Technical University of Munich

Department of Mathematics

Master's Thesis in Mathematics

Discontinuous Galerkin Schemes for Dispersive Non-Hydrostatic Shallow Water Equations

Discontinuous-Galerkin-Schemata für dispersive, nicht hydrostatische Flachwassergleichungen

Author	David Jonathan Schneller
Supervisor	Prof. Dr. Hans-Joachim Bungartz
Advisors	M. Sc. Leonhard Rannabauer, M. Sc. Lukas Krenz
Date of Submission	March 15, 2022

Eidesstattliche Erklärung

Ich versichere, dass ich diese Masterarbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

I confirm that this Masters's thesis is my own work and I have documented all sources and material used.

Place / Date / Signature

Abstract

We consider the H-BMSS-γ system published in [Escalante et al., 2019]. It is a shallow water-like system which captures non-hydrostatic pressure effects. In addition to that, it is a non-conservative hyperbolic system. We consider two types of solutions to the H-BMSS-γ system: firstly, we look at a solitary wave, for which we suggest better values for the constants compared to the already existing solutions. Secondly, we compute the solution to Riemann Problem without bathymetry, but also in the dry case. Furthermore, we implement a numerical discretization of the H-BMSS-γ system using both a finite volume method, and an ADER-DG (Adaptive DERivative Discontinuous Galerkin) method. The ADER-DG method which we implemented uses an additional a posteriori limiter with our finite volume method, in order to handle discontinuities better. We also construct four well-balanced fluxes for said system, and compute a Roe average matrix. Finally, we show experimentally that our scheme observes the Resting Lake Property, the convergence of it to reference solutions. Additionally, we verify that it conforms to our two solutions to the H-BMSS-γ system.

Keywords: ADER-DG, BMSS, H-BMSS, Riemann Problem, Solitary Wave, sam(oa)²

Zusammenfassung

Wir betrachten das H-BMSS-γ-System, welches in [Escalante et al., 2019] veröffentlich wurde. Es ist ein Flachwasser-artiges Gleichungssystem, welches nicht hydrostatische Druckeffekte einfängt. Zudem ist es ein nicht konservatives, hyperbolisches System. Wir betrachten zwei Typen von Lösungen zum H-BMSS-γ-System: Erstens betrachten wir ein Soliton, für welches wir bessere Konstanten vorschlagen, als diese in bestehenden Lösungen verwendet werden. Zweitens berechnen wir die Lösung des Riemann-Problems ohne Bathymetrie, aber auch im trockenen Fall. Zudem implementieren wir eine numerische Diskretisierung des H-BMSS-γ-Systems mit sowohl einem Finite-Volumen-Schema, also auch mit der ADER-DG-Methode (Abkürzung für Adaptive DERivative Discontinuous Galerkin). Die ADER-DG-Methode, so wie wir sie implementiert haben, benutzt einen zusätzlichen a posteriori angewandten Finite-Volumen-Limiter, um Unstetigkeiten besser behandeln zu können. Zusätzlich konstruieren wir vier gut ausgeglichene, numerische Flussfunktionen und berechnen eine Roe-Matrix. Zuletzt zeigen wir experimentell, dass unsere Implementierung die Ruhige-See-Bedingung einhält und dass die berechneten Lösungen zu Referenzlösungen konvergieren. Ebenfalls verifizieren wir, dass unsere zwei berechneten Lösungen zum H-BMSS-γ-System korrekt simuliert werden.

Schlüsselwörter: ADER-DG, BMSS, H-BMSS, Riemann-Problem, Soliton, sam(oa)²

Acknowledgments

I thank the Almighty God, and my parents for their continuous support not only through this thesis, but also during all of my higher education. Moreover, I would like to thank my advisors and my supervisor for their help and assistance with any questions I had while writing this thesis, and for many helpful discussions which I had with them. Furthermore, my thanks go to the Hanns Seidel Foundation (Hanns-Seidel-Stiftung) for their financial support, funded by means from the Federal Ministry of Education and Research (Bundesministerium für Bildung und Forschung).

Contents

1 Introduction

This work stems from the fundamental search after good fluid dynamics models for use cases where two-dimensional effects dominate. We want the model to capture dispersive effects, complicated bathymetry and pressure-related effects well, and it should be computationally simple at the same time. For example, taking models like the Navier-Stokes equations or just the Euler equations in three dimensions are be accurate enough, but it is usually computationally hard to use them. Or on the other hand, the shallow water equations which read without bathymetry

$$\begin{aligned} \partial_t h + \nabla \cdot (h\mathbf{u}) &= 0, \\ \partial_t (hu) + \nabla \cdot (h\mathbf{u}\mathbf{u}^T) + \nabla \left(\frac{g}{2} h^2\right) &= 0, \end{aligned} \tag{1.0.1}$$

are computationally comparably easy to solve and form a hyperbolic system. Here, h is the water height and $\mathbf{u}$ is the horizontal velocity vector. However, the shallow water equations do not capture non-trivial pressure-related effects. This is because they assume the pressure to be hydrostatic, i.e. only dependent on the water height [3]. To overcome these problems, a system with an additional non-hydrostatic pressure contribution has been introduced by [3]. It is constructed in the same way as the shallow water equations are, that is by depth-averaging the Euler equations. The pressure is decomposed into a hydrostatic and a non-hydrostatic part. The non-hydrostatic pressure is coupled with the vertical velocity, and both become new variables we have to consider. However, since the system includes a divergence-free condition, we lose the hyperbolicity in the process. Thus, the system becomes computationally more difficult. In [15], a hyperbolic system is proposed which relaxes the system from [3] by coupling the divergence-freedom constraint with an evolution equation the non-hydrostatic pressure. The objective of this is to obtain divergence freedom as time goes on. An artificial wavespeed c controls how close the system [15] is to the original, non-hyperbolic system. While the new system is hyperbolic, the equation for the non-hydrostatic pressure contains a non-conservative part which makes the analysis and the construction of numerical fluxes more difficult.

Given this hyperbolic system, we look in Chapter 2 at two kinds of one-dimensional solutions to it: first of all, we consider solitary waves, i.e. we introduce a similarity variable $(x - t)$ and solve the resulting system to obtain a quasi-exact solution. While this was already done in [15], the derivations provided there did not work and contained multiple errors in the formulas. Thus we re-do the work and correct some mistakes and propose new, more correct values for the parameters. Secondly, we look at the Riemann Problem, that is the equation as a Cauchy problem in a single dimension with discontinuous initial conditions. This requires some additional theory, including the so-

called DLM Paths [35, Definition 2.1], in order to deal with the non-conservative part that is present in the new system.

Next, in Chapter 3, we discretize the system from [15] using both a Finite Volume and the so-called Adaptive DERivative Discontinuous Galerkin (ADER-DG) method [11]. The ADER-DG method solves the time integration by a fixed-point iteration which is done locally on each element, named the predictor step. After that, the data from the neighboring elements is taken into account, in the so-called corrector step. In order to make this method work for discontinuous data, we employ an a posteriori finite volume limiter which uses the previously-constructed finite volume method for the system. In addition, we construct and implement four numerical fluxes for the system. While [15] uses a Rusanov flux and an unnamed PVM method [6], we implement a Roe average matrix and the Roe method[36], as well as the DOT flux [13] and the HLL flux [6].

In Chapter 4, we implement the system from [15] using the ADER-DG and Finite Volume discretizations in sam(oa)^2[32, 40], that is, in a two-dimensional setting. In addition, we gain immediate support of OpenMP and MPI through this implementation.

Finally, we evaluate our implementation in sam(oa)^2 in Chapter 5, and experimentally confirm that our solution converges. In addition, we test and verify the correct behavior with respect to our previously-calculated solutions for the solitary wave and the Riemann Problem. Lastly, we summarize our achievements in Chapter 6 and close with remarks on future work.

1.1 Notation and Basic Definitions

We use lower-case non-bold letters to denote scalar variables and scalar functions, e.g. f, g. For vector-valued functions, we use bold letters, e.g. $\mathbf{u}, \mathbf{v}$. Bold letters with subscript are also used for left and right state vectors which are not functions. The element-wise product of vector-valued functions is written as

$$\mathbf{u} \odot \mathbf{v}. \tag{1.1.1}$$

We use ∂_x to denote the partial derivative operator with respect to the variable x. The gradient $\nabla = (\partial_{x_1}, ..., \partial_{x_d})^T$ is the vector of partial derivatives w.r.t. the space directions $x_1, ..., x_d$. Moreover, we denote the scalar product in $\mathbf{R}^d$ by a dot (e.g. $\mathbf{u} \cdot \mathbf{v}$). Subsequently, the divergence of a function $\mathbf{u}$ can be written as $\nabla \cdot \mathbf{u}$. By DF, we denote the Jacobian of a function F.

To write that a function f is equal to a constant c everywhere, we write $f \equiv c$.

For curves, we use the letters Ψ and Φ, or other uppercase greek letters. The curve integral of a function $\mathbf{F} : \mathbb{R}^N \to \mathbb{R}^{M \times N}$ over an almost everywhere differentiable curve $\phi : [a, b] \to \mathbb{R}^N$ is written as

$$\int_\phi \mathbf{F}(z) \, \mathrm{d}z := \int_a^b \mathbf{F}(\phi(s))\phi'(s) \, \mathrm{d}s \in \mathbb{R}^M. \tag{1.1.2}$$

If not defined otherwise, we denote the dimension by d, e.g. if we say we consider the case $d = 2$, we mean that we consider the two-dimensional case.

2 The Governing Equations

The equation which system we consider in this work is originally described in [15]. As done in [16], we will call it H-BMSS-γ.

We will first explain how the equation system is built. The actual construction process from the Euler equations using depth-averaging is briefly described in Section 2.1.2, and also in [3, Sections 2, 3]. In particular, we look at depth-averaged variables in the following description. Firstly, we have the mass conservation equation

$$\partial_t h + \nabla \cdot (h\mathbf{u}) = 0, \tag{2.0.1}$$

where $h(t, x)$ is the water height, and $\mathbf{u}(t, x)$ is the horizontal velocity vector. Secondly, we couple it with a conservation of momentum equation which is written without bathymetry influence as

$$\partial_t(h\mathbf{u}) + \nabla \cdot (h\mathbf{u}\mathbf{u}^T) + \nabla(hp_T) = 0. \tag{2.0.2}$$

Note that we have d equations in (2.0.2) at the same time. Here, d denotes the space dimension, excluding the vertical axis. Thus, we have $\mathbf{u} = (u)$ in the one-dimensional case ($d = 1$), and $\mathbf{u} = (u, v)^T$ in the two-dimensional case ($d = 2$). p_T is the pressure which is decomposed into a hydrostatic, and a non-hydrostatic component

$$p_T = \frac{g}{2}h + p. \tag{2.0.3}$$

Only the non-hydrostatic component needs a variable on its own which we name $p(t, x)$. The gravitational acceleration is denoted by $g > 0$. If we had $p \equiv 0$, we would get the shallow water system. This equals the absence of non-hydrostatic pressure. In addition, we would have $(d+1)$ equations for $(d+1)$ unknowns, h and $h\mathbf{u}$. But because we assume that $p \equiv 0$ does not hold in general, we now still need to close the equation system. Thus, we need to specify the behavior of p. We couple it with the vertical velocity variable $w(t, x)$ which is a scalar, unlike $\mathbf{u}$. It is logical to take a new conservation of vertical momentum equation for w, but we allow p to contribute to w as a source term. We obtain

$$\partial_t(hw) + \nabla \cdot (h\mathbf{u}w) = \gamma p. \tag{2.0.4}$$

Here, $\gamma \geq 0$ is a parameter which controls the strength of the influence of p onto w. To close the equation system, we additionally impose that the velocities $(\mathbf{u}^T, w)^T$ in both horizontal and vertical directions are divergence-free (see the remark near [3, eq. (40)]). After depth-averaging, this gives us the condition

$$w + \frac{h}{2}(\nabla \cdot \mathbf{u}) = 0. \tag{2.0.5}$$

Combining all these four equations gives us the so-called BMSS-γ system.[1] However, the divergence-free condition (2.0.5) gives us a non-hyperbolic system which makes it computationally more difficult to solve.

To circumvent this, as suggested in [15], we look at (2.0.2) again. This time insert p_T (2.0.3) to obtain for a parameter $c > 0$

$$\left(\partial_t(h\mathbf{u}) + \nabla \cdot (h\mathbf{u}\mathbf{u}^T) + \nabla\left(\frac{1}{2}gh^2\right)\right) + c^2\nabla\left(\frac{1}{c^2}hp\right) = 0. \tag{2.0.6}$$

The approach is now re-interpret (hp/c^2) as a correction term for the divergence constraint (2.0.5), and c^2 as Lagrange multiplier. In addition, we couple the system with an evolution equation for (hp/c^2) which includes (2.0.5), given as[2]

$$\frac{1}{c^2}\left(\partial_t(hp) + \nabla \cdot (h\mathbf{u}p)\right) = -2\left(w + \frac{h}{2}(\nabla \cdot \mathbf{u})\right). \tag{2.0.7}$$

This equation becomes a mass conservation equation for (hp), iff (2.0.5) is fulfilled. Thus, $c \geq 0$ controls the relation the two equations in (2.0.7) are combined. In [15], c is chosen to be $c^2 = \alpha^2 gH$, for $\alpha \geq 0$ being a real number (it is suggested to take $\alpha > 1$), and $H \geq 0$ a reference water height level. Reformulating (2.0.5) by using the product rule (to be able to write it in conservative variables, at least when it comes to derivatives) gives

$$-(\mathbf{u} \cdot \nabla)h + \nabla \cdot (h\mathbf{u}) = -2w. \tag{2.0.8}$$

Thus, we get for (2.0.7) the equation

$$\partial_t(hp) + \nabla \cdot (h\mathbf{u}(p + c^2)) - c^2(\mathbf{u} \cdot \nabla)h = -2c^2 w. \tag{2.0.9}$$

Combining (2.0.1), (2.0.2), (2.0.4), (2.0.9), adding bathymetry and terms for friction and wave breaking, we obtain the system which is handled in this work. It reads

$$\begin{aligned}
\partial_t h + \nabla \cdot (h\mathbf{u}) &= 0, \\
\partial_t(h\mathbf{u}) + \nabla \cdot (h\mathbf{u}\mathbf{u}^T) + \nabla\left(\frac{g}{2}h^2 + hp\right) &= -(gh + \gamma p)\nabla b - \tau_b(h, \mathbf{u}), \\
\partial_t(hw) + \nabla \cdot (h\mathbf{u}w) &= \gamma p + R_b(\mathbf{U}, \nabla\mathbf{U}), \\
\partial_t(hp) + \nabla \cdot (h\mathbf{u}(p + c^2)) - c^2(\mathbf{u} \cdot \nabla)h &= 2c^2(\mathbf{u} \cdot \nabla)b - 2c^2 w.
\end{aligned} \tag{2.0.10}$$

In [15], this system is written in one dimensions [15, eqs. (4), (8)] and two dimensions [15, eq. (16)]. The only additional variable that is new in this system is the bathymetry $b(x)$. It is constant over time, so we could add the equation

$$\partial_t b = 0 \tag{2.0.11}$$

[1] The name BMSS appears in [16] for $\gamma = 2$, and it is named after the authors of [3]. But since [3] also publishes the whole family of equations, we decided to name the whole family of systems BMSS-γ.

[2] This choice stems from [15]. Other choices are possible, see [9], but they may not lead to a hyperbolic system.

to the equation system and write $b(t,x)$ to be dependent on time. The paper [15] uses H as a variable for $-b$ instead. In addition, we have the new parameter $\gamma \geq 0$ which controls the influence of the pressure onto the velocity. Depending onto which value γ is set, we obtain a different model.

Given all these definitions, we define the vector of conservative variables

$$\mathbf{U} = \begin{pmatrix} h \\ h\mathbf{u} \\ hw \\ hp \\ b \end{pmatrix}. \tag{2.0.12}$$

It effectively works as a short-hand notation and will be used in the following on many occasions. In addition, we have the vector of primitive variables

$$\mathbf{U}_P = \begin{pmatrix} h \\ \mathbf{u} \\ w \\ p \\ b \end{pmatrix}. \tag{2.0.13}$$

Next, we have the two additional terms τ_b and R_b. They are not required for the model itself to work, but they give them additional features. The friction term τ_b is given as

$$\tau_b(h,\mathbf{u}) = \frac{n_m^2 g}{h^{1/3}}(|\mathbf{u}| \odot \mathbf{u}), \tag{2.0.14}$$

using the point-wise vector multiplication symbol. n_m is the bottom friction coefficient has to be determined experimentally. The wave-breaking mechanism is defined as

$$R_b(\mathbf{U}, \nabla\mathbf{U}) = \begin{cases} 2e_B \left(\frac{\nabla\cdot(h\mathbf{u})}{B_1\sqrt{gh}} - 1\right) h\,|\nabla\cdot(h\mathbf{u})|\,(w - (\mathbf{u}\cdot\nabla)b) & \text{if } |\nabla\cdot(h\mathbf{u})| \geq B_2\sqrt{gh} \\ 0 & \text{else} \end{cases} \tag{2.0.15}$$

For the wave breaking, we need two additional parameters: B_1 is the wave breaking start parameter. In [15], it is chosen as $B_1 = 0.5$. B_2 is the wave breaking end parameter, and we require $B_1 > B_2$. In [15], it is chosen as $B_1 = 0.15$. In addition, we define $e_B \in \{0,1\}$ as a switch to enable/disable the wave breaking. The rule used in [15] is to set $e_B = 1$, once $|\nabla\cdot(h\mathbf{u})| \geq B_1\sqrt{gh}$ and set $e_B = 0$, once $|\nabla\cdot(h\mathbf{u})| < B_2\sqrt{gh}$. This is subsequently re-evaluated at each time.

Finally, we impose to positivity conditions onto the system, namely the positivity of water height which can be written as

$$h \geq 0, \tag{2.0.16}$$

as well as that the system has positive eigenvalues when writing it in matrix form. This is achieved, if

$$C := gh + p + c^2 \geq 0. \tag{2.0.17}$$

For hyperbolicity, we need $C > 0$, as we will see further below. Given these, we define the hyperbolicity domain

$$\Pi_d = \left\{ U \in \mathbb{R}^{d+4} \;\middle|\; h > 0;\ gh + p + c^2 > 0 \right\}. \tag{2.0.18}$$

For $d = 1$, the equations without vector notation look as follows:

$$\partial_t \begin{pmatrix} h \\ hu \\ hw \\ hp \end{pmatrix} + \partial_x \begin{pmatrix} hu \\ hu^2 + hp \\ huw \\ hu(p + c^2) \end{pmatrix} + \begin{pmatrix} 0 \\ gh(\partial_x h + \partial_x b) + \gamma p \partial_x b \\ 0 \\ -c^2 u(\partial_x h + 2\partial_x b) \end{pmatrix} = \begin{pmatrix} 0 \\ -\tau_b \\ \gamma p + R_b(\mathbf{U}, \nabla \mathbf{U}) \\ -2c^2 w \end{pmatrix}. \tag{2.0.19}$$

Throughout this work, we will often write the vector $\mathbf{U}$ with a subscript or diacritic, e.g. $\mathbf{U}_0$, $\mathbf{U}_L$, $\mathbf{U}_R$, or $\hat{\mathbf{U}}$. In these cases, the variable h_0, refers to the first element of $\mathbf{U}_0$, and $(hu)_0$ to the second element of $\mathbf{U}_0$, and similarly for the others. This does sadly create a small overload of notation, since $\mathbf{U}_L$ and $\mathbf{U}_R$ will usually refer to state vectors, and not functions.[3]

2.1 Structure of the Equations

We will shortly highlight some aspects of the model, as they are also reported by [15].

2.1.1 Special Cases

Firstly, we consider some special cases, when setting the variables to specific values. Some of them have already been mentioned, but we will look on how to get back to them from (2.0.10).

Shallow Water Equations

If we set $c = 0$, $w(0, \cdot) \equiv 0$, and $p(0, \cdot) \equiv 0$, we retrieve the shallow water equations with friction term and bathymetry: the hw equation becomes 0 since all its terms depend on w. Additionally, the hp equation becomes 0, since all terms either depend on p or c. Finally, we disable the wave-breaking term permanently. So we are left with:

$$\begin{aligned} \partial_t h + \nabla \cdot (h\mathbf{u}) &= 0, \\ \partial_t (h\mathbf{u}) + \nabla \cdot (h\mathbf{u}\mathbf{u}^T) + \nabla \left(\frac{g}{2} h^2 \right) &= -gh\nabla b - \tau_b. \end{aligned} \tag{2.1.1}$$

In comparison to the introduction, this version of the shallow water equations contains The shallow water equations form an important special case of (2.0.10), and they have been studied and simulated in much more detail than the system we deal with here. For a treatment of the shallow water equations, see e.g. [24, 37, 28].

[3] This is inspired by [15, eq. (20)], where that notation is used for the numerical flux.

