and Pascal, where the called routine in those languages has access to the original argument. This "called by value" property is an advantage since it leads to fewer variables involved in a program.
Another advantage that C has is that it is still possible to modify the variables (i.e. original arguments) in a function. This can be done by providing the address of the variables to the function. The called function accesses the variables indirectly through pointers to the variables. Note that before doing so, the called function must declare the parameter to be a pointer. Discussion on pointer is presented in Section A.5.

A variable can be set whether it will or will not retain its value after call. A special local variable which retains its value between calls is called a static variable. It can be set by writing the word static in front of the type of the variable definition. A variable which is private or local to a certain function where it is declared in that function is called an automatic variable. Automatic variables come into existence only when the function is called and disappear when the function is exited. They do not retain their values from one call to the next.

A variable which is declared in the outermost block is called an external variable. It can be accessed by name by any function. External variables retain their values even after the functions setting them have returned. These variables must be declared in each function that wants to access it by stating extern and the type of the variables. If the function is in the same file where the external variable is defined, then the explicit extern is optional; but if the function is in different file, then the explicit extern is a must.

A return statement is a mechanism for returning a value from a called function to its caller. The return statement has the form

\[
\text{return expression}
\]

The expression will be converted to the return type of the function if necessary. Whenever the type of a function is void, then the function returns no value. Considering the main program as a function, it may also return a value; if the main program has ”return 0” at the final line, then it will have a normal termination.

### A.2 Types, Operators, and Expressions

In this section, a brief tutorial relating to types of objects, operators between objects, and expressions for data objects is presented. It should be noted that
• The basic data objects manipulated in a C program are variables and constants.

• Operators specify what is to be done to the variables and the constants. (Note: When using two or more consecutive operators in one statement, care should be taken regarding the level of precedence of the operators.)

• Expressions combine variables and constants to generate new values.

The type of an object determines the set of values it can have and what operations can be done on it. Each type has its special properties, for example an object declared `const` are prevented from being changed.

There are some restrictions on creating the names of variables and symbolic constants. Names are made up of letters and digits where the first character must be a letter. The underscore ",_," counts as a letter. It is advisable not to begin variable names with underscore, because library routines often use such names. In C upper case and lower case are distinct.

Care should be taken to distinguish between a character constant and a string that contains a single character. For example, `'x'` is different from "x". `'x'` is an integer used to produce the numeric value of the letter x, while "x" is an array of character containing one character (the letter x) and a '\0'.

An enumeration constant is a list of constant integer values. This type of constant is written as `enum`. The first name in an `enum` has value 0, the next 1, and so on, unless specific values are stated explicitly. For example, in `enum boolean {No, Yes}` the value of No is 0, and the value of Yes is 1; in `enum days {Sun=1, Mon, Tue, Wed, Thu, Fri, Sat}` the value of Sat is 7.

The modulus operator is denoted by `\%`. The expression `x\%y` produces the remainder when `x` is divided by `y` where `x` and `y` are int. In C, `%` operator cannot be applied to float or double.

There are two kinds of type conversion, namely automatic and explicit conversion. The only automatic conversions are those that convert a narrower operand into a wider one without losing information. Explicit type conversion can be coerced with a unary operator called cast. For example, if `n` is an integer, `sqrt((double) n)` converts the value of `n` to double before passing it to `sqrt`. The cast `(double)` produces the value of `n` in the intended type, but `n` itself is not changed.
If arguments are declared by a function prototype, the declaration causes automatic coercion of any arguments when the function is called. For example, if we are given a function prototype `double sqrt(double)`, the call `root2=sqrt(2)` coerces the integer 2 into the double value 2.0 without any cast needed.

Increment (++) and decrement (--) operators can be used either as prefix or postfix operators. The expression `++n` adds 1 to `n` before its value is used, while `n++` adds 1 to `n` after its value has been used. Note that these operators can only be applied to variables. For example if `n=3` then `x=++n;` sets `n` to 4 and `x` is set to 4. On the other hand, `x=n++;` sets `x` to 3 and `n` becomes 4.

The assignment operator `+=` is a shorthand for a constant increment. For example, `i=i+5` can be shortly written as `i+=5`. In general, if `expr_1` and `expr_2` are expressions, then

```plaintext
expr_1 op= expr_2
```

is equivalent to

```plaintext
expr_1 = (expr_1) op (expr_2)
```

where `op` is one of binary operators `+, -, *, /, \%, <<, >>, &, ^, |`.

The ternary operator `?:` provides an alternate way to write conditional expression. To illustrate how it works, consider `z=max(a,b)`. Then this maximum can be written by the statement `z=(a>b)?a:b` which is equivalent to

```plaintext
if (a>b)
    z=a;
else
    z=b;
```

### A.3 Control Flow

The control-flow statements of C language specify the order in which computations are done.

Unlike in Pascal which treats semicolon as a separator, C treats semicolon as a statement-terminator. An expression such as `x=0` or `i++` or `printf (...)` becomes a statement when it is followed by semicolon, as in: `x=0; i++; printf (...) ;` .

The `switch` statement tests whether an expression matches one of a number of constant integer values and therefore whether it matches one of a number of
A.3. CONTROL FLOW

branches. The form of a switch is as follow.

```c
switch (expression)
{
    case const−expr: statements
    case const−expr: statements
    default: statements
}
```

There are two special statements, i.e. break and continue, which are useful related to loop or switch. The break statement provides an early exit from any loop (including for, while, and do loops) as well as from switch. A break causes the innermost enclosing loop or switch to be exited immediately. The continue statement causes the next iteration of the enclosing loop to begin. A break can be applied to either a loop or switch, but continue can only be applied to a loop. A continue inside a switch inside a loop causes the next loop iteration. In a special case where the continue is not involved in the loop statement, we have two equivalent loops:

```c
for (expr_1; expr_2; expr_3)
    statement
```

has the same meaning as

```c
expr_1;
while (expr_2)
{
    statement
    expr_3;
}
```

Comma "," operator is used to guarantee left to right evaluation where the type and the value of the result are the type of the value of the right operand. The commas that separate function arguments, variables in declaration, etc. are not comma operators and do not guarantee left to right evaluation. Comma operators often finds use in the for statement. For example, consider the function reverse(s) which reverses the string s in place and which processes two indices in parallel as follows.

```c
#include <string.h>
/* reverse: reverse string s in place */
```
void reverse(char s[]) {
    int c, i, j;
    for (i = 0, j = strlen(s)-1; i < j; i++, j--)
    {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}

The loop in the function above can be written in a single-exchange-operation using comma operators:

    for (i = 0, j = strlen(s)-1; i < j; i++, j--)
    c = s[i], s[i] = s[j], s[j] = c;

The goto can be used to abandon processing in some deeply nested structure, for example breaking out two or more loops at once. (Note: In practice it is almost always possible to create a code without goto.) The break statement cannot be used directly in this case since it only exists from the innermost loop.

A.4 Functions and Program Structures

Here basics of functions and variety of variables are covered. Some comparisons between types of variables are also presented.

Function definition has the form

    return -type function - name (argument declarations)
    {
        declarations and statements
    }

If return type of the function is omitted, int is assumed to be the return type of the function. The simplest function is do-nothing function, that is a function which has the form

    dummy() {} 

The declaration of a function must be consistent in terms of its type and its argument type with its definition. For example, consider the declaration
A.4. FUNCTIONS AND PROGRAM STRUCTURES

```c
double atof(char[]);
```

if `atof` itself and the call to it have inconsistent types in the same source file, the mismatch will be detected by the compiler. However, if `atof` is compiled in a different file, the error will not be detected and meaningless result will occur.

In addition, if a function takes arguments, they must be declared. If it takes no arguments, it is advisable to use `void` as the argument.