BMSS and Green-Naghdi System

Secondly, we may retrieve the system that the equation 2.0.10 is based on. That is, we get back to the BMSS-γ system [15, eq. (1)] which is defined in [3, eq. (53)] and already considered in the introduction to this chapter. In [16], it is called "BMSS" (if $\gamma = 2$) after the initials of the authors of the paper [3], and Green-Naghdi if $\gamma = 3/2$ (see also [25, eqs. (1.45), (1.46)] for the latter). For completeness, we will write the equation system as a whole here. It is taken from [3, eq. (53)], but generalized to arbitrary dimensions. The system reads

$$\begin{aligned}
\partial_t h + \nabla \cdot (h\mathbf{u}) &= 0 \\
\partial_t (h\mathbf{u}) + \nabla \cdot (h\mathbf{u}\mathbf{u}^T) + \nabla \left(\frac{g}{2}h^2 + hp\right) &= -(\gamma p + gh)\nabla b - \tau_b \\
\partial_t (hw) + \nabla \cdot (h\mathbf{u}w) &= -\gamma p \\
\nabla \cdot \mathbf{u} + \frac{w - (\mathbf{u} \cdot \nabla)b}{h/2} &= 0.
\end{aligned} \tag{2.1.2}$$

We may re-retrieve it from 2.0.10 in the limit, as mentioned in [15]. This is done by sending $c \to \infty$. To do so, we divide the fourth equation by hc^2 first, then we have

$$\frac{\partial_t (hp)}{hc^2} + \frac{\nabla \cdot (h\mathbf{u}p)}{hc^2} + \left(\nabla \cdot \mathbf{u} + \frac{w - (\mathbf{u} \cdot \nabla)b}{h/2}\right) = 0. \tag{2.1.3}$$

And so, in the limit $c \to \infty$, we obtain again

$$\nabla \cdot \mathbf{u} + \frac{w - (\mathbf{u} \cdot \nabla)b}{h/2} = 0. \tag{2.1.4}$$

Generalization of the Model

In addition to special cases, there is also an even more general formulation which aims at capturing more different shallow-water-like models [16]. It reads

$$\begin{aligned}
\partial_t h + \nabla \cdot (h\mathbf{u}) &= 0, \\
\partial_t (h\mathbf{u}) + \nabla \cdot (h\mathbf{u}\mathbf{u}^T) + \nabla \left(\frac{g}{2}h^2 + hp\right) &= -(gh + 2(p-q))\nabla b - \tau_b, \\
\partial_t (hw) + \nabla \cdot (h\mathbf{u}w - gh^2\zeta) + 2gh(\zeta \cdot \nabla h) &= 2(p-q) + \kappa w, \\
\partial_t (h\xi) + \nabla \cdot \left(h\mathbf{u}\xi - \frac{g}{2}h^2\zeta\right) + gh(\zeta \cdot \nabla h) &= q + \kappa\xi, \\
\partial_t (hp) + \nabla \cdot (h\mathbf{u}p) + \gamma_1 c^2 (w + \mu h(\nabla \cdot \mathbf{u})) &= \gamma_1 c^2 \mu_B \left((u \cdot \nabla)b + \partial_t b\right), \\
\partial_t (hq) + \nabla \cdot (h\mathbf{u}q) + \gamma_2 c^2 (\xi - \beta h(\nabla \cdot \mathbf{u})) &= 0, \\
\partial_t (h\zeta) + \nabla \cdot (h\mathbf{u}\zeta^T) + \nabla \left(\frac{B}{\beta}h\xi\right) &= 0.
\end{aligned} \tag{2.1.5}$$

Here, h is the water height, $\mathbf{u}$ is the horizontal velocity, w is the vertical velocity, and p is the non-hydrostatic pressure, just an in [3] and [15]. ξ, q, and ζ are additional helper

variables. In particular, h, w, ξ, p, q are scalar-valued variables, and $\mathbf{u}, \zeta$ are vector-valued. [4] In addition, c, B, β, γ_1, γ_2, μ_B, and μ are all scalar constants. If $\beta = 0$, we set $\frac{B}{\beta} = 0$. Furthermore, κ is a wave-breaking parameter similar to R_b [16, Section 3.2].

To retrieve (2.0.10) with $\gamma = 2$, we have to set $B = 0$, $\beta = 0$, $\gamma_1 = 2$, $\gamma_2 = 0$, $\mu_B = 2$, $\mu = 1$, and $\partial_t b = 0$ for all t.

2.1.2 Derivation of the Model

We will shortly outline how to get to the BMSS-γ system. The derivation process for this system is shown in [3] (different derivation processes are shown in [16], or [25]). Since it is described in detail there, we will only sketch it shortly in one dimension. While the introduction to this chapter already intuitively motivated on how to construct the equations, we will look into the actual process on how the equations were obtained in the paper [3].

Depth Averaging

We begin with the Euler equations which are integrated in the vertical direction, and averaged. This means that we have given at each position the bathymetry $b(x)$, and our water level $\eta(x)$. Based on this, we can define the density

$$\varphi(x,z) = \begin{cases} 1 & -b(x) \leq z \leq \eta(x) \\ 0 & \text{otherwise} \end{cases} \tag{2.1.6}$$

as well as the depth-dependent variables $U(x,z)$ (horizontal velocity), $W(x,z)$ (vertical velocity) and $P(x,z)$ (pressure). Our model variables are then derived as

$$h = \int_{\mathbb{R}} \varphi(z)\, \mathrm{d}z = \int_b^\eta 1\, \mathrm{d}z = \eta - b \tag{2.1.7}$$

and

$$u = \frac{\int_b^\eta U(z)\, \mathrm{d}z}{h},\ w = \frac{\int_b^\eta W(z)\, \mathrm{d}z}{h},\ p_T = \frac{\int_b^\eta P(z)\, \mathrm{d}z}{h} \tag{2.1.8}$$

In addition, we impose the kinematic boundary conditions at both the water surface and the bottom. This means

$$\partial_t b + U_b(\nabla \cdot b) - W_b = 0, \tag{2.1.9}$$

$$\partial_t \eta + U_\eta(\nabla \cdot \eta) - W_\eta = 0. \tag{2.1.10}$$

Here, $U_b(x) = U(x, b(x))$ and $U_\eta(x) = U(x, \eta(x))$, W_b and W_η are defined like-wise. We assume $\partial_t b = 0$.

[4] In [16], the system (2.1.5) is only derived for $d = 1$, so that $\mathbf{u}$ and ζ are also scalar-valued. However, for $d > 1$, naturally $\mathbf{u}$ is a vector. In addition, w and p are also scalar by construction. ζ is introduced to decompose the Laplace operator, so that $B\Delta\eta = B(\nabla \cdot \nabla\eta) = \nabla \cdot \zeta$. Naturally, ζ must be vector-valued as a result and have the same dimension as $\mathbf{u}$. The variables ξ and q are both scalar-valued as well.

Pressure Decomposition

In contrast to the shallow water equations, the pressure P is decomposed into a hydrostatic and a non-hydrostatic component:[5]

$$P(z) = g(\eta - z) + P_{nh}(z) \tag{2.1.11}$$

The first addend is the hydrostatic part, the second addend the non-hydrostatic pressure. We subsequently define

$$p_T := \frac{\int_b^\eta P(z)\,\mathrm{d}z}{h} \tag{2.1.12}$$

Expanding this gives:

$$hp_T = g\left(\eta(\eta - b) - \frac{\eta^2 - b^2}{2}\right) + \int_b^\eta P_{nh}(z)\,\mathrm{d}z = \frac{gh^2}{2} + hp \tag{2.1.13}$$

Inserting (2.1.11) into the depth-averaged system and integrating by parts, we also get terms of the form

$$P_{nh,\eta} - P_{nh,b} = P_{nh}(\,\cdot\,, \eta(\,\cdot\,)) - P_{nh}(\,\cdot\,, b(\,\cdot\,)) \tag{2.1.14}$$

which have to be given a meaning, since they depend on the vertical position.

Energy Minimization

The third step is to minimize the depth-averaged energy from the Navier-Stokes system, i.e. we minimize

$$\int_{\mathbb{R}} \varphi\left(\frac{U(z)^2 + W(z)^2}{2} + gz\right)\mathrm{d}z \tag{2.1.15}$$

w.r.t. U and W. Solving that yields us that U and W should be constant in z. All that is left now is to choose a value for $P_{nh,\eta} - P_{nh,b}$, such that an additional energy balance law is fulfilled, see [3, Proposition 1]. Thus, we get to the choice

$$P_{nh,\eta} - P_{nh,b} = \gamma p \tag{2.1.16}$$

with $\gamma = 2$. This is in fact the only place, where γ appears, and it is the only parameter we have to vary to get different models, e.g. [3, eqs. (49), (50)]. This gives us (2.1.2).

2.2 Basic Properties

Next, we discuss some simple properties of the equation system (2.0.10).

However, we will note that we do not discuss the general well-posedness for (2.0.10) in this work at all, since it was unfortunately out of scope. For a discussion of well-posedness discussions in the context of DG methods, we refer e.g. to [10].

[5] Actually, we would have the atmospheric pressure p_a as a third component. However, we assume $p_a = 0$.

2.2.1 Re-Writing in Vector Notation

Firstly, we introduce an additional notation for the equation system, just as it is done in [15]. We begin by fixing a space dimension $d \in \{1, 2\}$. Higher space dimensions would be possible, but they will probably not make any physical sense. After fixing d, we use the following form:

$$\partial_t \mathbf{U} + \nabla \cdot \mathbf{F}(\mathbf{U}) + \mathbf{B}(\mathbf{U}) \cdot \nabla \mathbf{U} = \mathbf{S}(\mathbf{U}, \nabla \mathbf{U}). \tag{2.2.1}$$

Here, again we have the vector of conservative variables given as

$$\mathbf{U} = \begin{pmatrix} h \\ h\mathbf{u} \\ hw \\ hp \\ b \end{pmatrix}, \tag{2.2.2}$$

and we also have included b into it, since we can just add $\partial_t b = 0$ as equation to (2.0.10), in order to get a system which contains all variables. For given $x \in \mathbb{R}^d$ and $t \in [0, \infty)$, we have $\mathbf{U}(t, x) \in \mathbb{R}^{d+4}$, since $\mathbf{u}(t, x) \in \mathbb{R}^d$. The gradient $\nabla \mathbf{U}$ is understood as (shown for $d = 2$)

$$\nabla \mathbf{U} = \begin{pmatrix} \partial_x \mathbf{U} \\ \partial_y \mathbf{U} \end{pmatrix}. \tag{2.2.3}$$

Then, we define for each space direction i, the functions $\mathbf{F} = (\mathbf{F}_1, ..., \mathbf{F}_d)$, and $\mathbf{B} = (\mathbf{B}_1, ..., \mathbf{B}_d)$. In particular, for all i, we have $\mathbf{F}_i : \Pi_d \to \mathbb{R}^{d+4}$, and $\mathbf{B}_i : \Pi_d \to \mathbb{R}^{(d+4)\times(d+4)}$. In both cases, we needed to restrict the area of definition to $h > 0$, since we are dealing with primitive variables as well. They both read in vector notation, where $\mathbf{u}_i$ is the i-th component of $\mathbf{u}$,

$$\mathbf{F}_i(\mathbf{U}) = \begin{pmatrix} h\mathbf{u}_i \\ h(\mathbf{u}\mathbf{u}_i + pI) \\ hw\mathbf{u}_i \\ h(p + c^2)\mathbf{u}_i \\ 0 \end{pmatrix}, \tag{2.2.4}$$

and

$$\mathbf{B}_i(\mathbf{U}) = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ ghI & 0 & 0 & 0 & (gh + \gamma p)I \\ 0 & 0 & 0 & 0 & \\ c^2\mathbf{u}_i & 0 & 0 & 0 & 2c^2\mathbf{u}_i \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{2.2.5}$$

Note that in particular that each $\mathbf{B}_i$ is linear in the primitive variables $h, \mathbf{u}, w, p$. The source term $\mathbf{S} : \Pi_d \times \Pi_d \to \mathbb{R}^{d+4}$ reads

$$\mathbf{S}(\mathbf{U}, \nabla \mathbf{U}) = \begin{pmatrix} 0 \\ \tau_b(h, \mathbf{u}) \\ \gamma p + R_b(\mathbf{U}, \nabla \mathbf{U}) \\ -2c^2 w \end{pmatrix}. \tag{2.2.6}$$

Given $\mathbf{F}$ and $\mathbf{B}$, we write the multiplication with a vector $v \in \mathbb{R}^d$ as

$$\mathbf{F} \cdot v := \sum_{i=1}^{d} \mathbf{F}_i v_i \tag{2.2.7}$$

$$\mathbf{B} \cdot v := \sum_{i=1}^{d} \mathbf{B}_i v_i \tag{2.2.8}$$

For the scalar product with the gradient, we write in accordance to what we wrote above

$$\mathbf{B}(\mathbf{U}) \cdot \nabla \mathbf{U} = \sum_{i=1}^{d} \mathbf{B}_i(\mathbf{U}) \partial_i \mathbf{U}. \tag{2.2.9}$$

Here ∂_i denotes the partial derivative in the i-th cardinal direction.

We may now define $\mathbf{A} = (\mathbf{A}_1, ..., \mathbf{A}_d)$ component-wise as

$$\mathbf{A}_i(\mathbf{U}) = D\mathbf{F}_i(\mathbf{U}) + \mathbf{B}_i(\mathbf{U}) \tag{2.2.10}$$

where $D\mathbf{F}_i$ denotes the Jacobian matrix of $\mathbf{F}_i$. The product $\mathbf{A} \cdot v$ and $\mathbf{A}(\mathbf{U}) \cdot \nabla \mathbf{U}(\mathbf{U})$ is defined like-wise. For the matrix $\mathbf{A}(\mathbf{U})$ without bathymetry, we write $\mathbf{A}^*(\mathbf{U})$. We close with some examples for small d. In particular, for $d = 1$, we obtain

$$\mathbf{A}_1^*(\mathbf{U}) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ gh - u^2 & 2u & 0 & 1 \\ -uw & w & u & 0 \\ -(c^2 + p)u & c^2 + p & 0 & u \end{pmatrix}. \tag{2.2.11}$$

For $d = 2$, we have

$$\mathbf{A}_1^*(\mathbf{U}) = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ gh - u^2 & 2u & 0 & 0 & 1 \\ -uv & v & u & 0 & 0 \\ -uw & w & 0 & u & 0 \\ -(c^2 + p)u & c^2 + p & 0 & 0 & u \end{pmatrix}, \tag{2.2.12}$$

and

$$\mathbf{A}_2^*(\mathbf{U}) = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ -vu & v & u & 0 & 0 \\ gh - v^2 & 0 & 2v & 0 & 1 \\ -vw & 0 & w & v & 0 \\ -(c^2 + p)v & 0 & c^2 + p & 0 & v \end{pmatrix}. \tag{2.2.13}$$

For $\mathbf{A}_1$ in $d = 1$, we have

$$\mathbf{A}_1(\mathbf{U}) = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ gh - u^2 & 2u & 0 & 1 & \gamma p + gh \\ -uw & w & u & 0 & 0 \\ -(c^2 + p)u & c^2 + p & 0 & u & -2c^2 u \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{2.2.14}$$

2.2.2 Rotational Invariance

Next, we want to discuss and simplify a particular special case, namely one-dimensional problems. This section is inspired by [43, Chapter 3]. Suppose that we are given a normal vector $n \in \mathbb{R}^d$, so that it holds $\|n\|_2 = 1$. Then, suppose that $R \in SO(d)$ is a rotation matrix, so that we have $Rn = e_1$, where e_1 is the first unit vector. Define then $T \in \mathbb{R}^{(d+4)\times(d+4)}$ as

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & R & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{2.2.15}$$

The goal of this section is to replace $(\mathbf{A} \cdot n)$ by applications of T and $T^T = T^{-1}$ and $\mathbf{A}_1$. In particular, we want to ignore $\mathbf{A}_i$ for $i > 1$, because this simplifies many operations computationally. We get the following result.

Proposition 2.1 (Rotational Invariance)**.** *It holds for all* $\mathbf{U} \in \mathbb{R}^{d+4}$ *that*

$$T^T \mathbf{F}_1(T\mathbf{U}) = \mathbf{F}(\mathbf{U}) \cdot n \tag{2.2.16}$$

$$T^T \mathbf{B}_1(T\mathbf{U})T = \mathbf{B}(\mathbf{U}) \cdot n \tag{2.2.17}$$

$$T^T \mathbf{A}_1(T\mathbf{U})T = \mathbf{A}(\mathbf{U}) \cdot n \tag{2.2.18}$$

We will only mention the main idea here. It is to consider the rows concerning h, hw, hp, and b separately from the rows concerning $h\mathbf{u}$. The rows of h, hw, hp, and b depend linearly on $\mathbf{u}$, while the rows of $h\mathbf{u}$ are the sum of a term that is quadratic in $\mathbf{u}$ and a term that does not depend on $\mathbf{u}$.

2.2.3 Steady States

Next, we consider steady states, that is, initial states for which we have $\partial_t \mathbf{U} = 0$ in time. Mainly, we consider two of them (as mentioned in [36, eqs. (5.88), (5.89)]), both of which our equation conforms with, as the following definition and proposition show.

Definition 2.2 (Two Steady States)**.** *Suppose that we have one of the following initial conditions:*

- *Dry/Vacuum Condition:* $\mathbf{U}(0, \cdot) \equiv 0$
- *Resting-Lake Condition (also called C-Property [25]):* $\eta(0, \cdot) \equiv const$*, and* $\mathbf{u}(0, \cdot) \equiv 0$*,* $w(0, \cdot) \equiv 0$*,* $p(0, \cdot) \equiv 0$*.*

Then $\mathbf{U}(t, x) = \mathbf{U}(0, x)$ *for* $t > 0$ *is a solution of the equation.*

Proposition 2.3. *Equation* (2.0.10) *fulfills both the Dry and the Resting-Lake Condition.*

Proof. For the vacuum condition: we note that inserting $\mathbf{U}_0 = 0$ into the equation immediately gives $\partial_t \mathbf{U} = 0$.

For the resting-lake condition: inserting the initial conditions $\mathbf{u} = 0$, $w = 0$, and $p = 0$ into (2.0.10) gives for all five terms

$$\partial_t \mathbf{U} + 0 + \begin{pmatrix} 0 \\ \nabla(\frac{1}{2}gh^2) \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ -(gh)\nabla b \\ 0 \\ 0 \end{pmatrix} + 0. \tag{2.2.19}$$

Next, since by the chain rule we get $\nabla(\frac{1}{2}gh^2) = gh\nabla h$, we get the relation

$$gh\nabla(h + b) = gh\nabla\eta = 0. \tag{2.2.20}$$

Hence, we get $\partial_t \mathbf{U} = 0$. □

2.2.4 Eigenstructure and Hyperbolicity

In the following, we will deal with the two-dimensional case $d = 2$, and we will ignore bathymetry for now.

If $C > 0$, then $\mathbf{A}_1^*(\mathbf{U})$ is diagonalizable, and we get the following eigenvectors and eigenvalues:

$$\lambda_1 = u - \sqrt{C},\ \lambda_2 = \lambda_v = \lambda_w = u,\ \lambda_3 = u + \sqrt{C}, \tag{2.2.21}$$

$$r_1 = \begin{pmatrix} 1 \\ u - \sqrt{C} \\ v \\ w \\ p + c^2 \end{pmatrix}, r_2 = \begin{pmatrix} 1 \\ u \\ 0 \\ 0 \\ -gh \end{pmatrix}, r_3 = \begin{pmatrix} 1 \\ u + \sqrt{C} \\ v \\ w \\ p + c^2 \end{pmatrix}, r_v = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, r_w = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}. \tag{2.2.22}$$

The eigenvalues, when ordered, fulfill

$$\lambda_1 \leq \lambda_2 = \lambda_v = \lambda_w \leq \lambda_3. \tag{2.2.23}$$

For $d = 1$, λ_v and r_v disappears, and the third component is erased in all remaining eigenvectors. Other than that, nothing changes compared to the eigenvectors that are written for $d = 2$. If $C = 0$, we lose the hyperbolicity: then u is the only eigenvalue, and $r_1 = r_3 = r_2 + vr_v + wr_w$. This leaves us with an eigenspace of dimension 3.

Note in particular that r_1 and r_3 do not correspond with the eigenvalues presented in the paper [15, eq. (10)], nor are they a scalar multiple of them: the last component of the corresponding vectors r_1 and r_3 in [15, eq. (10)] is 0. Since both the eigenspaces associated with λ_1 and λ_3 are one-dimensional, there is no possibility that the vectors could still be linearly dependent. We should also note that our computation of r_1 and r_3 conforms with the special case of taking the shallow water equations, where $p + c^2 = 0$, hence $\sqrt{C} = \sqrt{gh}$. By that, the second component of r_1 becomes $u - \sqrt{gh}$, and r_3 becomes $u + \sqrt{gh}$ respectively. In comparison to that, [15, eq. (10)] yields 0 for the second component in this case, and only a non-zero value, if $p + c^2 \neq 0$ and $w \neq 0$.

Characteristic Fields

Next, we may examine the characteristic field associated with each eigenvector (cf. Definition 2.17). For this, we compute

$$\nabla\lambda_{2,v,w} = \nabla\left(\frac{hu}{h}\right) = \frac{1}{h}\begin{pmatrix}-u\\1\\0\\0\\0\end{pmatrix}, \quad \nabla\lambda_{1,3} = \frac{1}{h}\begin{pmatrix}-u \pm \frac{gh-p}{2\sqrt{C}}\\1\\0\\0\\\pm\frac{1}{2\sqrt{C}}\end{pmatrix}. \tag{2.2.24}$$

Then, we compute $\nabla\lambda_i \cdot r_i$ for all $i = 1, 2, 3$ and $i = v$, $i = w$. This gives us:

$$\nabla\lambda_2 \cdot r_2 = \nabla\lambda_v \cdot r_v = \nabla\lambda_w \cdot r_w = 0 \tag{2.2.25}$$

And as long as $C > 0$, we have

$$\nabla\lambda_{1,3} \cdot r_{1,3} = -u \pm \frac{gh-p}{2\sqrt{C}} + u \pm C \pm \frac{p+c^2}{2\sqrt{C}} = \pm\frac{2C+gh+c^2}{2\sqrt{C}} \neq 0 \tag{2.2.26}$$

since $gh + c^2 \geq 0$. So, the characteristic fields associated with $\lambda_2, \lambda_v, \lambda_w$ are linearly degenerate, while the fields associated with λ_1 and λ_3 are genuinely nonlinear.

Adding Bathymetry

Next, we examine the matrix $\mathbf{A}_1$ which includes the bathymetry component. It turns out that the eigenvalues and eigenvectors of $\mathbf{A}_1^*$ are also eigenvalues and eigenvectors of $\mathbf{A}_1$, when adding a 0 in the b-component. In addition, we get a new eigenvalue and eigenvector

$$\lambda_b = 0, \; r_b = \begin{pmatrix}2c^2 + gh + \gamma p\\0\\\left(2c^2 + gh + \gamma p\right) v\\\left(2c^2 + gh + \gamma p\right) w\\2c^2u^2 + gh(p - c^2) + \gamma p(p + c^2)\\u^2 - C\end{pmatrix}. \tag{2.2.27}$$

As we see immediately by $\lambda_b = 0$, the associated characteristic field is linearly degenerate. For the hyperbolicity, the cases $\lambda_i = 0 = \lambda_b$ for $i = 1, 2, 3$ are of interest.