C does not allow functions to be defined inside other functions. Nonetheless, if two functions must share some data, but no one calls each other, it is convenient if the shared data is kept in external variables rather than passed in and out through arguments. (Note: External variables are discussed in Section A.1.) In a large program, all definitions can be placed in one file called a header file, so that they can be shared and it will be easier to maintain as the program evolves.

A variable or function can be made invisible outside of the scope in which it is declared, by stating `static` in the declaration. (Note: The scope of a name is the part of the program within which the name can be used.) The `static` declaration, applied to an external variable or function, limits the scope of that object to the rest of the source file being compiled. If `p` is a static variable of type `int` with 1 as its initial value, then the declaration can be stated by `static int p = 1;`.

There are two kinds of variables which are local to a particular function, namely automatic and internal `static` variables. The declaration form of internal `static` variable is the same as the external `static` variable. Internal `static` variables provide private permanent storage within a single function, while automatic variables provide temporary storage which means that automatic variables are coming and going each time when the function is activated.

A variable may be placed in machine register which may result a faster program by `register` declaration. A register declaration advises the compiler that the variable will be used heavily. However, the compiler is free either to take or to ignore the advice. Register declaration can only be applied to automatic variables and to the parameters of a function. The address of a register variable cannot be taken by a pointer. (Pointer is discussed in the next section.) The `register` declaration looks like `register int x;`.

Several properties of external and static variables regarding initialisation of their values are as follows.
1. In the absence of explicit initialisation, external and static variables are guaranteed to be initialised to zero.

2. Initialisation must be a constant expression.

3. Initialisation is done once before the program begins execution

The initialisation of automatic and register variables has properties as follows.

1. In the absence of initialisation, automatic and register variables have undefined, i.e. garbage, initial values.

2. Initialisers is not restricted to being a `const`, it may be any expression involving previously defined values, or even function calls.

3. Initialisation is done each time the function or block is entered.

### A.5 Pointers and Arrays

A pointer is a variable that contains the address of another variable (Note: However, a function is not a variable, but it is possible to define pointers to functions. The pointers can be assigned, placed in arrays, passed to functions, returned by functions, etc.) Pointers and arrays are closely related; that is why both of them are presented in the same section here.

The unary operator `&` gives the address of an object. Then, the statement `p=&c` assigns the address of `c` to the variable `p`, and furthermore `p` is said to point to `c` (i.e. `p` is a pointer to `c`; `&c` is the address of `c`). The `&` can only be applied to objects in memory including variables and array elements, and cannot be applied to expressions, constants, or register variables.

The unary operator `*` is the indirection or derefencing operator. When this operator is applied to a pointer, it accesses the object the pointer points to. Suppose that `x` is an integer and `ip` is a pointer to `int`, then `*ip` can occur in any context where `x` could. An illustration to show how to declare a pointer and how to use `&` and `*` is as follows:

```c
int x = 1, y = 2;    /* x=1 and y=2 are integers */
int z[10];           /* z is an array of 10 integers */
int *ip;             /* ip is a pointer to int */
```
A.5. POINTERS AND ARRAYS

Since C passes arguments to functions by value as stated in Section A.1, there is no direct way for the called function to alter a variable in the calling function. To change a variable, a pointer is needed. For example, to swap the values of two int variables, it is not enough to write `swap(a, b);` where the `swap` function is defined by

```c
void swap(int x, int y) /* This is wrong! */
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}
```

since that function only swaps the copy of `a` and that of `b`.

To get the expected result, the values have to be passed through pointers. That is by writing `swap(&a, &b);` where the `swap` function is defined by

```c
void swap(int *px, int *py) /* swap *px and *py */
{
    int temp;
    temp = *px;
    *px = *py;
    *py = temp;
}
```

Any operation that can be achieved by array subscripting can also be done with pointers. Consider `int a[10]` which defines an array `a` of size 10. If `pa` is a pointer to an integer (declared as `int *pa`), then the assignment `pa = &a[0]` sets `pa` to point to `a[0]`. Furthermore, `x = *pa` copies the contents of `a[0]` into `x`. The assignment `pa = &a[0]` can also be written as `pa = a` since the name of an array is a synonym for the location of the initial element. Moreover, if `pa = &a[0]`, `pa+i` is the address of `a[i]` and `*(pa+i)` is the contents of `a[i]`. (Note: The pointer `*(pa+i)` can also be written as `*(a+i);` and `pa[i] = *(pa+i).`) In addition, if `p` is a pointer to
some element of an array, then p++ increments p to point to the next element, and p+=i increments it to point i elements beyond where it currently points to.

Similarly, &a[i] and a+i are also identical. In other words, an array-and-index expression is equivalent to one written as a pointer-and-offset. Other equivalences are:

1. char s[] and char *s are equivalent, since s[] = *s, where s is string
2. f(&a[i]) and f(a+i) are equivalent, since &a[i]=a+i
3. f(int arr[]){...} and f(int *arr){...} are equivalent.

### A.6 Structures

This section deals with structure or struct. A struct is a collection of one or more variables, possibly of different types, grouped together under a single name for convenient handling. This means that a struct is a variable consisting of variables.

A variable described by struct can be written either directly or indirectly. If a variable A is declared by a direct struct, then it is written

```c
struct {...} A;
```

which means that x has the structure of whatever in {...}. The other option is the indirect struct which can be made by creating the struct prototype beforehand. For example, a structure prototype named `point`:

```c
struct point
{
    int x;
    int y;
};
```

may explain the structure of an arbitrary point \((x, y)\) in two-dimensional Cartesian coordinate. (Note: This is for indirect struct only that a name called structure tag is given by putting it after the word struct. In this example, point is the tag name.) After having the struct prototype, then if A has the structure of point, it can be declared by

```c
struct point A;
```

The variables named in a structure are called members. For example, for the `point` structure above, x and y are the members. The structure and the member
A.7. Input and Output

This section briefly discusses and concentrates on input-output.

A program running on a computer interacts with its environment, such as standard library as desired. The most basic kinds of inter-actions are input and output: input from the keyboard or a file, and output to the screen or a file. In C input and output facilities are not part of the C language, but the ANSI standard defines them such that they are compatible with C.

The function printf returns and prints output with a specific format. The format specification begins with \% and ends with a conversion character. Between the \% and the conversion character there may be, in order:

- A minus sign, which specifies left adjustment of the converted argument.
- A number that specifies the minimum field width.
- A period, which separates the field width from the precision.
- A number, the precision, that specifies the maximum number of characters to be printed from a string, or the number of digits after the decimal point of a floating point value, or the minimum number of digits for an integer.
- An h if the integer is to be printed as a short, or l if as a long.

To illustrate how printf works, assume that s is the string ”hello, world” which consists of 12 characters. Then, the call printf("\%s", s); will result hello, world
which is exactly the same as what `s` is. In addition,

```
printf(":\%−15.10s:);
```

will write

```
: hello, world :
```

In the later experiment, `\%−15.10s` is used, and the explanation is as follows:

- It is left adjustment since there is a minus sign.
- The minimum field width is 15 characters.
- Since there is a period, some precision is applied.
- The maximum number of characters to be printed is 10. (i.e. The string precision is 10.)

Another example, if `fahr=20.0` is a floating point,

```
printf(":\%6.2f:", fahr);
```

will print

```
: 20.00:
```
Appendix B

C-Python Tutorial for Computational Science

Most languages offer the possibility to call code written in other languages, such as Fortran, C, and C++. Combination of Python with C code is of interest in two reasons:

1. **Migration of slow code.** A new application is written in Python, but numerical intensive calculations are migrated to C.

2. **Access to existing numerical code.** Existing numerical libraries or applications in C are called from Python.

In both cases we want to benefit from using Python for non-numerical tasks. More explanation and examples on integration of Python with C and be found in some references, such as [38].