- If $\lambda_1 = \lambda_6$, or $\lambda_3 = \lambda_b$, (and $C > 0$) we get $u^2 = C$. Thus, the last component of rb becomes 0. As a result, the system loses its hyperbolicity, since r_b can now be represented as a linear combination of r_1, r_2, r_3, r_v, r_w.
- If $\lambda_2 = \lambda_v = \lambda_w = \lambda_b$, but $C > 0$, then we get $u = 0$. Yet, we do not lose the hyperbolicity, since the last component of r_b is still not equal to zero. Therefore, it is not linearly dependent to r_2, r_v, r_w.

- If $C = 0$ and $u = 0$, then all eigenvalues conflate: we once again get that the eigenspace has only dimension 3, since we especially also have $\lambda_1 = \lambda_3 = \lambda_b$.

Subsequently, if our system is hyperbolic, we have $u^2 \neq C$. Then, we may re-parametrize r_b to

$$\tilde{r}_b = \begin{pmatrix} \frac{2c^2+gh+\gamma p}{u^2-C} \\ 0 \\ \frac{2c^2+gh+\gamma p}{u^2-C} v \\ \frac{2c^2+gh+\gamma p}{u^2-C} w \\ \frac{2c^2u^2+gh(p-c^2)+\gamma p(p+c^2)}{u^2-C} \\ 1 \end{pmatrix}. \tag{2.2.28}$$

About Hyperbolicity

Since we now have all of these results, we can prove the hyperbolicity of the system in arbitrary dimensions, and with bathymetry. The following proposition is similar as it is shown in [43].

Proposition 2.4. *The system* (2.2.1) *is hyperbolic (when ignoring* $\mathbf{S}$*) at* $\mathbf{U}$*, as long as* $C > 0$ *and* $\|\mathbf{u}\|_2^2 \neq C$*. This means that for such* $\mathbf{U}$*, and all normal vectors* $n \in \mathbb{R}^d$ *(i.e.* $\|n\|_2 = 1$*), we have that*

$$\mathbf{A}(\mathbf{U}) \cdot n \tag{2.2.29}$$

is diagonalizable with real eigenvalues.

Proof. Let $n \in \mathbb{R}^d$ be a normal vector, and R and T be defined as in the section about rotational invariance (Section 2.2.2). By Proposition 2.1, we obtain

$$\mathbf{A}(\mathbf{U}) \cdot n = T^T \mathbf{A}_1(T\mathbf{U})T. \tag{2.2.30}$$

In addition, we have shown that $\mathbf{A}_1(T\mathbf{U})$ is in fact diagonalizable, since $C > 0$, and $u^2 \neq C$. At least we have shown this for $d = 2$, but the result can be generalized: all other dimensions behave like v in $\mathbf{A}_1$ then. Since R is a rotation matrix, we have that $(R\mathbf{u})_1 = \|\mathbf{u}\|$. So, let $D(\mathbf{U}), R(\mathbf{U})$ be matrices, so that $D(\mathbf{U})$ is diagonal and $R(\mathbf{U})$ is invertible (they both exist, since $\mathbf{A}_1(T\mathbf{U})$ is diagonalizable), and

$$\mathbf{A}_1(\mathbf{U}) = R(\mathbf{U})D(\mathbf{U})(R(\mathbf{U}))^{-1}. \tag{2.2.31}$$

And thus, we obtain

$$\begin{aligned} \mathbf{A}(\mathbf{U}) \cdot n &= T^T(R(T\mathbf{U}))D(T\mathbf{U})R(T\mathbf{U})^{-1}T \\ &= (T^T R(T\mathbf{U}))D(T\mathbf{U})(T^T R(T\mathbf{U}))^{-1}. \end{aligned} \tag{2.2.32}$$

This shows that $\mathbf{A}(\mathbf{U})\cdot n$ is diagonalizable for all n, and thus the system is hyperbolic. □

Furthermore, the same technique makes it possible for us to say something about the maximal eigenvalue of $\mathbf{A} \cdot n$.

Proposition 2.5. *If $C > 0$ and $\|\mathbf{u}\|^2 \neq C$ for $\mathbf{U}$, then we have that*

$$\lambda_{\max}(\mathbf{U}) = \max_{\|n\|_2=1} \|\mathbf{A}(\mathbf{U}) \cdot n\|_2 = \|\mathbf{u}\|_2 + \sqrt{gh + p + c^2}. \tag{2.2.33}$$

Here, $\|\mathbf{A}(\mathbf{U}) \cdot n\|_2$ denotes the maximal singular value of $(\mathbf{A}(\mathbf{U}) \cdot n)$.

Proof. We again apply Proposition 2.1. For clarity, let R_n and T_n be the rotation matrices associated with the normal vector n. We get that

$$\|\mathbf{A}_1(T_n\mathbf{U})\|_2 = \max\{|\lambda_1(T_n\mathbf{U}))|, |\lambda_3(T_n\mathbf{U}))|\} = |(R_n\mathbf{u})_1| + \sqrt{gh + p + c^2}, \tag{2.2.34}$$

where the maximum singular value of $\mathbf{A}_1(T\mathbf{U})$ conflates with the largest absolute eigenvalue, since $\mathbf{A}_1(T\mathbf{U})$ is diagonalizable. Next, we take the maximum over all normal vectors $n \in \mathbb{R}^d$, and thus over all possible matrices T. Firstly, the maximum exists, since $\mathbb{R}^d$ is finite-dimensional, so the unit circle is compact. Secondly, by the Cauchy-Schwarz equation, we obtain (using that we set $R_n e^1 = n$)

$$|(R_n\mathbf{u})_1| = \left|(e^1)^T R_n\mathbf{u}\right| = \left|n^T\mathbf{u}\right| \leq \|n\|_2 \|\mathbf{u}\|_2 = \|\mathbf{u}\|_2 . \tag{2.2.35}$$

Equality is achieved on the unit circle, by the choice

$$n = \frac{1}{\|\mathbf{u}\|_2}\mathbf{u}, \tag{2.2.36}$$

given $\mathbf{u} \neq 0$ (otherwise we can take any normal vector we like). Thus, we obtain

$$\lambda_{\max}(\mathbf{U}) = \max_{\|n\|_2=1} \left\|T_n^T \mathbf{A}_1(T_n\mathbf{U})T_n\right\|_2 = \max_{\|n\|_2=1} \|\mathbf{A}_1(T_n\mathbf{U})\|_2 = \|\mathbf{u}\|_2 + \sqrt{gh + p + c^2}. \tag{2.2.37}$$

□

In particular, we may note that $\lambda_{\max}$ is strictly monotonically increasing in h, p, and $\|u\|$.

2.3 Solitary Waves

Next, we are concerned with deriving some stationary solutions which we will later-on use as test cases.

2.3.1 Solitary Wave Solution for the BMSS system

Before we begin to derive such a system for (2.0.10), we mention the solitary wave solution which was given for (2.1.2) in [3, eq. (69)] for arbitrary γ. We define

$$\xi = \frac{x - c_A t}{l}, \tag{2.3.1}$$

where A is the amplitude of the solitary wave, H is the assumed stillwater height. While [3] also defines a convergence water height d, we will in this section assume that $d = H$ for simplicity. The re-scaling factor l is given as

$$l = H\sqrt{\frac{2}{\gamma A}(A+H)}, \tag{2.3.2}$$

as well as the wave propagation speed

$$c_A = \frac{l}{H}\sqrt{\frac{\gamma g A}{2}} = \sqrt{g(A+H)}. \tag{2.3.3}$$

Then, the solution reads as follows (cf. [3, eq. (69)])

$$\begin{aligned} h &= H + A(\operatorname{sech}(\xi))^2 \\ u &= c_A \frac{h-H}{h} \\ w &= -\frac{Ac_A H}{lh}\operatorname{sech}(\xi)\operatorname{sech}'(\xi) \\ p &= \frac{Ac_A^2 H^2}{2l^2h^2}\left((2H-h)\left(\operatorname{sech}'(\xi)\right)^2 + h\operatorname{sech}(\xi)\operatorname{sech}''(\xi)\right). \end{aligned} \tag{2.3.4}$$

2.3.2 Solitary Wave for the Governing Equations

A similar process can be done to obtain an at least quasi-exact solution for (2.0.10), as it is done in Section 5.1.1 in [15]. However, the derivation done there contains (at least) two errors in the relevant formulas which is why we will re-do this process in this work. We assume $n_m = 0$ and wave breaking to be disabled. Also, we consider a one-dimensional plane, and we ignore bathymetry.

Next, we re-use the l as defined in the previous section[6] which means that we use (2.3.2). The parameter c_A will be fixed later-on. It should be noted though that c_A is not the same parameter as c from (2.0.10). With all of that, we can define ξ as in (2.3.1).

Re-Derivation of the Solitary Wave ODE System

For that, we suppose that h, hu, hw, hp only depend on ξ. Therefore, we have

$$\mathbf{U}(\xi) = \begin{pmatrix} h(\xi) \\ hu(\xi) \\ hw(\xi) \\ hp(\xi) \end{pmatrix}. \tag{2.3.5}$$

[6] This is where the first error lies in [15]. This parameter is defined as $l = \sqrt{H(H+A)}$, yet it is written redundantly as $H\sqrt{(H+A)/H}$. Thus, we assume that this should actually mean $l = H\sqrt{(H+A)/A}$ which would give us the same as the l from [3] with $\gamma = 2$ as in (2.3.2).

For brevity, we write $\mathbf{U}' = \partial_\xi \mathbf{U}$ (just as in [15]). Then, we get the relations

$$\partial_x \mathbf{U} = \mathbf{U}' \cdot \frac{1}{l} \tag{2.3.6}$$

$$\partial_t \mathbf{U} = \mathbf{U}' \cdot (-c_A)\frac{1}{l} \tag{2.3.7}$$

by the chain rule. Hence, we may re-write (2.2.1) to

$$\frac{1}{l}\left(-c_A \mathbf{U}' + \mathbf{A}(\mathbf{U})\mathbf{U}'\right) = \mathbf{S}\left(\mathbf{U}, \frac{1}{l}\mathbf{U}'\right). \tag{2.3.8}$$

This can in turn be rewritten to

$$(\mathbf{A}(\mathbf{U}) - c_A I)\mathbf{U}' = l\mathbf{S}\left(\mathbf{U}, \frac{1}{l}\mathbf{U}'\right), \tag{2.3.9}$$

as written in the paper as well. Expanding $\mathbf{A}$ and $\mathbf{S}$ gives us the ODE system (again, we have $b' \equiv 0$)

$$\begin{aligned} (hu)' - c_A h' &= 0 \\ -u^2 h' + 2u(hu)' + (hp)' + ghh' - c_A(hu)' &= 0 \\ -uwh' + w(hu)' + u(hw)' - c_A(hw)' &= \gamma pl \\ -uph' + p(hu)' + u(hp)' + c^2(hu)' - c^2 uh' - c_A(hp)' &= -2c^2 wl. \end{aligned} \tag{2.3.10}$$

We have the initial conditions $h(0) = H + A$, and $w(0) = 0$. In addition, we impose boundary conditions for $\xi \to \infty$. So, we get[7]

$$\begin{aligned} h(\xi) &\to H \\ u(\xi) &\to 0 \\ w(\xi) &\to 0 \\ p(\xi) &\to 0. \end{aligned} \tag{2.3.11}$$

Since we now have an under-determined system, we choose c_A as our last free parameter. It will be determined later on, but it will be not the same as in [15].

To begin with, we integrate the first equation to get

$$hu - c_A h = C_h \tag{2.3.12}$$

for some constant C_h. Inserting the asymptotic conditions for h and u, we obtain

$$C_h = -c_A H, \tag{2.3.13}$$

and thus we get an expression for hu. It reads

$$hu = c_A(h - H). \tag{2.3.14}$$

[7] This is the place where we set $d = H$. If not, we would have $h \to d$ here, according to [3].

This is the same expression as in [15, eq. (29)]. As an expression for u, we get immediately

$$u = c_A \left(1 - \frac{H}{h}\right). \tag{2.3.15}$$

Next, we use this and the first equation re-write the second equation in terms of h' and h. This gives us

$$-c_A^2 \left(1 - \frac{H}{h}\right)^2 h' + 2c_A^2 \left(1 - \frac{H}{h}\right) h' + (hp)' + ghh' - c_A^2 h' = 0. \tag{2.3.16}$$

Now we put $(hp)'$ on one side to get

$$(hp)' = \left(c_A^2 \left(1 - \frac{H}{h}\right)^2 - 2c_A^2 \left(1 - \frac{H}{h}\right) - gh + c_A^2\right) h', \tag{2.3.17}$$

where we simplify

$$c_A^2 \left(1 - \frac{H}{h}\right)^2 - 2c_A^2 \left(1 - \frac{H}{h}\right) + c_A^2 = c_A^2 \left(1 - \frac{H}{h} - 1\right)^2 = c_A^2 \frac{H^2}{h^2}. \tag{2.3.18}$$

Next, we integrate this term to get

$$hp + C_{hp} = -c_A^2 \frac{H^2}{h} - \frac{1}{2} g h^2. \tag{2.3.19}$$

Again, C_{hp} is a constant. By looking at the case $\xi \to \infty$, we get

$$C_{hp} = -c_A^2 H - \frac{1}{2} g H^2. \tag{2.3.20}$$

This gives us

$$hp = c_A^2 \left(H - \frac{H^2}{h}\right) + \frac{g}{2}(H^2 - h^2). \tag{2.3.21}$$

This equation conforms with [15, eq. (30)]. For the third and the fourth equation, we once again insert the first equation.

$$w(c_A - u)h' + (u - c_A)(hw)' = l\gamma p \tag{2.3.22}$$

then we use the product rule on $(hw)'$ which reduces the equation to

$$h(u - c_A)w' = l\gamma p. \tag{2.3.23}$$

Next, we have

$$h(u - c_A) = h\left(c_A - c_A \frac{H}{h} - c_A\right) = -c_A H \tag{2.3.24}$$

which in turn gives us

$$w' = \frac{-l\gamma}{c_A H} p. \tag{2.3.25}$$

If we set $\gamma = 2$, we obtain the same expression as in [15, eq. (31)]. Lastly, we consider the fourth equation. Re-writing it gives

$$-(p+c^2)uh' + (p+c^2)(hu)' + (u-c_A)(hp)' = -2c^2wl. \tag{2.3.26}$$

Once again, we insert the relation between h and hu and obtain

$$(c_A-u)(p+c^2)h' + (u-c_A)(hp)' = -2c^2wl. \tag{2.3.27}$$

Differentiating (2.3.21) hp and h gives us

$$(hp)' = \left(c_A^2\frac{H^2}{h^2} - gh\right)h'. \tag{2.3.28}$$

Inserting it into (2.3.27) yields

$$c_A\frac{H}{h}\left(c^2 + p - c_A^2\frac{H^2}{h^2} + gh\right)h' = -2c^2lw, \tag{2.3.29}$$

and thus

$$h' = \frac{-2c^2lh}{c_AH}\left(c^2 + p - c_A^2\frac{H^2}{h^2} + gh\right)^{-1}w. \tag{2.3.30}$$

As the last step, we divide (2.3.21) by h to get

$$p = c_A^2\left(\frac{H}{h} - \frac{H^2}{h^2}\right) + \frac{g}{2}\left(\frac{H^2}{h} - h\right), \tag{2.3.31}$$

and insert it into (2.3.30) to finally get

$$c^2 + p - c_A^2\frac{H^2}{h^2} + gh = c^2 + c_A^2\frac{H}{h} - 2c_A^2\frac{H^2}{h^2} + \frac{g}{2}\left(\frac{H^2}{h} + h\right). \tag{2.3.32}$$

This equation is *almost* as in [15], where it is written $2c_A$ instead of $2c_A^2$.

In summary, we get the following equation system.

$$\begin{aligned} h' &= \frac{-2c^2lh}{c_AH}\left(c^2 + c_A^2\frac{H}{h} - 2c_A^2\frac{H^2}{h^2} + \frac{g}{2}\left(\frac{H^2}{h} + h\right)\right)^{-1}w \\ w' &= \frac{-l\gamma}{c_AH}p \\ h(0) &= A + H \\ w(0) &= 0 \end{aligned} \tag{2.3.33}$$

$$\begin{aligned} p &= c_A^2\left(\frac{H}{h} - \frac{H^2}{h^2}\right) + \frac{g}{2}\left(\frac{H^2}{h} - h\right) \\ u &= c_A\frac{h-H}{h} \end{aligned}$$

Comparison to the BMSS Solitary Wave Solution

If we let $c \to \infty$, we may retrieve a simpler system. In particular, we obtain

$$\begin{aligned} h' &= \frac{-2lh}{c_A H} w \\ w' &= \frac{-l\gamma}{c_A H} p \\ h(0) &= A + H \\ w(0) &= 0 \end{aligned} \tag{2.3.34}$$

$$\begin{aligned} p &= c_A^2 \left(\frac{H}{h} - \frac{H^2}{h^2} \right) + \frac{g}{2} \left(\frac{H^2}{h} - h \right) \\ u &= c_A \frac{h - H}{h}. \end{aligned}$$

In particular, the equation concerning h' has become much simpler. Written differently, we get the second-order ODE

$$h'' = \frac{-2\gamma l^2}{c_A^2 h H^2} \left(c_A^2 \left(\frac{H}{h} - \frac{H^2}{h^2} \right) + \frac{g}{2} \left(\frac{H^2}{h} - h \right) \right), \tag{2.3.35}$$

with the initial conditions

$$\begin{aligned} h(0) &= H + A, \\ h'(0) &= 0. \end{aligned} \tag{2.3.36}$$

Choice of c_A

In [15], the parameter c_A is set to $\sqrt{g(A+H)}$. However, our calculations have shown that this choice produces incorrect results. In the following, we will heuristically explain why this is the case and derive an expression for c_A that works for finite c.

For this, we may resolve (2.3.33) even further (the paper [15] ended its derivation at (2.3.33)), by multiplying the equation for w' and the equation for h' with each other. This gives us

$$\frac{\gamma}{2hc^2} p \left(c^2 + c_A^2 \frac{H}{h} - 2c_A^2 \frac{H^2}{h^2} + \frac{1}{2} g (\frac{H^2}{h} + h) \right) h' = ww'. \tag{2.3.37}$$

The right-hand side can be integrated, since

$$\left(\frac{1}{2} w^2 \right)' = ww'. \tag{2.3.38}$$

The left-hand side can be integrated as well, since p only depends on h, and so we have an equation of the form

$$\left(\frac{1}{2} w^2 \right)' = f(h) h'. \tag{2.3.39}$$

where f can be represented as a Laurent sum.[8] This means that there are coefficients (a_n), so that

$$f(x) = \sum_{n=-N_{\min}}^{N_{\max}} a_n x^n. \tag{2.3.40}$$

for some $N_{\min}, N_{\max}$. Thus, the antiderivatives of f possess a closed-form representation. Take F to be any antiderivative of f (this ambiguity is resolved by the choice of C_{w^2} later). We get

$$\frac{1}{2}w^2 + C_{w^2} = F(h). \tag{2.3.41}$$

Here, C_{w^2} is a constant which we determine so that we obey the conditions for $\xi \to \infty$. So, we set $w = 0$ and $h = H$ to get

$$C_{w^2} = F(H). \tag{2.3.42}$$

Finally, we may choose c_A by enforcing the condition $w(0) = 0$ (and we lose all dependence on γ by this). To fulfill this (in addition to all other conditions), we have four possible choices. In particular, since we have $w(0) = 0$, we lose all dependence on γ and l. We get

$$c_{A,\pm}^2 = g(H+A) + \frac{gA(A+H)}{H}\left(\frac{1}{2} \pm \frac{1}{1+\sqrt{1+\frac{2gAH}{c^2(A+H)}}}\right). \tag{2.3.43}$$

If we require c_A to be positive, and h to decrease in the positive ξ direction (and the negative ξ therefore as well)[9], we get to the choice

$$c_A = \sqrt{c_{A,-}} = \sqrt{g(H+A) + \frac{gA(A+H)}{H}\left(\frac{1}{2} - \frac{1}{1+\sqrt{1+\frac{2gAH}{c^2(A+H)}}}\right)}. \tag{2.3.44}$$

For the limit of $c \to \infty$, we get $c_A = \sqrt{g(A+H)}$ which equals (2.3.3), and it is also the value given by [15]. However, if we were to use $\sqrt{g(A+H)}$ with our finite values of c, we would get a negative value for w^2. This would result in w having a non-zero imaginary component.

The difference between the different values for c_A is seemingly only marginal: For example, if we have $A = 0.2$, $H = 1$, and $c = 5\sqrt{gH}$, then we get $c_A \approx 3.431\mathbf{603}$. On the other hand, we have that $\sqrt{g(A+H)} \approx 3.431\mathbf{035}$. As we observe, the error is smaller than 10^{-3}. The difference becomes more noticeable for smaller c, since then (2.0.10) and (2.1.2) differ more.

[8] We will not show the full expression of f here for brevity.

[9] This was determined empirically by experiments rather than formal calculations.

2.3.3 Evaluation

Lastly, we show some plots for the solitary wave, and compare it for different values of c and c_A. We used the formulation (2.3.33) to compute a quasi-exact solution using the Python package scipy in version 1.8.0 [45], and there the `scipy.integrate.solve_ivp` method with the `DOP853` solver, though other solvers (both implicit and explicit, including the `RK45` solver) worked just as well – although here it really showed that the solitary wave heavily depends on the numerical accuracy of all variables. Thus, we set the relative numerical tolerance to 10^{-12} and the absolute numerical tolerance to 10^{-15}. With c_A chosen as in (2.3.44), we get satisfactory results up to certain values of ξ (until $h < H$, then the solution diverges). In all cases, we take $\Delta\xi = 10^{-3}$. We normalize w.r.t. l, and so we plot $l \cdot \xi = x - c_A t$ on the x-axis. Also, $\gamma = 2$ always.