To illustrate how functions in Python and C work, we consider writing a function for \( z = \sin(x + y) \). A pure Python implementation reads

```python
"""Pure Python Scientific Hello World module."""
import math, sys

def hw1(r1, r2):
    s = math.sin(r1 + r2)
    return s

def hw2(r1, r2):
    s = math.sin(r1 + r2)
```

115
Here hw1 function returns a value and contains pure numerical computations. Calling in the Python Shell hw1(3.0,0.14) results 0.0015926529164868282. On the other hand, hw2 does not return a value and not merely contains numerical computations but also performs input-output. Calling

\[ \text{hw2}(3.0,0.14) \]

gives

\[ \text{Hello, World! } \sin(3+0.14) = 0.00159265 \]

The two functions hw1 and hw2 above can be written in C as well. The implementation in C could take the form as follows; here hw3 is a special version of hw1 which takes the result as an argument.

```c
#include <stdio.h>
#include <math.h>

double hw1(double r1, double r2)
{
    double s;
    s = \text{sin}(r1 + r2);
    return s;
}

void hw2(double r1, double r2)
{
    double s;
    s = \text{sin}(r1 + r2);
    printf("Hello, World! \sin(%g + %g)=%g \n", r1, r2, s);
}

/* special version of hw1, the result is an argument: */
void hw3(double r1, double r2, double *s)
{
    *s = \text{sin}(r1 + r2);
}
```
B.1 Combining Python with C

Suppose we in a Python script want to call a C function \textit{hw1}, available in a module \textit{hw}, which takes two \texttt{double}s as arguments and returns a double:

\begin{verbatim}
extern double hw1(double r1, double r2);
\end{verbatim}

The Python code must call a wrapper function, written in C. Then, the wrapper function can call the C function \textit{hw1}. A wrapper function can look as follows:

\begin{verbatim}
static PyObject *wrap_hw1(PyObject *self, PyObject *args)
{
    double arg1, arg2, result;
    if (!PyArg_ParseTuple(args, "dd: hw1", &arg1, &arg2))
    {
        return NULL; /* wrong arguments provided */
    }
    result = hw1(arg1, arg2);
    return Py_BuildValue("d", result);
}
\end{verbatim}

A wrapper function typically takes two arguments, \texttt{self} and \texttt{args}. The \texttt{self} is for dealing with instance methods, and \texttt{args} represents for a tuple of the arguments from Python (\texttt{r1}, and \texttt{r2} in this case). The utility \texttt{PyArg_ParseTuple} in the Python C library is for converting the \texttt{args} object to two \texttt{double} variables which are specified as the string \texttt{dd}. These two double variables are stored in \texttt{arg1} and \texttt{arg2}, and therefore, the function \textit{hw1} which has been written in C can be called. In addition, since the \textit{hw1} function returns a \texttt{double}, we need to convert this \texttt{double} to a Python object that can be returned to the calling Python code; the returned value from \textit{hw1} must be converted to a proper Python \texttt{float} object, with aid of the function \texttt{Py_BuildValue} in the Python C library.

B.2 Calling C from Python

Having the function \textit{hw1} and its wrapper in C, then in the Python script we can write

\begin{verbatim}
from hw import hw1
r1 = 1.2
r2 = -1.2
\end{verbatim}
s = hw1(r1, r2)

B.3 Example I: Dot Product of Vectors

To illustrate how the combination between C and Python works, we create a script for evaluating dot-product of two vectors. As has been presented in front, in C we need two main parts namely the evaluation function and the Python wrapper. In the code mul_arrays.c, we also include the method table for handling the NumPy structures.

Calling the module mul_arrays.c, we can give any input from the Python Shell to that module and the output will be sent to the Python Shell. As an instance to test the module we create multest.py. Here, we take two vectors of the size 10
\[
x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14] \\
y = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
\]

and we get the output \(x.y = 90.0\). (mul_arrays.c and multest.py are attached in Example III.)

B.4 Example II: Unit Testing

Unit testing is a test suite that puts functions through their paces and makes sure that they behave the way we want them to. The reason why it is called unit testing is that the set of two conversion functions can be written and tested as a unit, independent from any larger program in which they are part of the later.

Python has a framework for unit testing, named unittest module. This module is included in Python 2.1 and later. It is advisable that the unit tests are written before the code that they test, and keeping them updated as code and requirements evolve. The advantages of unit testing are for instance:

- It details the code requirements in a useful fashion.
- It helps us form over-coding.
- When the code is refactored, it assures that the new version behaves the same way as the old version.
B.4. EXAMPLE II: UNIT TESTING

- It increases the confidence to break up an assignment in a team, since all the work can be shared and tested.

More discussion on unit testing can be found in [56].

As an illustration how to make a unit test, suppose that we want to test a function of Lagrange polynomial interpolation called `lagrange` that will be created in `interpolate.py`. The function will look like

```python
lagrange(x, w)
```

where \(x\) and \(w\) are arrays of the given points and its function values, respectively. This function takes two arrays as its arguments (or parameters), where the first parameter is an array of interpolating points and the second parameter is an array of the exact function values at those point. In addition, this function results an array of interpolating polynomial coefficients.

To create a unit test, we consider the function \(f(x) = \exp(x)\). Given the three interpolating points \(-1.0, 0.0, 1.0\), we can do hand calculation that the function evaluations at those points are \(0.367878441, 1.0, 2.718281828\). Furthermore, by hand calculation the interpolating polynomial is \(F(x) = 0.54308013x^2 + 1.17520169x + 1\). Having these all, the unit test can be written as follows:

```python
from interpolate import lagrange
import unittest, scipy
from numpy import array, allclose

class TestInterpolation(unittest.TestCase):
    def setUp(self):
        self.x = array([-1.0, 0.0, 1.0])
        self.w = array([0.367878441, 1.0, 2.718281828])
        self.p = array([0.54308013, 1.17520169, 1.0])

    def tearDown(self):
        pass

    def testInterpolateTry(self):
        pc = lagrange(self.x, self.w)
        assert allclose(self.p, pc)
```

B.5 Example III: C-Code for Dot-Product of Vectors

The `mul_arrays.c` module is written as follows:

```c
#include "Python.h"
#include "Numeric/arrayobject.h"
#include "math.h"
#include "stdio.h"

double _multiply_arrays(int n, double x, double y, double z[])
{
    int k;
    double sum = 0.0;
    for (k=0; k<n; k++)
    {
        z[k] = x[k] * y[k];
        sum += z[k];
    }
    return sum;
}
```

---

```python
suite = unittest.makeSuite(Test_Interpolation, 'test')
runner = unittest.TextTestRunner()
runner.run(suite)
```

---

PLAGIAT MERUPAKAN TINDAKAN TIDAK TERPUJI
/* multiply components of x and y into array z */
PyArrayObject
* x,
* y,
* z;

// Local variables
int nx, ny, nz;
double sum;

// Convert Python arguments to C
if (!PyArg_ParseTuple(args, "OOO", &x, &y, &z))
{
   PyErr_SetString(PyExc_RuntimeError,
                   "multiply_arrays.c:input could not be parsed");
   return NULL;
}

nx = x->dimensions[0];
ny = y->dimensions[0];
nz = z->dimensions[0];
sum = _multiply_arrays(nx, (double*) x->data,
                       (double*) y->data,
                       (double*) z->data);

// Release and return
printf("sum = \%f\n", sum);
return Py_BuildValue("d", sum);

// Method table for python module
static struct PyMethodDef MethodTable[] =
{
    {"multiply_arrays", multiply_arrays,
     METH_VARARGS, "Print_out"},
    {NULL, NULL}
};

// Module initialisation
//     void initmul_arrays(void)
//       {
//         Py_InitModule("mul_arrays", MethodTable);
//         import_array();
//       }

The multest.py module is written as follows:

```python
import Numeric as N
from Numeric import arange, zeros, Float
import mul_arrays as ma

x = N.array(N.arange(10), Float)
y = N.array(N.zeros(10) + 2, Float)
z = N.zeros(10, Float)
mul = ma.multiply_arrays(x, y, z)
print 'x.y=', mul
```
Bibliography