In Figure 2.1, the results from [15] are reproduced. For this, we choose $H = 1$, $A = 0.2$, $c = \alpha\sqrt{gH}$ and $\alpha = 5$. For this plot, we used a resolution of $\Delta\xi = 10^{-3}$, as well as the c_A we computed in (2.3.44). As it can be seen, the plots show the same function with the same scaling as in [15, Figure 4-7]. In particular, [15, Figure 4] matches the plot over h, [15, Figure 5] matches the plot over u, [15, Figure 6] matches the plot over w, and [15, Figure 7] matches the plot over p.

Concerning the choice of c_A, we observe: choosing c_A too small lets the solution diverge (this corresponds to the problem discussed in the previous section), and it crosses $h = H$ while having $w \neq 0$. This is also what the solver produces when we choose $\sqrt{g(A+H)}$. The smaller c_A, the larger w when $h = H$. If c_A is too large, $h = H$ cannot happen. Instead, the solution oscillates between $A + H$ and some value $\tilde{H} > H$, and increasing values of c_A cause the period of the solution decreases. For example, this is the case when we take $\sqrt{c_{A,+}}$ from (2.3.43). See Figure 2.2 for a plot of different values of c_A for the same c. Whilst that, we have $H = 1$, $A = 0.2$, $c = \alpha\sqrt{gH}$ and $\alpha = 5$, and $\Delta\xi = 10^{-3}$.

Figure 2.3 shows the error for $\alpha \to \infty$ w.r.t. (2.3.4) (where $c = \alpha\sqrt{gH}$). As we can see, the solution converges for $\alpha \to \infty$ to (2.3.4). We also generated the plots for $\alpha = 1$ or smaller, but these skewed the axes too much to be shown here. The error for h can be explained by that we take $(H+A)$ as initial condition for our quasi-exact solution. Then, moving away from $\xi = 0$, we have an approximation error, and finally, the asymptotic conditions become dominant. Similar behavior is seen for w, however, here we have in addition the two extremal points of w. These do not only seem to lie at the same position for all c, but also conform exactly to the extremal points of w from (2.3.4). In total, the error is already smaller than 10^{-2} for both h and w here.

2.4 Handling Non-Conservativity

It is possible to write (2.2.1) with some functions $\tilde{\mathbf{F}} = 0$ and $\tilde{\mathbf{B}} = \mathbf{A}$ to ultimately get the same equation – though the numerical discretization may differ dependent on how $\tilde{\mathbf{F}}$ and $\tilde{\mathbf{B}}$ are used in a numerical method afterwards. But on the other hand, it is not possible to get a representation where $\tilde{\mathbf{B}} = 0$, since we have a non-conservative component.

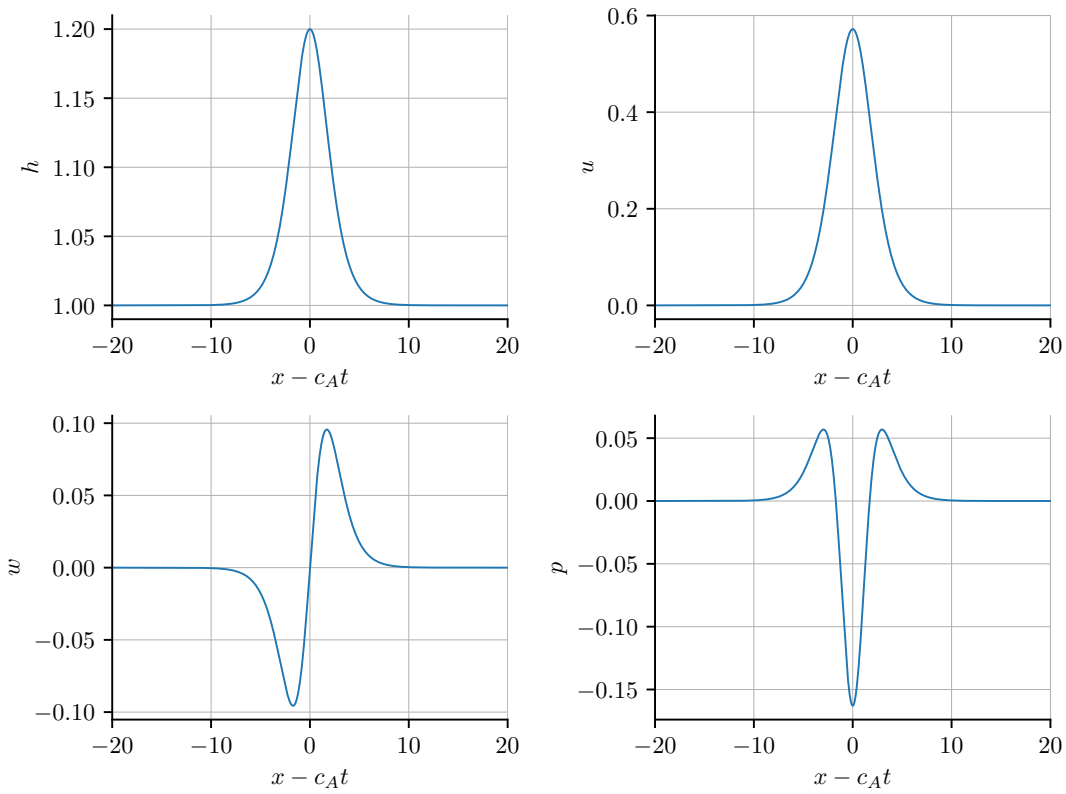


Figure 2.1: Quasi-exact eproduction of the solitary wave shown in [15, Figure 4-7] using an ODE solver. That is, we have $H = 1$, $A = 0.2$, $c = \alpha\sqrt{gH}$ and $\alpha = 5$. The quasi-exact solution was evaluated for $\Delta\xi = 0.001$ (in between, we had rtol $= 10^{-12}$ and atol $= 10^{-15}$).

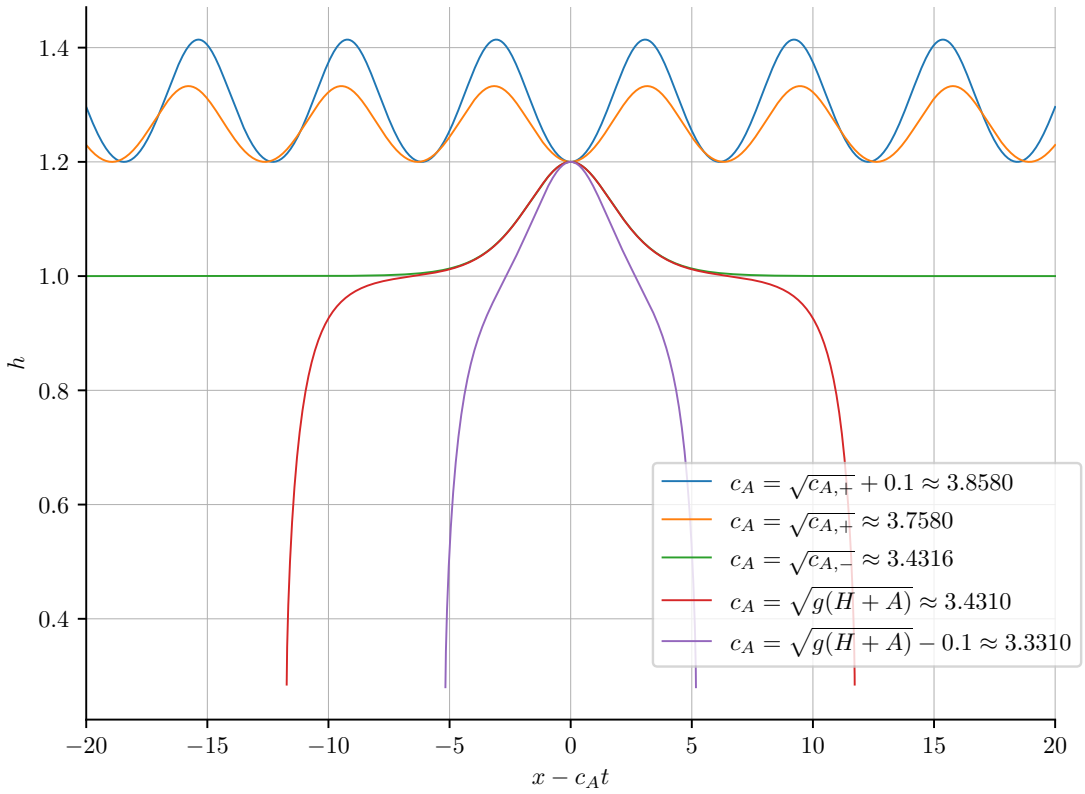


Figure 2.2: Comparison of the quasi-exact solitary wave solution (or rather, the height) for different values of c_A. Here, $c_{A,\pm}$ is defined in (2.3.43). We also have $H = 1$, $A = 0.2$, $c = \alpha\sqrt{gH}$ and $\alpha = 5$. The quasi-exact solution was evaluated for $\Delta\xi = 0.001$ (in between, we had rtol $= 10^{-12}$ and atol $= 10^{-15}$).

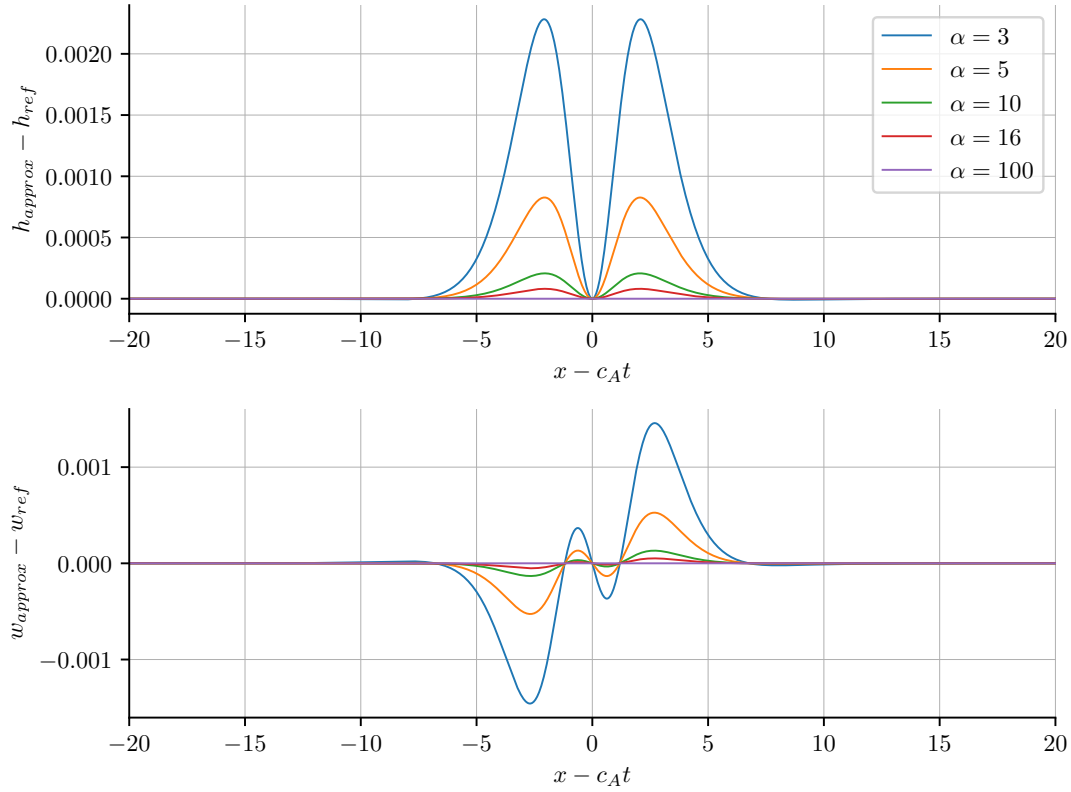


Figure 2.3: The error in h and w of the quasi-linear solution (2.3.33) w.r.t. the reference solution for $c \to \infty$. That is, (2.3.4) for different values of $c = \alpha\sqrt{gH}$. We have $H = 1$, $A = 0.2$, $c = \alpha\sqrt{gH}$. The quasi-exact solution was evaluated for $\Delta\xi = 0.001$ (in between, we had rtol $= 10^{-12}$ and atol $= 10^{-15}$).

To quantify this, a framework which builds on path integrals has been introduced. We will cite many definitions from [35] in this section for completeness, and since they are useful for solving the Riemann problem.

For the scope of this section, we have the following definitions taken from [10]. We define the L^p norms on some space with set $A \subseteq \mathbb{R}^N$ which is supposed to be measurable as

$$\|f\|_{L^p(A)} = \left(\int_A |f(x)|^p \, \mathrm{d}x \right)^{\frac{1}{p}} \tag{2.4.1}$$

for $p \in [1, \infty)$, and for $p = \infty$ as

$$\|f\|_{L^\infty(A)} = \operatorname{ess\,sup}_{x \in A}(|f(x)|). \tag{2.4.2}$$

Here $f : A \to \mathbb{R}$ is assumed to be measurable w.r.t. the respective measures. We additionally obtain the spaces

$$L^p(A) = \left\{ f : A \to \mathbb{R}^N \;\middle|\; \|f\|_{L^p(A)} < \infty \right\}. \tag{2.4.3}$$

We need one further space definition. The BV norm is given as

$$\|f\|_{BV(A)} = \sum_{k=1}^{N} \sup \left\{ \int_A f \partial_k \varphi \, \mathrm{d}x \;\middle|\; \varphi \in C_0^\infty(A), \|\varphi\|_{L^\infty(A)} \leq 1 \right\}, \tag{2.4.4}$$

and so

$$BV(A) = \left\{ f : A \to \mathbb{R} \;\middle|\; \|f\|_{BV(A)} < \infty \right\}. \tag{2.4.5}$$

Subsequently, we also can define the product norm for $\mathbf{f} \in [L^p(A)]^N$ or $\mathbf{f} \in [BV(A)]^N$ which maps from A to $\mathbb{R}^N$ as

$$\|\mathbf{f}\|_{[L^p(A)]^N} = \left(\sum_{k=1}^{N} \|\mathbf{f}_k\|_{L^p(A)}^p \right)^{\frac{1}{p}}, \tag{2.4.6}$$

for $p \in [1, \infty)$ and for $p = \infty$ as

$$\|\mathbf{f}\|_{[L^p(A)]^N} = \max_{k=1,\ldots,N} \|\mathbf{f}_k\|_{L^\infty(A)}. \tag{2.4.7}$$

Likewise, for matrices $\mathbf{F} \in [L^p(A)]^{N \times N}$, we obtain

$$\|\mathbf{F}\|_{[L^p(A)]^{N \times N}} = \left(\sum_{i,j=1}^{N} \|\mathbf{F}_{ij}\|_{L^p(A)}^p \right)^{\frac{1}{p}}, \tag{2.4.8}$$

for $p \in [1, \infty)$ and for $p = \infty$ as

$$\|\mathbf{F}\|_{[L^p(A)]^{N \times N}} = \max_{i,j=1,\ldots,N} \|\mathbf{F}_{ij}\|_{L^\infty(A)}. \tag{2.4.9}$$

2.4.1 DLM Path Theory

For the scope of this section, we will write

$$\|\mathbf{f}\|_{L^1} := \|\mathbf{f}\|_{[L^1([0,1])]^N}\,, \tag{2.4.10}$$

$$\|\mathbf{f}\|_{L^\infty} := \|\mathbf{f}\|_{[L^\infty([0,1])]^N}\,. \tag{2.4.11}$$

In particular, we obtain that for $\mathbf{f} : [0,1] \to \mathbb{R}^N$ we have

$$\|\mathbf{f}\|_{L^1} = \sum_{k=1}^{N} \|\mathbf{f}_k\|_{L^1([0,1])} \leq \sum_{k=1}^{N} \|\mathbf{f}_k\|_{L^\infty([0,1])} \leq \|\mathbf{f}_{L^\infty}\|\,. \tag{2.4.12}$$

For $\mathbf{G} : [0,1] \to \mathbb{R}^{N\times N}$, we get by Hölder's inequality that

$$\begin{aligned}\int_0^1 \mathbf{G}(s)\mathbf{f}(s)\,\mathrm{d}s &\leq \sum_{j=1}^{N} \|\mathbf{G}_j\mathbf{f}_j\|_{L^1} \\ &= \sum_{i=1,j=1}^{N} \|\mathbf{G}_{ij}\mathbf{f}_j\|_{L^1([0,1])} \\ &\leq \sum_{i=1,j=1}^{N} \|\mathbf{G}_{ij}\|_{L^\infty([0,1])}\,\|\mathbf{f}_j\|_{L^1([0,1])} \\ &\leq N\,\|\mathbf{G}\|_{[L^\infty([0,1])]^{N\times N}}\,\|\mathbf{f}\|_{L^1}\,.\end{aligned} \tag{2.4.13}$$

Here $\mathbf{G}_j$ denoted the j-th column of $\mathbf{G}$.

We now begin with the main part. For that, we cite a small helper theorem.

Theorem 2.6 (Rademacher)**.** *Let $\Omega \subseteq \mathbb{R}^N$ be open, and let $f : \Omega \to \mathbb{R}^M$ be Lipschitz continuous. Then f is differentiable almost everywhere in Ω.*

Proof. See [23, Theorem 3.1]. □

Next, we introduce the main definition (as found e.g. in [35, Definition 2.1]). DLM stands for "DelMaso-LeFloch-Murat" which introduced this concept (cf. [18]).

Definition 2.7 (Admissible Paths / DLM Paths)**.** *Let $\Omega \subseteq \mathbb{R}^N$ be open. A map $\Psi[\mathbf{U}_L, \mathbf{U}_R]$*

$$\Psi : \Omega \times \Omega \to [C([0,1])]^N \tag{2.4.14}$$

is called admissible, if it fulfills:

1. *$\Psi[\mathbf{U}_L, \mathbf{U}_R](0) = \mathbf{U}_L$ and $\Psi[\mathbf{U}_L, \mathbf{U}_R](1) = \mathbf{U}_R$.*
2. *For all $[\mathbf{U}_L, \mathbf{U}_R] \in \Omega \times \Omega$, we have that $\Psi[\mathbf{U}_L, \mathbf{U}_R]$ is (globally) Lipschitz continuous.*

3. *For all bounded $P \subseteq \Omega$, there is a constant $C_P > 0$, such that for $\mathbf{U}_L, \mathbf{U}_R \in P$, we have*
$$\left\|\Psi'[\mathbf{U}_L, \mathbf{U}_R]\right\|_{L^\infty} \leq C_P \left\|\mathbf{U}_L - \mathbf{U}_R\right\| \tag{2.4.15}$$

4. *For all bounded $P \subseteq \Omega$, there is a constant $C'_P > 0$, such that for $\mathbf{U}_L, \mathbf{U}_R \in P$ and $\mathbf{V}_L, \mathbf{V}_R \in P$, we have*
$$\left\|\Psi'[\mathbf{U}_L, \mathbf{U}_R] - \Psi'[\mathbf{V}_L, \mathbf{V}_R]\right\|_{L^\infty} \leq C'_P \left(\left\|\mathbf{U}_L - \mathbf{V}_L\right\| + \left\|\mathbf{U}_R - \mathbf{V}_R\right\|\right) \tag{2.4.16}$$

In the definition, we get the existence of $\Psi'[\mathbf{U}_L, \mathbf{U}_R]$ almost everywhere due to Theorem 2.6. The exact choice of norm in the definition in properties 3 and 4 can be chosen freely, since $\mathbf{U}_L, \mathbf{U}_R$ are in a finite-dimensional space, and thus, we have norm equivalence (and also, the exact values of C_P and C'_P are usually not important). For the scope of this work, it suffices to look at piece-wise continuously differentiable paths.

Assumption 2.8. *Let $\Omega \subseteq \mathbb{R}^N$ be open. For a family of paths $\Psi[\mathbf{U}_L, \mathbf{U}_R]$ on Ω, we assume in addition that $\Psi[\mathbf{U}_L, \mathbf{U}_R]$ is piecewise C^1 and continuous on $[0, 1]$ for all $\mathbf{U}_L, \mathbf{U}_R \in \Omega$. This means that there are at most finitely many points, where $\Psi[\mathbf{U}_L, \mathbf{U}_R]$ is not C^1.*

Most likely, all the following results can be easily generalized to not needing Assumption 2.8, using e.g. results from [23, Section 3]. However, we will not do that here. In essence, the property that we require is that the Fundamental Theorem of Calculus can be applied.

To begin with, we show why the definition is useful. We get for Ψ fulfilling Definition 2.7 and Assumption 2.8 by the chain rule that for the conservative part $\mathbf{F}$ of (2.2.1) we have
$$D\left[\mathbf{F}(\Psi[\mathbf{U}_L, \mathbf{U}_R](t))\right] = D\mathbf{F}(\Psi[\mathbf{U}_L, \mathbf{U}_R](t))\Psi'[\mathbf{U}_L, \mathbf{U}_R](t), \tag{2.4.17}$$
and thus, by the Fundamental Theorem of Calculus, we obtain
$$\begin{aligned} &\int_0^1 D\mathbf{F}(\Psi[\mathbf{U}_L, \mathbf{U}_R](t))\Psi'[\mathbf{U}_L, \mathbf{U}_R](t)\,\mathrm{d}t \\ &= \mathbf{F}(\Psi[\mathbf{U}_L, \mathbf{U}_R](1)) - \mathbf{F}(\Psi[\mathbf{U}_L, \mathbf{U}_R](0)) \\ &= \mathbf{F}(\mathbf{U}_R) - \mathbf{F}(\mathbf{U}_L). \end{aligned} \tag{2.4.18}$$
and so for $\mathbf{A}$:
$$\begin{aligned} &\int_{\Psi[\mathbf{U}_L, \mathbf{U}_R]} \mathbf{A}(\mathbf{U})\,\mathrm{d}\mathbf{U} \\ &= \int_0^1 \left(D\mathbf{F}(\Psi[\mathbf{U}_L, \mathbf{U}_R](t)) + \mathbf{B}(\Psi[\mathbf{U}_L, \mathbf{U}_R](t))\right)\Psi'[\mathbf{U}_L, \mathbf{U}_R](t)\,\mathrm{d}t \\ &= \mathbf{F}(\mathbf{U}_R) - \mathbf{F}(\mathbf{U}_L) + \int_0^1 \mathbf{B}(\Psi[\mathbf{U}_L, \mathbf{U}_R](t))\Psi'[\mathbf{U}_L, \mathbf{U}_R](t)\,\mathrm{d}t. \end{aligned} \tag{2.4.19}$$

In the following, we are going to prove some simple results for convenience.[10] Definition 2.7 and Assumption 2.8 implies that we have that $\Psi[\mathbf{U}, \mathbf{U}]$ (for $\mathbf{U} \in \Omega$) is a constant curve.

Lemma 2.9. *Let Ψ be an admissible family of paths according to Definition 2.7 and Assumption 2.8. Then it holds for all $\mathbf{U} \in \Omega$ that for all $t \in [0, 1]$*

$$\Psi[\mathbf{U}, \mathbf{U}](t) = \mathbf{U} \tag{2.4.20}$$

In particular, we get for all integrable $\mathbf{F} : \Omega \to \mathbb{R}^{N \times N}$ that

$$\int_{\Psi[\mathbf{U},\mathbf{U}]} f(\mathbf{U}) \, d\mathbf{U} = 0 \tag{2.4.21}$$

Proof. Let $\mathbf{U} \in \Omega$. By property 3 from Definition 2.7, we get if we set $P = \{\mathbf{U}\}$, there exists a C_K, such that

$$\left\|\Psi'[\mathbf{U}, \mathbf{U}]\right\|_{L^\infty} \leq C_K \left\|\mathbf{U} - \mathbf{U}\right\| = 0. \tag{2.4.22}$$

From that, we may infer directly for all integrable $\mathbf{F} : \Omega \to \mathbb{R}^{N \times N}$ that

$$\int_{\Psi[\mathbf{U},\mathbf{U}]} \mathbf{F}(\mathbf{U}) \, \mathrm{d}\mathbf{U} = \int_0^1 \mathbf{F}(\Psi[\mathbf{U}, \mathbf{U}](t)) \Psi'[\mathbf{U}, \mathbf{U}](t) \, \mathrm{d}t = 0. \tag{2.4.23}$$

Thus, $\Psi[\mathbf{U}, \mathbf{U}]$ is constant almost everywhere. To get that it is constant *everywhere*, we use that it holds for all t due to Assumption 2.8 that

$$\Psi[\mathbf{U}, \mathbf{U}](t) - \Psi[\mathbf{U}, \mathbf{U}](0) = \int_0^t \Psi'[\mathbf{U}, \mathbf{U}](s) \, \mathrm{d}s = 0. \tag{2.4.24}$$

Thus, for all $t > 0$, we get

$$\Psi[\mathbf{U}, \mathbf{U}](t) = \Psi[\mathbf{U}, \mathbf{U}](0) = \mathbf{U}. \tag{2.4.25}$$

□

In addition, we obtain an easier-to-check condition, equivalent for Definition 2.7 and Assumption 2.8.

Lemma 2.10. *In Definition 2.7 and given Assumption 2.8, property 3 can be exchanged for: for all $\mathbf{U} \in \Omega$ we have*

$$\Psi[\mathbf{U}, \mathbf{U}] \equiv \mathbf{U} \tag{2.4.26}$$

Proof. To see this, take a bounded set P, take $\mathbf{U}_L, \mathbf{U}_R \in P$, and apply property 4 onto

$$\begin{aligned} &\left\|\Psi'[\mathbf{U}_L, \mathbf{U}_R]\right\|_{L^\infty} \\ &= \left\|\Psi'[\mathbf{U}_L, \mathbf{U}_R] - \Psi'[\mathbf{U}_L, \mathbf{U}_L]\right\|_{L^\infty} \\ &\leq C'_P \left\|\mathbf{U}_R - \mathbf{U}_L\right\|. \end{aligned} \tag{2.4.27}$$

Here $C'_P > 0$ was the constant from property 4. This is exactly property 3. The other direction was proven with Lemma 2.9. □

[10] We did not find these results in the literature we looked at.

For smooth paths, we obtain even easier conditions to check, similarly to when checking the conditions for e.g. Picard-Lindelöf.

Corollary 2.11. *Let Ω be open, and $\Phi[\mathbf{U}_L, \mathbf{U}_R]$ fulfills for all $\mathbf{U}_L, \mathbf{U}_R \in \Omega$ that:*

1. *$\Phi[\mathbf{U}_L, \mathbf{U}_R](0) = \mathbf{U}_L$ and $\Phi[\mathbf{U}_L, \mathbf{U}_R](1) = \mathbf{U}_R$.*
2. *$\Phi[\mathbf{U}_L, \mathbf{U}_R](t)$ is continuously differentiable in $t \in [0, 1]$.*
3. *For all $\mathbf{U} \in \Omega$, we have $\Phi[\mathbf{U}, \mathbf{U}] \equiv \mathbf{U}$.*
4. *$(t, \mathbf{U}_L, \mathbf{U}_R) \mapsto \Phi'[\mathbf{U}_L, \mathbf{U}_R]$ is continuously differentiable w.r.t. $\mathbf{U}_L$ and $\mathbf{U}_R$.*

Then $\Phi[\mathbf{U}_L, \mathbf{U}_R]$ fulfills Definition 2.7, and Assumption 2.8.

Proof. (partially only sketched) By condition 2, Φ fulfills Assumption 2.8. Conditions 1 and 3 equal properties 1 and 3 from Definition 2.7 with Lemma 2.10. Condition 2 yields that $\Phi[\mathbf{U}_L, \mathbf{U}_R](t)$ is globally Lipschitz continuous, since its derivative w.r.t. t is continuous, and thus bounded on $[0, 1]$. We only sketch how property 4 can be shown, since it is technical and tedious, and therefore a detailed proof will be omitted here. For $P \in \Omega$ bounded, its closure $\mathrm{cl}(P)$ which is compact. Thus, $(t, \mathbf{U}_L, \mathbf{U}_R) \mapsto \Phi'[\mathbf{U}_L, \mathbf{U}_R]$ is bounded in all three variables on $[0, 1] \times P \times P$. Then, we treat both the $\mathbf{U}_L$ and $\mathbf{U}_R$ components separately, and look w.l.o.g. for $\mathbf{U}_R$ at the function

$$T(s) = \Phi'[\mathbf{W}, \mathbf{U}(1 - s) + \mathbf{V}s](t). \tag{2.4.28}$$

We apply the Mean Value Theorem onto $v^T T(s)$ for all $v \in \mathbb{R}^N$ to obtain that there is for each v a $\xi_v \in (0, 1)$ for which

$$v^T (T(1) - T(0)) = v^T T'(\xi_v). \tag{2.4.29}$$

If we take $v = T(1) - T(0)$, we obtain

$$\|T(1) - T(0)\|^2 \leq \|T(1) - T(0)\| \left\|T'\right\|_\infty. \tag{2.4.30}$$

Then, we can bound for some $C > 0$ which only depends on P that

$$\left\|T'\right\|_\infty \leq C \|\mathbf{U} - \mathbf{V}\|. \tag{2.4.31}$$

Doing this for the $\mathbf{U}_L$ component as well gives us property 4. □

Next, we note that we also get an estimate for $\|\Phi[\mathbf{U}_L, \mathbf{U}_R] - \mathbf{U}_L\|$ from Definition 2.7 and Assumption 2.8.

Lemma 2.12. *Let Ω be open, and $\Phi[\mathbf{U}_L, \mathbf{U}_R]$ fulfills Definition 2.7 and Assumption 2.8. Then, for $P \subseteq \Omega$ bounded, there is a constant $C_P > 0$, such that for $\mathbf{U}_L, \mathbf{U}_R \in P$, we have for $t \in [0, 1]$ the estimate*

$$\|\Phi[\mathbf{U}_L, \mathbf{U}_R](t) - \mathbf{U}_L\| \leq C_P \|\mathbf{U}_R - \mathbf{U}_L\|. \tag{2.4.32}$$

Proof. Let P be such a set, and $\mathbf{U}_L, \mathbf{U}_R \in P$. In addition, we get a C_P by property 3 from Definition 2.7.

We have

$$\mathbf{U}_L = \Phi[\mathbf{U}_L, \mathbf{U}_R](0). \tag{2.4.33}$$

We obtain by the fundamental theorem of calculus, that we have

$$\begin{aligned} &\|\Phi[\mathbf{U}_L, \mathbf{U}_R](t) - \mathbf{U}_L\| \\ &= \left\| \int_0^t \Phi'[\mathbf{U}_L, \mathbf{U}_R](s) \, \mathrm{d}s \right\| \\ &\leq \int_0^1 \left\| \Phi'[\mathbf{U}_L, \mathbf{U}_R](s) \right\| \mathrm{d}s \\ &\leq \left\| \Phi'[\mathbf{U}_L, \mathbf{U}_R] \right\|_{L^\infty} \\ &\leq C_P \left\| \mathbf{U}_L - \mathbf{U}_R \right\|, \end{aligned} \tag{2.4.34}$$

where in the penultimate step, we used that the $L^1([0,1])$ norm is bounded above by the $L^\infty([0,1])$ norm, and in the last step, we used property 3. □

Next, we look at the smoothness of the integrals with paths, with respect to their endpoints.

Lemma 2.13. *Suppose Φ fulfills Definition 2.7 and Assumption 2.8. Suppose we have given $f \in C(\Omega, \mathbb{R}^{N\times N})$ is locally Lipschitz continuous, let $\mathbf{U}_L, \mathbf{U}_R \in \Omega$. Define*

$$I_{\mathbf{U}_L, f}(\mathbf{U}_R) = \int_{\Phi[\mathbf{U}_L, \mathbf{U}_R]} f(\mathbf{U}) \, d\mathbf{U}. \tag{2.4.35}$$

Then $I_{\mathbf{U}_L, f}$ is differentiable at $\mathbf{U}_R = \mathbf{U}_L$, and we obtain

$$DI_{\mathbf{U}_L, f}(\mathbf{U}_L) = f(\mathbf{U}_L). \tag{2.4.36}$$

Similarly, we obtain that

$$J_{\mathbf{U}_R, f}(\mathbf{U}_L) = \int_{\Phi[\mathbf{U}_L, \mathbf{U}_R]} f(\mathbf{U}) \, d\mathbf{U} \tag{2.4.37}$$

is differentiable for $\mathbf{U}_R = \mathbf{U}_L$ and

$$DJ_{\mathbf{U}_R, f}(\mathbf{U}_L) = -f(\mathbf{U}_L). \tag{2.4.38}$$

Proof. Let $\mathbf{U} \in \Omega$. Then we have $P := B(\mathbf{U}, \epsilon)$ for $\epsilon > 0$, so that f has the Lipschitz constant L on P, and there is C_P, so that property 3 from Definition 2.7 is fulfilled. Let $\mathbf{V} \in P$ by arbitrary. To begin with, we note that $I_f(\mathbf{U}, \mathbf{U}) = 0$ by Lemma 2.9. Subsequently, we may write

$$\begin{aligned} &I_{\mathbf{U}, f}(\mathbf{V}) - I_{\mathbf{U}, f}(\mathbf{U}) - f(\mathbf{U})(\mathbf{V} - \mathbf{U}) \\ &= I_{\mathbf{U}, (f - f(\mathbf{U}))}(\mathbf{V}). \end{aligned} \tag{2.4.39}$$

Next, we use Lemma 2.12 with P, to get a constant $C'_P > 0$ and

$$\|\Phi[\mathbf{U},\mathbf{V}](\,\cdot\,) - \Phi[\mathbf{U},\mathbf{V}](0)\|_{L^\infty} \leq C'_P \|\mathbf{U}-\mathbf{V}\|. \tag{2.4.40}$$

Applying Hölder's inequality, and using the Lipschitz continuity, property 3, and using that the norm of L^1 is bounded from above by the L^∞ norm, we obtain

$$\begin{aligned}
&\left\|I_{\mathbf{U},(f-f(\mathbf{U}))}(\mathbf{V})\right\| \\
&\leq N \|f(\Phi[\mathbf{U},\mathbf{V}](\,\cdot\,)) - f(\mathbf{U})\|_{L^\infty} \left\|\Phi'[\mathbf{U},\mathbf{V}]\right\|_{L^1} \\
&\leq LN \|\Phi[\mathbf{U},\mathbf{V}](\,\cdot\,) - \Phi[\mathbf{U},\mathbf{V}](0)\|_{L^\infty} \left\|\Phi'[\mathbf{U},\mathbf{V}]\right\|_{L^\infty} \\
&\leq (NLC_PC'_P) \|\mathbf{U}-\mathbf{V}\|^2.
\end{aligned} \tag{2.4.41}$$

Thus, we get

$$\frac{\|I_{\mathbf{U},f}(\mathbf{V}) - I_{\mathbf{U},f}(\mathbf{U}) - f(\mathbf{U})(\mathbf{V}-\mathbf{U})\|}{\|\mathbf{V}-\mathbf{U}\|} \leq (NLC_PC'_P)\|\mathbf{U}-\mathbf{V}\|. \tag{2.4.42}$$

Sending $\|\mathbf{U}-\mathbf{V}\| \to 0$ gives us the result. The result for $J_{\mathbf{U}_R,f}(\mathbf{U}_L)$ can be obtained the same way, by swapping $\mathbf{V}$ and $\mathbf{U}$, and subsequently reversing the sign. □

Lastly, we are going to look at two examples for Definition 2.7. To begin with, we define

$$P[\mathbf{U}_L,\mathbf{U}_R](t) = (1-t)\mathbf{U}_L + t\mathbf{U}_R. \tag{2.4.43}$$

Firstly, we then have straight lines in conservative variables which is written

$$\Psi_C[\mathbf{U}_L,\mathbf{U}_R] = P[\mathbf{U}_L,\mathbf{U}_R]. \tag{2.4.44}$$

Next, we have straight lines in primitive variables which read

$$\Psi_P[\mathbf{U}_L,\mathbf{U}_R] = \begin{pmatrix} P[h_L,h_R] \\ P[h_L,h_R]P[u_L,u_R] \\ P[h_L,h_R]P[v_L,v_R] \\ P[h_L,h_R]P[w_L,w_R] \\ P[h_L,h_R]P[p_L,p_R] \\ P[b_L,b_R] \end{pmatrix}. \tag{2.4.45}$$

The latter is of special interest, since then $\mathbf{B}$ is linear in primitive variables. However, we need to restrict Ω to $h > 0$ for Ψ_P.

Corollary 2.14. *Ψ_C fulfills Definition 2.7 and Assumption 2.8 on $\Omega = \mathbb{R}^6$, and Ψ_P fulfills Definition 2.7 and Assumption 2.8 on $\Omega = \mathbb{R}^+ \times \mathbb{R}^5$.*

Proof. Apply Corollary 2.11, since both families of paths are just polynomials in $\mathbf{U}_L, \mathbf{U}_R$, and t (therefore conditions 2 and 4 are fulfilled). Also, they are built out of linear paths, so conditions 1 and 3 are fulfilled. □

In the following sections, families of paths Φ (or Ψ) will appear, always for the six variables h, hu, hv, hw, hp, b. Thus, we will from now on denote the first component of Φ by Φ_h, the second component by Φ_{hu}, the third component by Φ_{hv}, and so on. We also extend this to the primitive formulation, so Φ_u stands for

$$\Phi_u = \frac{\Phi_{hu}}{\Phi_h}. \tag{2.4.46}$$

The process is similar for the other primitive variables v, w, and p.

We will have to deal with integrals which are written in the form

$$\int_{\Phi[\mathbf{U}_L,\mathbf{U}_R]} f_h(h, hu, hv, hw, hp, b)\ \mathrm{d}h, \tag{2.4.47}$$

or for all other variables. This is defined as

$$\begin{aligned}
&\int_{\Phi[\mathbf{U}_L,\mathbf{U}_R]} f_h(h, hu, hv, hw, hp, b)\ \mathrm{d}h \\
&= \int_{\Phi[\mathbf{U}_L,\mathbf{U}_R]} \begin{pmatrix} f_h(h, hu, hv, hw, hp, b) & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \mathrm{d}\mathbf{U} \\
&= \int_0^1 f_h\left(\Phi_h(t), \Phi_{hu}(t), \Phi_{hv}(t), \Phi_{hw}(t), \Phi_{hp}(t), \Phi_b(t)\right) \Phi_h'(t)\ \mathrm{d}t,
\end{aligned} \tag{2.4.48}$$

where we wrote for brevity $\Phi = \Phi[\mathbf{U}_L, \mathbf{U}_R]$. In particular, we also obtain by Lemma 2.13 that

$$\partial_R \int_{\Phi[\mathbf{U}_L,\mathbf{U}_L]} f_h(h, hu, hv, hw, hp, b)\ \mathrm{d}h = f_h(h_L, (hu)_L, (hv)_L, (hw)_L, (hp)_L, b_L). \tag{2.4.49}$$

Here ∂_R denoted the derivative by the second argument in Φ.

2.4.2 The Riemann Problem

Next, we want to consider the Riemann problem, and solve it for (2.0.10). Before we do that, we recapitulate some general facts about the Riemann problem and its structure.

In general, the Riemann problem for some smooth function $\hat{\mathbf{A}} : \Omega \subseteq \mathbb{R}^N \to \mathbb{R}^{N\times N}$ is written as

$$\begin{aligned}
&\partial_t \mathbf{U} + \hat{\mathbf{A}}(\mathbf{U})\partial_x \mathbf{U} = 0 \\
&\mathbf{U}(0,x) = \begin{cases} \mathbf{U}_L & x < 0 \\ \mathbf{U}_R & x > 0, \end{cases}
\end{aligned} \tag{2.4.50}$$

so that the system is strictly hyperbolic. This means $\hat{\mathbf{A}}$ has the eigenvalues $\hat{\lambda}_1 < ... < \hat{\lambda}_N$, and eigenvectors $\hat{r}_1, ..., \hat{r}_N$. In particular, the Riemann Problem is a one-dimensional problem[11], hence we are able to talk about left-hand side and right-hand side states for

[11] There are more-dimensional extensions to the Riemann Problem, see e.g. [46].

which we write $\mathbf{U}_L$ and $\mathbf{U}_R$. The Riemann Problem has been extensively studied, and there is a vast literature to it. For example, one may look at [27] for an overview of the conservative case (including proofs), or [24] on how to compute it. For reference, and since we will look at the Riemann problem of (2.0.10), we will cite some results for the non-conservative case, with a focus on computing the Riemann problem. As a general guideline, everywhere in the formulas, where $\hat{\mathbf{F}}(\mathbf{U})$ appears in the conservative case, we need to handle the non-conservativity once it appears. Everywhere, where only $\hat{\mathbf{A}}$ appears for the conservative case, we may copy the argumentation. In essence, the problem and the structure stay the same as in the conservative case.

The main change is the definition of the weak solution. As in the conservative case [27], we cannot expect the existence of classical solutions for all time anymore, but the regular concept of weak solutions fail in the non-conservative case as well [18]. Instead, we will use the adjusted concept of weak solutions which also works for non-conservative systems (see e.g. [18]). However, these introduce more ambiguity in the form of that we have to choose a family of paths which fulfills Definition 2.7. Assumption 2.8 is not required in [18]. In total, we will speak here of Φ-weak solutions, where Φ is an admissible family of paths in the sense of Definition 2.7. The following definition taken from [34, eq. (2.2)] and [18, Definition 3.1, eq. (3.3)].

Definition 2.15 (Weak Solutions)**.** *A function* $\mathbf{U} \in [L^\infty(\mathbb{R}^+ \times \mathbb{R}) \cap BV(\mathbb{R}^+ \times \mathbb{R})]^N$ *is a* Φ*-weak solution, if we have for all* $\varphi : \mathbb{R}^+ \times \mathbb{R} \to \mathbb{R}$ *which are continuously differentiable with compact support*[12] *that*

$$\begin{aligned} 0 = &\int_{\mathbb{R}_+} \int_{\mathbb{R}} \partial_t \varphi(t,x) \mathbf{U}(t,x) \; dx \; dt \\ &+ \int_{\mathbb{R}_+} \int_{\mathbb{R}} \varphi(t,x) \hat{\mathbf{A}}(\mathbf{U}(t,x)) \partial_x \mathbf{U}(t,x) \; dx \; dt \\ &+ \sum_k \varphi(t, x_k(t)) \int_{\Phi[\mathbf{U}_k^-, \mathbf{U}_k^+]} \hat{\mathbf{A}}(\mathbf{V}) \; d\mathbf{V}. \end{aligned} \tag{2.4.51}$$

Here, $x_k(t)$ *is the* k*-th discontinuity at time* $t > 0$*, and* $\mathbf{U}_k^+$ *and* $\mathbf{U}_k^-$ *denote the limits in positive and negative direction towards* $x_k(t)$ *at time* t*, respectively. There are at most countably many discontinuities at all times* $t > 0$*.*

Only the third term is new in comparison to the conservative case, and it is the only term which depends on the family of paths Φ. With Definition 2.15, we obtain a generalized notion of the Rankine-Hugoniot conditions.

Lemma 2.16 (Generalized Rankine-Hugoniot Conditions)**.** *Suppose that* $\mathbf{U}^*$ *is a* Φ*-weak solution to* (2.4.50)*. If we have a discontinuity at a point* ξ *with values* $\mathbf{U}_L$ *and* $\mathbf{U}_R$ *on the left and right-hand side respectively, the generalized Rankine-Hugoniot conditions*

[12] It is be very well possible to let φ map to $\mathbb{R}^N$ instead, but this would give us the same definition just with more vector notation.

must be satisfied. That is, we require for $s \in \mathbb{R}$ that

$$\int_{\Phi[\mathbf{U}_L,\mathbf{U}_R]} sI - \hat{\mathbf{A}}(\mathbf{U})\ d\mathbf{U} = 0. \tag{2.4.52}$$

Proof. See the comment near [18], or [27] for a proof in the conservative case and a single conservation law. □

As a remark which is also mentioned in [18], we may write (2.4.52) as

$$\int_{\Phi[\mathbf{U}_L,\mathbf{U}_R]} \hat{\mathbf{A}}(\mathbf{U})\ \mathrm{d}\mathbf{U} = s(\mathbf{U}_R - \mathbf{U}_L). \tag{2.4.53}$$

For a conservative equation, i.e. $\mathbf{A} = D\mathbf{F}$, we get the classical Rankine-Hugoniot conditions. Then, we have

$$\hat{\mathbf{F}}(\mathbf{U}_R) - \hat{\mathbf{F}}(\mathbf{U}_L) = \int_{\Phi[\mathbf{U}_L,\mathbf{U}_R]} \hat{\mathbf{A}}(\mathbf{U})\ \mathrm{d}\mathbf{U} = s(\mathbf{U}_R - \mathbf{U}_L). \tag{2.4.54}$$

While we now have a generalized concept for weak solutions, we have the problem that the weak solutions are not unique, since they were even not so in the conservative case. An approach that is usually chosen to select a useful weak solution is the artificial viscosity approach which we will also follow here. That is, we consider for $\epsilon > 0$ the system (see also [18])

$$\begin{aligned} &\partial_t \mathbf{U} + \hat{\mathbf{A}}(\mathbf{U})\partial_x \mathbf{U} = \epsilon \partial_x (\mathbf{E}(\mathbf{U})\partial_x \mathbf{U}) \\ &\mathbf{U}(0,x) = \begin{cases} \mathbf{U}_L & x < 0 \\ \mathbf{U}_R & x > 0, \end{cases} \end{aligned} \tag{2.4.55}$$

where $\mathbf{E}$ is smooth and matrix-valued (see e.g. [18]), and $\mathbf{E}(\mathbf{U}) \geq 0$ for all $\mathbf{U}$ where it is defined. This problem is then solved for $\epsilon > 0$, yielding a $\mathbf{U}_\epsilon$ as solution. Then, the limit of the $\mathbf{U}_\epsilon$ is taken for $\epsilon \to 0$ (for the conservative case, see [27], for the non-conservative case [29]), and this is then declared to be our desired solution. We obtain, in the conservative case, a certain solution structure which is handled in [27]. In the non-conservative case, the dependency on paths unfortunately cannot be removed this way, although some heuristic techniques exist [18]. But at least, we obtain a solution structure for a given family of paths Ψ.

So now, we will discuss said solution structure. It is, up to the Generalized Rankine-Hugoniot conditions, exactly the same as in the conservative case. For completeness, we provide here a small overview. We begin with the well-known classification of characteristic fields, as it can be seen in e.g. [27, eq. (5.3)].

Definition 2.17. *The characteristic field associated to $\hat{r}_k$ is called linearly degenerate, if it holds everywhere on Ω that*

$$\nabla \hat{\lambda}_k \cdot \hat{r}_k = 0. \tag{2.4.56}$$

It is called genuinely nonlinear, if we have on Ω that

$$\nabla \hat{\lambda}_k \cdot \hat{r}_k \neq 0. \tag{2.4.57}$$

In addition, there is the notion of Riemann invariants (see e.g. [26, Definition 8.1]).

Definition 2.18. *A function i is called a Riemann invariant to the field associated with $\hat{r}_k$ (or a k-Riemann invariant), if*

$$\nabla i \cdot \hat{r}_k = 0. \tag{2.4.58}$$

In particular, $\hat{\lambda}_k$ is a Riemann invariant, if the field of $\hat{r}_k$ is linearly degenerate. We can say even more about Riemann invariants, namely that there are at most $N-1$ of them (for $\mathbb{R}^N$ being the space dimension) whose gradients are linearly independent [26, Theorem 8.1]. In addition, we also use the Lax entropy condition just as in the conservative case (see e.g. [35, Definition 2.2] or [27, eq. (5.5)], or [21]) in order to achieve an empirically "good" solution which is related to the artificial viscosity approach. In addition to that, it is rather simple to check.

Definition 2.19 (Lax Entropy Condition)**.** *Suppose* $\mathbf{A}$ *has eigenvalues* $\hat{\lambda}_1 < ... < \hat{\lambda}_N$*. For this definition, we also set* $\hat{\lambda}_0 = -\infty$ *and* $\hat{\lambda}_{N+1} = \infty$*. A* Φ*-weak solution* $\mathbf{U}$ *is called entropic, if at each discontinuity with left and right states* $\mathbf{U}_L$ *and* $\mathbf{U}_R$*, and Rankine-Hugoniot shock speed* s*, there is a* $k = 1, ..., N$*, such that we have:*

- *If* r_k *is genuinely nonlinear, then we have*

$$\begin{aligned} \hat{\lambda}_{k-1}(\mathbf{U}_L) <& s < \hat{\lambda}_k(\mathbf{U}_L) \\ \hat{\lambda}_k(\mathbf{U}_R) <& s < \hat{\lambda}_{k+1}(\mathbf{U}_R). \end{aligned} \tag{2.4.59}$$

- *If* r_k *is linearly degenerate, then we have*

$$\hat{\lambda}_k(\mathbf{U}_L) = s = \hat{\lambda}_k(\mathbf{U}_R). \tag{2.4.60}$$

Given all these definitions, we are able to consider three types of simple waves (cf. [27]) from which we are able to build a solution. All of these can be written in the similarity variable $\xi = x/t$.

- Centered rarefaction. Continuous variation though a single, no discontinuity. Thus, we need to examine the characteristic field of $\hat{r}$. As described in [27], the idea is to use the variable transformation to $\xi = x/t$. Inserting this into (2.4.50), we obtain

$$-\frac{1}{t}\xi\partial_\xi \mathbf{U} + \frac{1}{t}\hat{\mathbf{A}}(\mathbf{U})\partial_\xi \mathbf{U} = 0 \tag{2.4.61}$$

  which is solved for $t > 0$, iff $\partial_\xi \mathbf{U}$ is an eigenvector of $\hat{\mathbf{A}}(\mathbf{U})$. In particular, for $\hat{\lambda}(\xi)$ being an eigenvalue, and $\hat{r}(\xi)$ being an eigenvector of $\hat{\mathbf{A}}(\mathbf{U})$, we obtain the relation (see e.g. [27])

$$\xi = \hat{\lambda}(\mathbf{U}(\xi)), \tag{2.4.62}$$

  and also the relation

$$\partial_\xi \mathbf{U} = a(\xi)\hat{r}(\xi), \tag{2.4.63}$$

where a is a normalization factor. Suppose we are able to solve (2.4.63) for $\mathbf{U}$, yielding us a solution $\mathbf{U}^*$. Then, we obtain for some ξ_L, ξ_R the weak solution

$$w_r(\xi) = \begin{cases} \mathbf{U}_L & \xi < \xi_L \\ \mathbf{U}^*(\xi) & \xi_L < \xi < \xi_R \\ \mathbf{U}_R & \xi_R < \xi. \end{cases} \tag{2.4.64}$$

- Shock wave. A discontinuity, we have to check the Generalized Rankine-Hugoniot conditions (2.4.52) for our choice of path. Once we have checked the conditions, we obtain a shock speed s. Then, the solution is given by

$$w_s(\xi) = \begin{cases} \mathbf{U}_L & \xi < s \\ \mathbf{U}_R & \xi > s. \end{cases} \tag{2.4.65}$$

- Contact Discontinuity. In this case, both the relation $\partial_\xi \mathbf{U} = a(\xi)\hat{r}(\mathbf{U}(\xi))$, and the Generalized Rankine-Hugoniot conditions hold. Since we get a discontinuity, we once again obtain the solution

$$w_c(\xi) = \begin{cases} \mathbf{U}_L & \xi < s \\ \mathbf{U}_R & \xi > s \end{cases}, \tag{2.4.66}$$

  where $s = \hat{\lambda}$ is the speed of the discontinuity which in this case equals $\hat{\lambda}$ itself.

Finally, we are able to give an existence and uniqueness theorem for the solution structure we want.

Theorem 2.20. *Let Φ be a family of paths fulfilling Definition 2.7. Suppose that the characteristic fields of $\hat{\mathbf{A}}$ are either genuinely nonlinear or linearly degenerate, and let $\hat{\mathbf{A}}$ be strictly hyperbolic with eigenvalues $\hat{\lambda}_1 < ... < \hat{\lambda}_N$. Let $\mathbf{U}_L, \mathbf{U}_R \in \Omega$. Then, given that $\|\mathbf{U}_L - \mathbf{U}_R\|$ is small enough, the Riemann problem* (2.4.50) *has a unique Φ-weak solution $\mathbf{U}^*$ with bounded variation which which consists of $(N+1)$ states $\mathbf{U}_L = \mathbf{U}_0, \mathbf{U}_1, ..., \mathbf{U}_N = \mathbf{U}_R$ which are connected by simple waves. In particular, we have for all k that $\mathbf{U}_{k-1}$ and $\mathbf{U}_k$ are connected by*

- *a centered rarefaction, if $\hat{r}_k$ is genuinely nonlinear and Definition 2.19 fails to hold,*
- *a shock, if $\hat{r}_k$ is genuinely nonlinear and Definition 2.19 holds, or*
- *a contact discontinuity, if $\hat{r}_k$ is linearly degenerate.*

Proof. This is shown in [27, Section 5] for the conservative case. For the non-conservative case, it is cited as [34, Theorem 3.3] in the non-conservative case, and proof details for a more general setting can be found e.g. [29], and a sketch in [18, Theorem 3.1][13]. The latter mentions that we only need to exchange the definition of shocks. □

[13] The fourth assumption in [18, Theorem 3.1] is shown by Lemma 2.13, once Assumption 2.8 is fulfilled, since then we have

$$\partial_{\mathbf{U}_R} \Psi'[\mathbf{U}_L, \mathbf{U}_L](1) - \partial_{\mathbf{U}_R} \Psi'[\mathbf{U}_L, \mathbf{U}_L](0) = \partial_{\mathbf{U}_R} \int_{\Psi[\mathbf{U}_L, \mathbf{U}_L]} I \, \mathrm{d}\mathbf{U} = I. \tag{2.4.67}$$

If we from now on speak of a solution to the Riemann Problem, we speak of a solution of this type.

2.5 Examining the Riemann Problem

Next, we look at the Riemann Problem for (2.0.10). To our knowledge, it has not been solved before. The problem reads

$$\begin{aligned} &\partial_t \mathbf{U} + \mathbf{A}_1(\mathbf{U}) \cdot \nabla \mathbf{U} = 0 \\ &\mathbf{U}(0,x) = \begin{cases} \mathbf{U}_L & x < 0 \\ \mathbf{U}_R & x > 0 \end{cases} \end{aligned} \tag{2.5.1}$$

We may instead write (2.5) with regard to any other normal vector, since it is rotational invariant. Most notably, the dependence on $\mathbf{S}$ is removed. In comparison to models like the shallow water equations (2.1.1), this does alter our model. Hence the Riemann problem does not completely describe all of (2.0.10) anymore. However, we may nullify the effect of $\mathbf{S}$ (even when it is present) by setting $n_m = 0$, disabling wave breaking, and also $w(0) \equiv 0$ and $\gamma = 0$. The latter two ensure that $w \equiv 0$ over time, and thus there is no contribution to the equation concerning p.

In this analysis, we will skip the bathymetry for now, and consider the eigenvectors of $\mathbf{A}_1^*(\mathbf{U})$ instead. Information on how to potentially include the bathymetry in this analysis as well could be most likely also taken from corresponding papers which deal with the problems in the shallow water equations, see e.g. [28, 22]. From now on, we assume that $b_L = b_R$ for the rest of this section.

Next, we consider the subsystem r_1, r_2, r_3 which contains the non-hydrostatic pressure-velocity coupling and is subsequently the most complex to analyze. Scaled down, it reads

$$\partial_t \begin{pmatrix} h \\ hu \\ hp \end{pmatrix} + \partial_x \begin{pmatrix} hu \\ h(u^2+p) \\ hu(c^2+p) \end{pmatrix} + \begin{pmatrix} 0 \\ gh\partial_x h \\ -c^2 u \partial_x h \end{pmatrix} = 0 \tag{2.5.2}$$

With the theory we recapitulated in the last section, we obtain the following solution structure. Throughout this section, we will write $C_L := C(\mathbf{U}_L)$ and $C_R := C(\mathbf{U}_R)$.

Corollary 2.21. *Given a family of paths Ψ which is admissible in the sense of Definition 2.7, The equation system (2.5.2) has, for initial states $\mathbf{U}_L, \mathbf{U}_R$ which are close enough with $h_L, h_R > 0$, and $C_L, C_R > 0$, a unique solution of the structure as described in Theorem 2.20. In particular, there are two intermediate states $\mathbf{U}_{ML}$ and $\mathbf{U}_{MR}$, and we have that*

1. *the field of r_1 connects $\mathbf{U}_L$ and $\mathbf{U}_{ML}$ through a genuinely nonlinear field (which we call the "left genuinely nonlinear field"),*

2. *the field of r_2 connects $\mathbf{U}_{ML}$ and $\mathbf{U}_{MR}$ through a contact discontinuity (which we call the "central contact discontinuity"),*

3. *the field of r_3 connects $\mathbf{U}_{MR}$ and $\mathbf{U}_R$ through a genuinely nonlinear field (which we call the "right genuinely nonlinear field").*

Proof. Follows directly from Theorem 2.20. □

It should be noted that once we have the solution for the subsystem h, u, and p, we immediately obtain the solutions for v and w, since they are just passive contact discontinuities. Since $\lambda_v = \lambda_w = \lambda_2$, we need v and w to switch values when the field associated with r_2 switches its value (since it is a contact discontinuity). This means that we can simply set $v_{ML} = v_L$ and $v_{MR} = v_R$, and do the same for w.

Other than that, we will mostly deal with the subsystem concerning the equations h, hu, and hp. The eigenvectors r_v and r_w play only a passive role—for the latter especially, since we removed the source term. Nonetheless, we will include r_v and r_w in the analysis in the following sections, just to show that they do not play a role in the system r_1, r_2, r_3. This section is therefore concerned with computing the values for h_{ML} and h_{MR}, and also u_{ML}, u_{MR} and p_{ML}, p_{MR} for a possibly large amount of possible paths Ψ. In practice, we will consider mostly Ψ_C and Ψ_P.

In the following subsections, $\mathbf{U}_L$, $\mathbf{U}_{ML}$, $\mathbf{U}_{MR}$, and $\mathbf{U}_R$ denote state *vectors* from $\mathbb{R}^5$, and not functions.

2.5.1 Examining the Characteristic Fields

From now on, we assume that $h_R, h_L > 0$ and $C_R, C_L > 0$.

We now begin with the analysis of the system, and we do so by examining the characteristic fields.

The Contact Discontinuities

We start by discussing the Riemann invariants of the contact discontinuities. Again, the fields associated with λ_v and λ_w, respectively, are only passive and only receive influence of the h, u, p subsystem. Thus, we can give their Riemann invariants immediately. We look at r_v: we have the Riemann invariants h, hu, hw, and hp. For r_w, we get h, hu, hv, and hp.

Next, we look at the field associated with r_2. Here, we have that $i_{2,1} = \lambda_2 = u$ is constant which we checked in the last section. Also, we see that $i_{2,2} = \frac{1}{2}gh^2 + hp$ is constant as well: since $h' = 1$, integrating

$$(hp)'(\xi) = -a(\xi)gh(\xi) = (-gh)h' \tag{2.5.3}$$

gives

$$hp - h_0p_0 = -\frac{g}{2}\left(h^2 - h_0^2\right) \tag{2.5.4}$$

and so

$$i_{2,2} = hp + \frac{g}{2}h^2 = h_0 p_0 + \frac{g}{2}h_0^2 \tag{2.5.5}$$

And indeed, inserting it gives

$$\nabla i_{2,2} \cdot r_2 = gh - gh = 0 \tag{2.5.6}$$

One might notice that therefore, the whole characteristic field of λ_2 can be understood as an exchange between hydrostatic and non-hydrostatic pressure—while the total amount of pressure times water height hp_T is conserved. This gives us a generalized view onto the shallow water equations, where the conservation of hp_T conflates with the conservation of h itself.

Centered Rarefaction Solution

This part follows roughly [24, 30], since in essence, these parts can be handled like the respective procedure for the shallow water equations.

For the genuinely nonlinear fields r_1 and r_3, we begin by looking at the centered rarefaction solutions. That is, we want to solve ($a(\xi)$ is a normalization factor) the ordinary differential equation

$$\begin{aligned} \mathbf{U}'_{1,3}(\xi) &= a(\xi) r_{1,3}(\mathbf{U}_{1,3}) \\ \mathbf{U}_{1,3}(0) &= \mathbf{U}_0. \end{aligned} \tag{2.5.7}$$

Here, $\mathbf{U}_0$ is a given initial vector. Expanding the equation gives us

$$\begin{aligned} h'(\xi) &= a(\xi) \\ (hu)'(\xi) &= a(\xi)\left(u \pm \sqrt{gh + p + c^2}\right) \\ (hv)'(\xi) &= a(\xi) v \\ (hw)'(\xi) &= a(\xi) w \\ (hp)'(\xi) &= a(\xi)(p + c^2), \end{aligned} \tag{2.5.8}$$

and initial values

$$\begin{aligned} h(0) &= h_0 \\ hu(0) &= h_0 u_0 \\ hv(0) &= h_0 v_0 \\ hw(0) &= h_0 w_0 \\ hp(0) &= h_0 p_0. \end{aligned} \tag{2.5.9}$$

In our case, it is convenient to switch to primitive variables first. So, exemplary done for hu, we write

$$(hu)' = h'u + hu' = au + hu'. \tag{2.5.10}$$

Here, $a(\xi)$ is the normalization function from above. Thus, by solving for u', v', w', and p' respectively gives us

$$\begin{array}{rcl} h'(\xi) & = & a(\xi) \\ u'(\xi) & = & \pm\frac{a}{h}\sqrt{gh+p+c^2} \\ v'(\xi) & = & 0 \\ w'(\xi) & = & 0 \\ p'(\xi) & = & c^2\frac{a}{h}. \end{array} \tag{2.5.11}$$

Next, we remove a by inserting the first equation into the other four. This yields the system

$$\begin{array}{rcl} u'(\xi) & = & \pm\frac{h'}{h}\sqrt{gh+p+c^2} \\ v'(\xi) & = & 0 \\ w'(\xi) & = & 0 \\ p'(\xi) & = & c^2\frac{h'}{h}. \end{array} \tag{2.5.12}$$

Now we may start solving the equations. To begin with, we get that $v = v_0$, and $w = w_0$ which give us two Riemann invariants, so v and w stay constant throughout both r_1 and r_3. Secondly, we also note that the equation for p' does now not depend on p itself anymore. So we integrate it to get

$$p(\xi) - p_0 = c^2\left(\log h(\xi) - \log h_0\right). \tag{2.5.13}$$

Note that we substituted with h on the right-hand side. Equivalently, re-writing it in terms of h, we get

$$p(h) = c^2 \log\left(\frac{h}{h_0}\right) + p_0. \tag{2.5.14}$$

We also get a Riemann invariant re-writing it to

$$i_{1,1} = i_{3,1} = p - c^2 \log h = p_0 - \log h_0. \tag{2.5.15}$$

And indeed, we have that

$$\nabla i_{1,1}\cdot r_1 = \begin{pmatrix} -\frac{hp}{h^2} - \frac{c^2}{h} \\ 0 \\ 0 \\ 0 \\ \frac{1}{h} \end{pmatrix} \cdot \begin{pmatrix} 1 \\ \left(u \pm \sqrt{gh+p+c^2}\right) \\ v \\ w \\ p+c^2 \end{pmatrix} = -\frac{p}{h} - \frac{c^2}{h} + \frac{1}{h}(p+c^2) = 0. \tag{2.5.16}$$

Furthermore, since u' only depends on h and p, we get

$$\begin{aligned} u_\pm(\xi) - u_0 &= \pm\int_{h_0}^{h(\xi)} \frac{\sqrt{gs+p(s)+c^2}}{s}\,\mathrm{d}s \\ &= \pm\int_{h_0}^{h(\xi)} \frac{\sqrt{gs+c^2\left(\log\left(\frac{s}{h_0}\right)+1\right)+p_0}}{s}\,\mathrm{d}s. \end{aligned} \tag{2.5.17}$$

Thus, we may write u dependent on h

$$u_{\pm}(h) = u_0 \pm \int_{h_0}^{h} \frac{\sqrt{gs + c^2\left(\log\left(\frac{s}{h_0}\right) + 1\right) + p_0}}{s}\,\mathrm{d}s. \tag{2.5.18}$$

We can also (this is convenient when evaluating the integral) substitute the log inside to get the term

$$u_{\pm}(h) = u_0 \pm \int_{\log h_0}^{\log h} \sqrt{ge^t + c^2 t + c^2\left(1 - \log h_0\right) + p_0}\,\mathrm{d}t. \tag{2.5.19}$$

Unfortunately, the integral cannot be integrated exactly anymore using elementary and trigonometric functions, at least multiple CAS did not give a positive answer. Most notably since we have the sum of linear, and an exponential term under the square root. For reference, integrating $g(x) = \sqrt{\log(x)}$ gave a solution that already involved an integral over e^{x^2}. Likewise for $u_{\pm}(h)$, we would get an integral over $W(e^{x^2})$ where W is the Lambert function [8]. On the opposite, once $g = 0$ or $c = 0$, a closed-form representation could be found.

However, it is important to note that $u_{\pm}(h)$ is cannot be defined anymore for all $h > 0$, if we want u to have a value in $\mathbb{R}$. While $u_{\pm}(h)$ is real for all $h \geq h_0$, we have that for $s \to 0$, we have

$$gs + c^2\left(\log\left(\frac{s}{h_0}\right) + 1\right) + p_0 \to -\infty, \tag{2.5.20}$$

and hence $u(h)$ is not real anymore. If we define $C_{\mathbf{U}_0}(h)$ as

$$C_{\mathbf{U}_0}(h) = gh + c^2\left(\log\left(\frac{h}{h_0}\right) + 1\right) + p_0, \tag{2.5.21}$$

we obtain the following result.

Lemma 2.22. *Given $g > 0$, and an initial state $\mathbf{U}_0$ with $h_0 > 0$, $C_{\mathbf{U}_0}$ is strictly monotonically increasing. In particular, there is a unique value $h_{low} = C^{-1}_{\mathbf{U}_0}(0)$ for h, so that for $h < h_{low}$, we have that $\mathbf{A}_1^*(\mathbf{U})$ has imaginary eigenvalues, and for $h = h_{low}$, the matrix $\mathbf{A}_1^*(\mathbf{U})$ has only three eigenvalues.*

Proof. Calculating the derivative gives us the result. It is given as

$$C'_{\mathbf{U}_0}(h) = g + \frac{c^2}{h} > 0. \tag{2.5.22}$$

Therefore $C'_{\mathbf{U}_0}(h)$ is strictly montononically increasing. □

Thus, we need to restrict the area of definition of $u_{\pm}(h)$ to $[C^{-1}_{\mathbf{U}_0}(0), \infty)$. (we allow the corner case, where we lose hyperbolicity, since $u(h)$ is perfectly valid there) For $p(h)$, we get that $(0, \infty)$ can be chosen for the area of definition.

2.5.2 The Rankine-Hugoniot Conditions

Next, we look at the Rankine-Hugoniot conditions. Essentially, we proceed similarly to [24, 30, 43]. Also, this is the only place, where we are actually required to evaluate paths. We begin with (2.4.52), and then we do the variable transformation $\hat{u} = u - s$, and $\hat{\mathbf{U}} = (h, h\hat{u}, hv, hw, hp, b)^T$. We still assume $b_L = b_R$ for now. Given $\hat{u}$, we can re-write the first equation

$$s(h_R - h_L) = h_R u_R - h_L u_L \tag{2.5.23}$$

to

$$h_R \hat{u}_R = h_L \hat{u}_L. \tag{2.5.24}$$

This can be done for the other equations as well, although for

$$s(h_R u_R - h_L u_L) = h_R u_R^2 - h_L u_L^2 + \frac{g}{2}\left(h_R^2 - h_L^2\right) + h_R p_R - h_L p_L, \tag{2.5.25}$$

we additionally need to use that

$$\hat{u}_R - \hat{u}_L = u_R - u_L. \tag{2.5.26}$$

Transforming all equations, we obtain

$$\begin{aligned}
h_L \hat{u}_L &= h_R \hat{u}_R \\
h_L \hat{u}_L^2 + h_L p_L + \frac{g}{2} h_L^2 &= h_R \hat{u}_R^2 + h_R p_R + \frac{g}{2} h_R^2 \\
h_L \hat{u}_L v_L &= h_R \hat{u}_R v_R \\
h_L \hat{u}_L w_L &= h_R \hat{u}_R w_R \\
h_L \hat{u}_L (p_L + c^2) &= h_R \hat{u}_R (p_R + c^2) - c^2 \int_{\Phi[\mathbf{U}_L, \mathbf{U}_R]} \hat{u} \, \mathrm{d}h.
\end{aligned} \tag{2.5.27}$$

We now assume $\hat{\mathbf{U}}_L$ to be known, and $\hat{\mathbf{U}}_R$ is unknown. In addition, we may switch $\hat{\mathbf{U}}_L$ and $\hat{\mathbf{U}}_R$ and get the same equations for the other variable. Just as in [43], we use the abbreviation

$$Q := h_L \hat{u}_L = h_R \hat{u}_R \tag{2.5.28}$$

which lets us re-write the other conditions (all but the first) as

$$\begin{aligned}
Q\hat{u}_L + h_L p_L + \frac{g}{2} h_L^2 &= Q\hat{u}_R + h_R p_R + \frac{g}{2} h_R^2 \\
Q v_L &= Q v_R \\
Q w_L &= Q w_R \\
Q p_L &= Q p_R - c^2 \int_{\Phi[\mathbf{U}_L, \mathbf{U}_R]} \hat{u} \, \mathrm{d}h.
\end{aligned} \tag{2.5.29}$$

We also removed $c^2 Q$ from the fourth equation on both sides. What will happen any further depends on Q and the family of paths we choose for the remaining integral. In the following, we need to distinguish the cases $Q = 0$ and $Q \neq 0$.

Choice of Paths

Next, we will discuss some simple choices for the path integral, so that we may resolve the equation concerning p. In order to cover a larger possible family of paths, we just put together all assumptions we need to solve the problem in an easy way. So, we make the following assumption on Φ.

Assumption 2.23. *Given a family of paths Φ defined for $C_R, C_L > 0$ and $h_R, h_L > 0$, there is a function $p_\Psi(h_L, h_R)$, so that we obtain*

$$Qp_\Psi(h_L, h_R) = \int_{\Phi[\mathbf{U}_L, \mathbf{U}_R]} \hat{u}\, dh. \tag{2.5.30}$$

In addition, we require if $Q \neq 0$, that

1. *$p_\Psi(h_L, h_R)$ is continuously differentiable in both arguments*
2. *$p_\Psi(h_L, h_R)$ is strictly monotonically decreasing in the first argument, strictly monotonically increasing in the second argument*
3. *We have that $p_\Psi(h_L, h_R) = -p_\Psi(h_R, h_L)$ for all $h_R, h_L > 0$*
4. *The function*
$$q_\Psi(h_L, h_R) := \frac{h_R p_\Psi(h_L, h_R)}{h_R - h_L} \tag{2.5.31}$$
is strictly monotonically increasing in h_R for $h_R > 0$ and $h_R \neq h_L$.

For brevity, we will write $\partial_R p_\Psi(h_L, h_R)$ for $\partial_{h_R} p_\Psi(h_L, h_R)$, and for h_L likewise.

Given Assumption 2.23, we can rewrite the fourth equation as

$$0 = Q(p_R - p_L - c^2 p_\Psi(h_L, h_R)). \tag{2.5.32}$$

The last assumption assures that we may freely swap h_L and h_R.

We also get the following results for families of paths which have such a function p_Ψ from Assumption 2.23.

Lemma 2.24. *Suppose $Q \neq 0$, and p_Ψ is the function from Assumption 2.23. Then, we obtain for all $h > 0$ that*

$$p_\Psi(h, h) = 0 \tag{2.5.33}$$

$$\partial_{h_R} p_\Psi(h, h) = \frac{1}{h} \tag{2.5.34}$$

$$\partial_{h_L} p_\Psi(h, h) = -\frac{1}{h}. \tag{2.5.35}$$

Proof. This follows directly from Lemmas 2.9 and 2.13 and the definition of p_Ψ from Assumption 2.23. In particular, we have that by $h > 0$ we obtain

$$\partial_R p_\Psi(h, h) = \frac{\hat{u}}{Q} = \frac{Q}{Qh} = \frac{1}{h}. \tag{2.5.36}$$

(alternatively, two of the three statements could be obtained from the Assumption 2.23 by $p_\Psi(h_L, h_R) = -p_\Psi(h_R, h_L)$, without even touching Lemma 2.9) □

It remains to give some families of paths which fulfill Assumption 2.23. Namely, we firstly discuss taking linear paths for h and hu, i.e. we take Ψ_C. Secondly, we discuss taking linear paths for h and u; i.e. we take Ψ_P. Both will only matter for $Q > 0$.

For Ψ_C, we get by the fact that $Q = h_L\hat{u}_L = h_R\hat{u}_R$ that we obtain

$$h\hat{u}(t) = (hu)(t) - sh(t) = h_L\hat{u}_L(1-t) + h_R\hat{u}_R t = Q \tag{2.5.37}$$

Thus, the path for $h\hat{u}$ is constant Q over the whole path. Therefore, we get

$$\begin{aligned} &\int_{\Psi_C[\mathbf{U}_L,\mathbf{U}_R]} \hat{u}\,\mathrm{d}h \\ &= \int_{\Psi_C[\mathbf{U}_L,\mathbf{U}_R]} \frac{h\hat{u}}{h}\,\mathrm{d}h \\ &= Q(\log(h_R) - \log(h_L)) \end{aligned} \tag{2.5.38}$$

We emphasize that we really needed here that h and hu followed the *same* path.

For Ψ_P, we compute

$$\begin{aligned} &\int_{\Psi_P[\mathbf{U}_L,\mathbf{U}_R]} \hat{u}\,\mathrm{d}h \\ &= \int_0^1 (u_L + t(u_R - u_L))(h_R - h_L) - s(h_R - h_L)\,\mathrm{d}t \\ &= \frac{((u_R + u_L) - 2s)(h_R - h_L)}{2} \\ &= \frac{(\hat{u}_R + \hat{u}_L)(h_R - h_L)}{2}. \end{aligned} \tag{2.5.39}$$

Given that $h_R, h_L > 0$, we may insert $Q = h_L\hat{u}_L = h_R\hat{u}_R$.

$$\begin{aligned} &\frac{(\hat{u}_R + \hat{u}_L)(h_R - h_L)}{2} \\ &= Q\frac{\left(\frac{1}{h_R} + \frac{1}{h_L}\right)(h_R - h_L)}{2} \\ &= Q\frac{h_R^2 - h_L^2}{2h_Rh_L} \end{aligned} \tag{2.5.40}$$

In total, we obtain the following result. Note that if $Q = 0$, the exact value of p_Ψ does not matter anymore.

Proposition 2.25. *Both Ψ_P and Ψ_C fulfill Assumption 2.23. We obtain the functions*

$$p_P(h_L, h_R) := p_{\Psi_P}(h_L, h_R) = \frac{h_R^2 - h_L^2}{2h_Rh_L}, \tag{2.5.41}$$

and

$$p_C(h_L, h_R) := p_{\Psi_C}(h_L, h_R) = \log(h_R) - \log(h_L). \tag{2.5.42}$$

From now on, we will use p_Ψ and therefore assume that Assumption 2.23 holds for the family of paths we chose.

Shock Solution: $Q \neq 0$

We continue now with the analysis of the Rankine-Hugoniot conditions. If $Q \neq 0$ in the Rankine-Hugoniot conditions, we get $v_R = v_L$, $w_R = w_L$. Secondly, we get

$$\hat{u}_R = \frac{Q}{h_R}, \hat{u}_L = \frac{Q}{h_L}. \tag{2.5.43}$$

This is the case, since $h_L, h_R > 0$ has to hold by $Q \neq 0$. With that, we may re-write the first equation as

$$\frac{Q^2}{h_L} + h_L p_L + \frac{g}{2} h_L^2 = \frac{Q^2}{h_R} + h_R p_R + \frac{g}{2} h_R^2. \tag{2.5.44}$$

We reformulate this further to

$$Q^2 \left(\frac{1}{h_L} - \frac{1}{h_R} \right) = h_R p_R - h_L p_L + \frac{g(h_R^2 - h_L^2)}{2}, \tag{2.5.45}$$

and after some more manipulations, we get to

$$Q^2 = h_L h_R \left(g \frac{h_R + h_L}{2} + \frac{h_R p_R - h_L p_L}{h_R - h_L} \right). \tag{2.5.46}$$

In addition, we recapitulate our re-written fourth equation which reads now

$$Q p_L = Q p_R - Q c^2 p_\Psi(h_L, h_R). \tag{2.5.47}$$

Dividing by Q gives us

$$p_R(h_R) = p_L + c^2 p_\Psi(h_L, h_R). \tag{2.5.48}$$

Inserting the fourth equation into the first one yields

$$Q^2 = h_L h_R \left(g \frac{h_R + h_L}{2} + p_L + c^2 \frac{h_R p_\Psi(h_L, h_R)}{h_R - h_L} \right). \tag{2.5.49}$$

We may take the square root of it which solves Q, but gives us two values for it, namely

$$Q_\pm = \pm \sqrt{h_L h_R \left(\frac{h_R + h_L}{2} g + p_L + c^2 \frac{h_R p_\Psi(h_L, h_R)}{h_R - h_L} \right)}. \tag{2.5.50}$$

Given $Q_\pm$, we now solve for s, and obtain

$$s^{\mp}_{\mathbf{U}_L}(h_R) = u_L - \frac{Q_\pm}{h_L} = u_L \mp \sqrt{\frac{h_R}{h_L} \left(\frac{h_R + h_L}{2} g + p_L + c^2 \frac{h_R p_\Psi(h_L, h_R)}{h_R - h_L} \right)}. \tag{2.5.51}$$

We write $s^+_{\mathbf{U}_L}$ for the shock speed expression that contains a plus, and $s^-_{\mathbf{U}_L}$ which contains a minus. In particular, we also get an expression for u_R which reads

$$\begin{aligned} u_R^{\mp}(h_R) &= u_L + \left(\frac{1}{h_R} - \frac{1}{h_L} \right) Q_\pm \\ &= u_L \mp \frac{h_R - h_L}{h_R h_L} \sqrt{h_L h_R \left(\frac{h_R + h_L}{2} g + p_L + c^2 \frac{h_R p_\Psi(h_L, h_R)}{h_R - h_L} \right)} \\ &= u_L \mp \sqrt{\frac{h_R - h_L}{h_R h_L} \left(\frac{h_R^2 - h_L^2}{2} g + (h_R - h_L) p_L + c^2 h_R p_\Psi(h_L, h_R) \right)}. \end{aligned} \tag{2.5.52}$$

Once again, we assign the plus sign to u_R^-, and the minus sign to u_R^+. Similarly, by resolving the Rankine-Hugoniot conditions for $\hat{\mathbf{U}}_L$ and leaving $\hat{\mathbf{U}}_R$ fixed, we obtain with the signs already switched

$$s^{\pm}_{\mathbf{U}_R}(h_L) = u_R \pm \sqrt{\frac{h_L}{h_R}\left(\frac{h_L+h_R}{2}g + p_R + c^2\frac{h_L p_\Psi(h_R,h_L)}{h_L-h_R}\right)}. \tag{2.5.53}$$

Similar terms can be derived for $u_L^{\pm}$. In particular, we have by the symmetry that for given $\mathbf{U}_R, \mathbf{U}_L$ it holds

$$s^{\pm}_{\mathbf{U}_R}(h_L) = s^{\pm}_{\mathbf{U}_L}(h_R). \tag{2.5.54}$$

About the Shock Speed

Next, we are going to discuss the shock speed in more detail.

We begin by its area of definition. Since $s^{\pm}_{\mathbf{U}_L}$ contains a square root, we naturally need

$$\frac{h_R+h_L}{2}g + p_L + c^2\frac{h_R p_\Psi(h_L,h_R)}{h_R-h_L} \geq 0. \tag{2.5.55}$$

Firstly, we consider

$$\frac{p_\Psi(h_L,h_R)}{h_R-h_L}. \tag{2.5.56}$$

However, since we assumed that $p_\Psi(h_L,h_R)$ is strictly increasing in the second argument, and $p_\Psi(h_L,h_L)=0$, we obtain that for both $h_R > h_L$ and $h_R < h_L$, we have

$$\frac{p_\Psi(h_L,h_R)}{h_R-h_L} \geq 0. \tag{2.5.57}$$

In the case $h_R < h_L$, both $p_\Psi(h_L,h_R)$ and (h_R-h_L) are negative, so the sign gets cancelled out. Additionally, we get for $h_R \to h_L$ that

$$\frac{p_\Psi(h_L,h_R)}{h_R-h_L} \to \frac{1}{h_L}, \tag{2.5.58}$$

due to Lemma 2.24. However, p_L may be negative. So, we obtain the bound that we require

$$\frac{h_R+h_L}{2}g + c^2\frac{h_R p_\Psi(h_L,h_R)}{h_R-h_L} > -p_L. \tag{2.5.59}$$

The left-hand side is strictly monotonically increasing by Assumption 2.23. Thus, if these is a $h_R^* > 0$, so that

$$\frac{h_R^*+h_L}{2}g + c^2\frac{h_R^* p_\Psi(h_L,h_R^*)}{h_R^*-h_L} = -p_L, \tag{2.5.60}$$

then it is unique. If it does not exist, we set $h_R^* = 0$. Thus, we need to restrict the area of definition for $s^{\pm}_{\mathbf{U}_L}$ to $(h_R^*,\infty)\backslash\{h_L\}$.

We may now prove some simple properties about $s^{\pm}_{\mathbf{U}_L}$ which help us for checking the Lax Entropy Condition (Definition 2.19). They can be subsequently extended once again to $s^{\pm}_{\mathbf{U}_R}$, if we simply swap $\mathbf{U}_R$ and $\mathbf{U}_L$.

Lemma 2.26. *The following properties hold about $s^{\pm}_{\mathbf{U}_L}$:*

1. *For $h_R \to h_L$ (given $h_L > 0$), we have that $s^{+}_{\mathbf{U}_L} \to \lambda_3$ and $s^{-}_{\mathbf{U}_L} \to \lambda_1$.*
2. *For $h_R \to \infty$, we get $s^{+}_{\mathbf{U}_L} \to \infty$, and $s^{-}_{\mathbf{U}_L} \to -\infty$.*
3. *$s^{+}_{\mathbf{U}_L}$ is strictly increasing on its area of definition, and $s^{-}_{\mathbf{U}_L}$ is decreasing on its area of definition.*

Proof. For convenience, we define

$$t(h_R) = \frac{h_R}{h_L}\left(\frac{h_R + h_L}{2}g + p_L + c^2\frac{h_R p_\Psi(h_L, h_R)}{h_R - h_L}\right) \tag{2.5.61}$$

Then, we may prove the above given properties for t and extend them to $s^{\pm}_{\mathbf{U}_L}$ later-on.

By Lemma 2.24, similarly to what we have just done before, we have that for $h_R \to h_L$ that

$$\frac{h_R p_\Psi(h_L, h_R)}{h_R - h_L} \to h_L(\partial_L p_\Psi(h_L, h_L)) = 1. \tag{2.5.62}$$

Thus, we obtain for t that

$$t(h_R) \to h_L g + p_L + c^2. \tag{2.5.63}$$

This proves the first assertion for $s^{\pm}_{\mathbf{U}_L}$ by using the continuity of the square root.

For the second assertion, we again compute the limit for t to get for $h_R \to \infty$ that

$$t(h_R) \geq \frac{h_R}{h_L}\left(\frac{h_R + h_L}{2}g + p_L\right) \to \infty. \tag{2.5.64}$$

We simply removed $p_\Psi(h_L, h_R)$ which is a strictly increasing function. Once $h_R > h_L$, we have $p_\Psi(h_L, h_R) > 0$ by Lemma 2.24. This shows the second assertion for $s^{\pm}_{\mathbf{U}_L}$ as well.

The third assertion is given by Assumption 2.23, since

$$\frac{h_R p_\Psi(h_L, h_R)}{h_R - h_L} \tag{2.5.65}$$

is strictly monotonically increasing by it. Thus, $t(h_R)/h_R$ is strictly monotonically increasing, and so $t(h_R)$ is strictly monotonically increasing as well. □

As an immediate consequence, we get a statement about the Lax Entropy Condition (Definition 2.19), and the genuinely nonlinear fields.

Lemma 2.27. *Suppose $h_R, h_L > 0$ and $C_R, C_L > 0$. Suppose that h_R lies in the area of definition of $s^{\pm}_{\mathbf{U}_L}$. Then fields associated with r_1 and r_3 fulfill the entropy condition, if the following conditions are fulfilled.*

- *For the field associated with r_1, it is fulfilled, iff $h_L < h_R$*
- *For the field associated with r_3, it is fulfilled, iff $h_R < h_L$*

Proof. We will show this only for $s^+_{\mathbf{U}_L}(h_R)$ and r_3, the other case goes similarly. So, we need to show that

$$\lambda_{2,L} < s^+_{\mathbf{U}_L}(h_R), \tag{2.5.66}$$
$$\lambda_{3,L} > s^+_{\mathbf{U}_L}(h_R), \tag{2.5.67}$$
$$\lambda_{3,R} < s^+_{\mathbf{U}_L}(h_R). \tag{2.5.68}$$

The first inequality reads

$$u_L < s^+_{\mathbf{U}_L}(h_R), \tag{2.5.69}$$

and it is automatically fulfilled in the whole area of definition of $s^\pm_{\mathbf{U}_L}$. The second inequality

$$u_L + \sqrt{gh_L + p_L + c^2} > s^+_{\mathbf{U}_L}(h_R) \tag{2.5.70}$$

follows from Lemma 2.26, due to the strict monotonicity, and $s^+_{\mathbf{U}_L}(h_R) \to \lambda_{3,L}$ for $h_R \to h_L$. As for the third inequality, we can rewrite it to

$$u_R + \sqrt{gh_R + p_R + c^2} = \lambda_{3,R} < s^+_{\mathbf{U}_R}(h_L) = s^+_{\mathbf{U}_L}(h_R). \tag{2.5.71}$$

Then, by applying Lemma 2.26 onto $s^+_{\mathbf{U}_L}(h_R)$, we obtain that this is exactly the case, if and only if $h_R < h_L$. □

The last question with which we need to concern us is when h_R lies in the area of definition of $s^\pm_{\mathbf{U}_L}$. For the scope of this work, we will skip this question and simply assume that h_R always lies in the area of definition of $s^\pm_{\mathbf{U}_L}$, when we need it. In fact, it is enough to demand that $C^{-1}_{\mathbf{U}_L}(0) \geq h^*_R$, since then we will always lose hyperbolicity through a rarefaction, before $s^\pm_{\mathbf{U}_L}$ cannot be defined anymore.

The Case $Q = 0$

If we have $Q = 0$, we cannot predict anything about v_R nor about w_R. The first equation reduces to

$$h_L p_L + \frac{g}{2}h_L^2 = h_R p_R + \frac{g}{2}h_R^2 \tag{2.5.72}$$

which is exactly the Riemann invariant $i_{2,2}$. The last equation becomes (for $c \neq 0$)

$$\int_{\Phi[\mathbf{U}_L, \mathbf{U}_R]} \hat{u}\, \mathrm{d}h = 0. \tag{2.5.73}$$

If $h_L, h_R > 0$, then it means that $\hat{u}_L = \hat{u}_R = 0$, and so $s = \hat{u}_L = \hat{u}_R$. This gives us the following result.

Lemma 2.28. *Suppose $h_L, h_R > 0$. Then field associated with r_2 fulfills the Lax Entropy Condition (Definition 2.19).*

Once $h_L = 0$ or $h_R = 0$, we will need special treatment [24], so we postpone this case for later.

2.5.3 Constructing the Intermediate States

Next, we join the states together, as Theorem 2.20 and Corollary 2.21 told us that we can do so, given that $\mathbf{U}_L$ and $\mathbf{U}_R$ lie close to each other, and $C(\mathbf{U}_L), C(\mathbf{U}_R) > 0$ and $h_L, h_R > 0$.

The Genuinely Nonlinear Fields

We begin by looking at the genuinely nonlinear fields. Our goal is now to build, given an initial state $\mathbf{U}_0$, to build a function which gives us the state $\mathbf{U}$ on the other side, if we know its height h. Of course, we need the functions to obey the Lax Entropy Condition (Definition 2.19) in order to fulfill Corollary 2.21. Thus, we obtain a case distinction: if $h > h_0$, we take the shock solution, and it $h \leq h_0$, we take the rarefaction solution. In the case $h = h_0$, we get that both cases conflate.

For the non-hydrostatic pressure in the genuinely nonlinear fields, we get the function

$$p_{\mathbf{U}_0}(h) = \begin{cases} p_0 + c^2 p_\Psi(h_0, h) & h > h_0 \\ p_0 + c^2 \log(\frac{h}{h_0}) & h \leq h_0. \end{cases} \tag{2.5.74}$$

which is defined for $(0, \infty)$. We have $p_{\mathbf{U}_0}(h_0) = p_0$, and so $p_{\mathbf{U}_0}$ is continuous, since $p_\Psi(h_0, h_0) = 0$. For the velocity, we get for the genuinely nonlinear fields the function

$$u^\pm_{\mathbf{U}_0}(h) = \begin{cases} u_0 \pm \sqrt{\frac{h-h_0}{hh_0}\left(\frac{h^2-h_0^2}{2} g + (h-h_0)p_0 + c^2 h p_\Psi(h_0, h)\right)} & h > h_0 \\ u_0 \pm \int_{\log(h_0)}^{\log(h)} \sqrt{ge^s + c^2(s + 1 - \log(h_0)) + p_0}\, \mathrm{d}s & h \leq h_0. \end{cases} \tag{2.5.75}$$

which is defined for $[C^{-1}_{\mathbf{U}_0}(0), \infty)$. Once again, we have $u^\pm_{\mathbf{U}_0}(h_0) = u_0$, and so $u^\pm_{\mathbf{U}_0}$ are both continuous.

To begin with, we check that both of these functions are continuously differentiable.

Lemma 2.29. *Suppose $h, h_0 > 0$. Then $p_{\mathbf{U}_0}$ is continuously differentiable, and strictly monotonously increasing once $c > 0$.*

Proof. We only need to check $h = h_0$, since for $h < h_0$ it is clear, and for $h > h_0$ we have Assumption 2.23. Once again, we only need to check if the one-sided derivatives for $h \downarrow h_0$ (which we write ∂_h^+) and $h \uparrow h_0$ (which we write ∂_h^-) yield the same value. We compute them using Lemma 2.24, and obtain

$$\partial_h^+ p_{\mathbf{U}_0}(h_0) = c^2 \partial_R p_\Psi(h_0, h_0) = \frac{c^2}{h_0}, \tag{2.5.76}$$

$$\partial_h^- p_{\mathbf{U}_0}(h_0) = \frac{c^2}{h_0}. \tag{2.5.77}$$

Thus, the derivatives from both sides are the same, and thus p is differentiable. The monotonicity is also visible from the derivative, since for $h < h_0$, we have

$$p_{\mathbf{U}_0}(h) = \frac{c^2}{h} > 0, \tag{2.5.78}$$

and for $h > h_0$, we obtain the result immediately from Assumption 2.23. □

Lemma 2.30. *Suppose $h, h_0 > 0$ and $C_0 = C_{\mathbf{U}_0}(h_0) > 0$. In addition, suppose that $h \geq C^{-1}_{\mathbf{U}_0}(0)$. Then, $u^{\pm}$ is continuously differentiable. In addition, $u^{+}_{\mathbf{U}_0}$ is strictly monotonically increasing, and $u^{-}_{\mathbf{U}_0}$ is strictly monotonically decreasing.*

We will prove the Lemma later-on.

Combining States

For the two genuinely nonlinear fields, we introduce the middle left and middle right states, written $\mathbf{U}_{ML}$ and $\mathbf{U}_{MR}$ respectively. This means that it holds

$$u_{ML} = u^{-}_{\mathbf{U}_L}(h_{ML}) \tag{2.5.79}$$
$$u_{MR} = u^{+}_{\mathbf{U}_R}(h_{MR}) \tag{2.5.80}$$
$$p_{ML} = p_{\mathbf{U}_L}(h_{ML}) \tag{2.5.81}$$
$$p_{MR} = p_{\mathbf{U}_R}(h_{MR}). \tag{2.5.82}$$

The two genuinely nonlinear fields are connected through the central contact discontinuity, for which we know that u and $\frac{gh^2}{2} + hp$ are both constant throughout it. In other words, the contact discontinuity connects the middle left and right states with each other. In particular, we get

$$\frac{gh^2_{ML}}{2} + h_{ML}p_{ML} = \frac{gh^2_{MR}}{2} + h_{MR}p_{MR} \tag{2.5.83}$$
$$u_{ML} = u_{MR}. \tag{2.5.84}$$

Combining all these equations, we next introduce the short-hand function

$$\phi_{\mathbf{U}_0}(h) = \frac{gh^2}{2} + hp_{\mathbf{U}_0}(h). \tag{2.5.85}$$

In addition, we set $\phi_{\mathbf{U}_0}(0) = 0$ which makes $\phi_{\mathbf{U}_0}$ continuous also in 0. Inserting ϕ gives the two equations

$$\begin{aligned} \phi_{\mathbf{U}_L}(h_{ML}) &= \phi_{\mathbf{U}_R}(h_{MR}) \\ u^{-}_{\mathbf{U}_L}(h_{ML}) &= u^{+}_{\mathbf{U}_R}(h_{MR}). \end{aligned} \tag{2.5.86}$$

We now have a system with two equations and two unknowns which we could now try to solve for both of them, given that we knew that it was solvable. As it turns out, this is the case under certain conditions. We will continue to prove it now. Conceptually, the proof is essentially only based on applying monotonicity in a suitable way (it was already hinted at in e.g. [20, Appendix B]).

To begin with, we note that there is only a single central state for both u and ϕ which we will call u_M and ϕ_M, respectively. Then, we have

$$\begin{aligned} \phi_M &= \phi_{\mathbf{U}_L}(h_{ML}) = \phi_{\mathbf{U}_R}(h_{MR}) \\ u_M &= u^{-}_{\mathbf{U}_L}(h_{ML}) = u^{+}_{\mathbf{U}_R}(h_{MR}). \end{aligned} \tag{2.5.87}$$

Next, we look at ϕ, and compute its derivative. We already know that ϕ is continuously differentiable for $h > 0$ by Lemma 2.29. We obtain

$$\phi'_{\mathbf{U}_0}(h) = \begin{cases} C_{\mathbf{U}_0}(h) + c^2(h\partial_R p_\Psi(h_0, h) - 1) & h > h_0 \\ C_{\mathbf{U}_0}(h) & h \leq h_0 \end{cases}. \tag{2.5.88}$$

It is immediate that $\phi_{\mathbf{U}_0}$ is continuously differentiable even for $h = h_0$ since it is built out of continuously differentiable functions, due to Lemma 2.29. If we look at $h = h_0$, we obtain indeed that

$$\begin{aligned} &\lim_{h \downarrow h_0} \phi'_{\mathbf{U}_0}(h) \\ &= C_{\mathbf{U}_0}(h_0) + c^2 h_0 \partial_R p_\Psi(h_0, h_0) - c^2 \\ &= C_{\mathbf{U}_0}(h_0) \\ &= \lim_{h \uparrow h_0} \phi'_{\mathbf{U}_0}(h). \end{aligned} \tag{2.5.89}$$

In addition, we are able to prove the following about ϕ.

Lemma 2.31. *Suppose $h, h_0 > 0$, $C_0 > 0$, and $g > 0$. Then, $\phi_{\mathbf{U}_0}$ is a strictly convex function on $(0, \infty)$ with a global minimum at $C^{-1}_{\mathbf{U}_0}(0)$. For $h > h_0$, we have that $\phi_{\mathbf{U}_0}$ is strictly monotonously increasing.*

Proof. We need to distinguish the cases $c = 0$ and $c > 0$. We begin with $c > 0$. Then, the statement follows for $0 < h < h_0$ by noting that $C^{-1}_{\mathbf{U}_0}(0) \leq h$, thus $\phi'_{\mathbf{U}_0}(h) = C_{\mathbf{U}_0}(h)$ which is a strictly monotonically increasing function for $h > 0$, thus ϕ is strictly convex for $h < h_0$, and $C^{-1}_{\mathbf{U}_0}(0)$ is the unique minimum.

For $h_0 < h$, we have that $p_{\mathbf{U}_0}$ is strictly monotonically increasing, and $gh^2/2$ is also strictly monotonically increasing, thus $\phi_{\mathbf{U}_0}$ is build out of a strictly convex part ($h_0 > h > 0$) connected with a strictly monotonous part ($h_0 < h$) which in total is strictly convex as a whole. And for $h = h_0$, we have $\phi'_{\mathbf{U}_0}(h_0) = C_{\mathbf{U}_0}(h_0) = C_0 > 0$. This shows that $\phi_{\mathbf{U}_0}$ is a strictly convex function, and strictly increasing for $h > h_0$.

Now, we turn to $c = 0$. Then, we can use for $h \geq 0$ once again that $x \mapsto gx^2/2$ is strictly increasing on $\mathbb{R}$, thus also for the restriction onto $[0, \infty)$. □

Therefore, we get that $\phi_{\mathbf{U}_0}$ is strictly monotonous for $C_{\mathbf{U}_0}(h) \geq 0$ which is the required condition for our system to stay hyperbolic. We are therefore going to write $\phi^{-1}_{\mathbf{U}_0}$ for the inverse of $\phi_{\mathbf{U}_0}$ on $[C^{-1}_{\mathbf{U}_0}(0), \infty)$. Interestingly enough, ϕ can also be continuously extended for $h \to 0$.

Next, we examine $u^\pm$ a bit closer. Given ϕ, we may re-write $u^\pm$ to

$$u^\pm_{\mathbf{U}_0}(h) = \begin{cases} u_0 \pm \sqrt{\frac{h - h_0}{h h_0} \left(\phi_{\mathbf{U}_0}(h) - \phi_{\mathbf{U}_0}(h_0)\right)} & h > h_0 \\ u_0 \pm \int_{h_0}^{h} \frac{\sqrt{C_{\mathbf{U}_0}(s)}}{s} \, \mathrm{d}s & h \leq h_0 \end{cases} \tag{2.5.90}$$

Once again, we know that $u^{\pm}$ is continuously differentiable, since ϕ and C are. For the derivative of $u^{\pm}$, we obtain

$$u'^{\pm}_{\mathbf{U}_0}(h) = \pm \begin{cases} \frac{1}{\sqrt{hh_0}} \frac{\left(\phi_{\mathbf{U}_0}(h)-\phi_{\mathbf{U}_0}(h_0)\right)+(h-h_0)\phi'_{\mathbf{U}_0}(h)}{2\sqrt{(h-h_0)\left(\phi_{\mathbf{U}_0}(h)-\phi_{\mathbf{U}_0}(h_0)\right)}} & h > h_0 \\ \frac{\sqrt{C_{\mathbf{U}_0}(h)}}{h} & h \leq h_0 \end{cases}, \tag{2.5.91}$$

We can also directly show that for $h = h_0$ the left-hand and right-hand side derivatives coincide, as can be seen by the computation

$$\begin{aligned} &\lim_{h\downarrow h_0} u'^{\pm}_{\mathbf{U}_0}(h) \\ &= \frac{\pm 1}{2h_0} \lim_{h\downarrow h_0} \left(\frac{\sqrt{\phi_{\mathbf{U}_0}(h)-\phi_{\mathbf{U}_0}(h_0)}}{\sqrt{h-h_0}} + \frac{\sqrt{h-h_0}\phi'_{\mathbf{U}_0}(h)}{\sqrt{\phi_{\mathbf{U}_0}(h)-\phi_{\mathbf{U}_0}(h_0)}} \right) \\ &= \frac{\pm 1}{2h_0} \lim_{h\downarrow h_0} \left(\sqrt{\phi'_{\mathbf{U}_0}(h)} + \frac{\phi'_{\mathbf{U}_0}(h)}{\sqrt{\phi'_{\mathbf{U}_0}(h)}} \right) \\ &= \pm \frac{\sqrt{C_{\mathbf{U}_0}(h_0)}}{h_0} \\ &= \lim_{h\uparrow h_0} u'^{\pm}_{\mathbf{U}_0}(h). \end{aligned} \tag{2.5.92}$$

In addition, we may finally prove some facts about $u^{\pm}$.

Proof for 2.30. The derivative has been already calculated, so it remains to argue for strict monotonicity. Once again, we will only concern ourselves with $u^{+}_{\mathbf{U}_0}$. For $C^{-1}_{\mathbf{U}_0}(h) < h \leq h_0$, the result is again imminent, since then

$$u'^{+}_{\mathbf{U}_0}(h) = \frac{\sqrt{C_{\mathbf{U}_0}(h)}}{h} > 0. \tag{2.5.93}$$

For $h > h_0$, we have due to the previous lemma (Lemma 2.31) that

$$\phi_{\mathbf{U}_0}(h) > \phi_{\mathbf{U}_0}(h_0), \tag{2.5.94}$$

as well as

$$\phi'_{\mathbf{U}_0}(h) > 0. \tag{2.5.95}$$

From that, the result

$$u'^{+}_{\mathbf{U}_0}(h) > 0 \tag{2.5.96}$$

is immediate which gives us that $u^{+}_{\mathbf{U}_0}$ is strictly monotonous. □

Since ϕ is invertible, we are able to re-write (2.5.86). We get

$$\begin{aligned} h_{ML} &= \phi^{-1}_{\mathbf{U}_L}(\phi_M) \\ h_{MR} &= \phi^{-1}_{\mathbf{U}_R}(\phi_M) \end{aligned} \tag{2.5.97}$$

Thus, by introducing the functions

$$\varphi^{\pm}_{\mathbf{U}_0} = (u^{\pm}_{\mathbf{U}_0} \circ \phi^{-1}_{\mathbf{U}_0}), \tag{2.5.98}$$

we get that (2.5.86) is equivalent to

$$\Phi_{[\mathbf{U}]}(\phi_M) := \varphi^{+}_{\mathbf{U}_R}(\phi_M) - \varphi^{-}_{\mathbf{U}_L}(\phi_M) = 0 \tag{2.5.99}$$

This is the equivalent to the shallow water function $\Phi_{i-1/2}$ from [20, Appendix B]. Thus, in case we can show that there is a solution to $\Phi(\phi_M) = 0$, we are done. As soon as we obtain ϕ_M, we can immediately compute u_M, and all of $\mathbf{U}_{ML}$ and $\mathbf{U}_{MR}$.

But first, we need to clarify where these functions are defined at all. We have that $\varphi^{\pm}_{\mathbf{U}_0}(\phi)$ is defined, given $\mathbf{U}$, for

$$\phi \geq \phi_{\mathbf{U}_0}(C^{-1}_{\mathbf{U}_0}(0)). \tag{2.5.100}$$

Subsequently, $\Phi_{[\mathbf{U}]}(\phi)$ is defined for

$$\phi_{\min} := \max\left\{\phi_{\mathbf{U}_R}(C^{-1}_{\mathbf{U}_R}(0)), \phi_{\mathbf{U}_L}(C^{-1}_{\mathbf{U}_L}(0))\right\} \leq \phi. \tag{2.5.101}$$

To prove that ϕ_M exists in most of the cases, we need a small helper Lemma.

Lemma 2.32. *Suppose $g > 0$, and $C_R, C_L, h_R, h_L > 0$. Then, the following statements hold:*

- $\varphi^{+}_{\mathbf{U}_R}$ *is strictly monotonically increasing*
- $\varphi^{-}_{\mathbf{U}_L}$ *is strictly monotonically decreasing*

As a result, $\Phi_{[\mathbf{U}]}$ is strictly monotonous function as well, and $\Phi_{[\mathbf{U}]}(\phi) \to \infty$ for $\phi \to \infty$.

Proof. The two statements follow from the fact that the concatenation of two strictly monotonous functions, as shown by the Lemmas 2.30 and 2.31, is again a strictly monotonous function, and that inverse functions of strictly monotonous functions are also strictly monotonous as well. As a result, $\Phi_{[\mathbf{U}]}$ is strictly monotonically increasing as well, since it is the difference between a strictly monotonically increasing and a strictly monotonically decreasing function. □

Finally, we are able to prove the following existence and uniqueness statement.

Theorem 2.33. *Suppose $g > 0$, and $C_R, C_L, h_R, h_L > 0$. There is a solution ϕ_M to $\Phi(\phi_M) = 0$, if and only if*

$$\Phi_{[\mathbf{U}]}(\phi_{min}) \leq 0 \tag{2.5.102}$$

Furthermore, if the solution exists, it is unique.

Proof. By the previous Lemma, Φ is strictly increasing. Thus, if $\Phi_{[\mathbf{U}]}(\phi_{\min}) \leq 0$, it means that since $\Phi_{[\mathbf{U}]}(\phi) \to \infty$ for $\phi \to \infty$ that by the Intermediate Value Theorem, there is a ϕ_M which solves $\Phi_{[\mathbf{U}]}(\phi_M) = 0$. The strict monotonicity of $\Phi_{[\mathbf{U}]}$ gives us the uniqueness. On the other hand, if $\Phi_{[\mathbf{U}]}(\phi_{\min}) > 0$, then we get for all $\phi \in [\phi_{\min}, \infty)$ that $\Phi_{[\mathbf{U}]}(\phi) \geq \Phi_{[\mathbf{U}]}(\phi_{\min}) > 0$, so there is no solution. □

In addition, we obtain a small result which allows us to determine before we solve $\Phi(\phi) = 0$, if we will get a shock or a rarefaction in each of the genuinely nonlinear fields. Subsequently, this generalizes [20, eqs. (B.3)-(B.5)]. We write $\phi_L = \phi_{\mathbf{U}_L}(h_L)$ and $\phi_R = \phi_{\mathbf{U}_R}(h_R)$.

Lemma 2.34. *The following statement holds, given that h_{ML}, h_{MR}, ϕ_M are the solutions from $\Phi_{[\mathbf{U}]}(\phi_M) = 0$. Suppose $h_L, h_R > 0$, we get:*

- *If $\phi_{min} \leq \phi_L$, then $\Phi_{[\mathbf{U}]}(\phi_L) < 0 \Leftrightarrow h_L < h_{ML}$*
- *If $\phi_{min} \leq \phi_R$, then $\Phi_{[\mathbf{U}]}(\phi_R) < 0 \Leftrightarrow h_R < h_{MR}$*

Proof. We will only concern ourselves with the first equation, since the second equation can be proven in a similar fashion. To prove this, we combine Lemma 2.32 with Lemma 2.31. Due to $\Phi_{[\mathbf{U}]}$ being strictly increasing, we obtain from $\Phi_{[\mathbf{U}]}(\phi_L) < 0$ that $\phi_L < \phi_M$, since $\Phi_{[\mathbf{U}]}(\phi_M) = 0$. Next, we apply $\phi_{\mathbf{U}_L}^{-1}$ onto both sides which yields us $h_L < h_{ML}$, since $\phi_{\mathbf{U}_L}^{-1}$ is strictly increasing as well. All steps are possible in the other direction as well. □

Thus, we directly obtain the following result.

Corollary 2.35. *Suppose $h_L, h_R > 0$ and h_{ML}, h_{MR}, ϕ_M are the solutions from the equation $\Phi_{[\mathbf{U}]}(\phi_M) = 0$.*

- *If $\phi_L > \phi_{min}$, we get $\Phi_{[\mathbf{U}]}(\phi_L) < 0$, iff we have a shock for r_1.*
- *If $\phi_R > \phi_{min}$, we get $\Phi_{[\mathbf{U}]}(\phi_R) < 0$, iff we have a shock for r_3.*

This helps us during implementation, since we subsequently do not need to solve against functions which have a case distinction.

2.5.4 Special States

In (2.0.10), we have two positivity conditions ($h \geq 0$ and $C \geq 0$), both of which require special attention once we have $h = 0$ or $C = 0$.

The above computation works in the case that we have $\Phi_{[\mathbf{U}]}(\phi_{\min}) \leq 0$. However, we need a slightly different approach, once we consider dry states or states where the hyperbolicity is lost which equals $C = 0$. See e.g. [24] for the shallow water equations in this situation. In the case of the shallow water equations, the dry states and the states where hyperbolicity is lost conflate: we get $C = 0 \Leftrightarrow h = 0$.

We will (for now) only handle the dry states.

Dry States

As we have seen during the computation of the rarefaction solutions, p and u cannot be given a meaningful value, if $h = 0$.

Border Dry State Suppose $h_L = 0$ and $h_R = 0$. Then, it makes sense to just assume that the whole solution is constant 0.

Now suppose $h_L = 0$ and $h_R > 0$. Then, we may take $h_{ML} = 0$ (similarly to [24] with the shallow water equations) again, but h_{MR} does is not zero anymore, as soon as $c > 0$. So, we connect h_R and h_{MR} as normal, and we connect h_{ML} and h_{MR} as normal. These choices give us values u_M and p_{ML} for the left-middle state. In addition, the values of u and p can be varied freely between the left and middle-left state, since $h_L = h_{ML} = 0$. This behavior is equivalent to choosing $\phi_M = 0$, so we obtain

$$h_{MR} = \phi^{-1}_{\mathbf{U}_R}(0). \tag{2.5.103}$$

The case $h_L > 0$ and $h_R = 0$ is similar, in that we get $h_{MR} = 0$, and

$$h_{ML} = \phi^{-1}_{\mathbf{U}_L}(0). \tag{2.5.104}$$

Central Dry States It turns out that once $c > 0$, solutions which have $h_{ML} = 0$ or $h_{MR} = 0$ cannot exist. This is because we then have for $h \to 0$ that $p_{\mathbf{U}_0}(h) \to -\infty$, and thus $C_{\mathbf{U}_0}(h) \to -\infty$. As a result, $h = C^{-1}_{\mathbf{U}_0}(0)$ is the minimum water height we may reach.

If we have $c = 0$, then $C_{\mathbf{U}_0}(h) = 0$ conflates with $h = 0$, thus we can get dry states again. Yet, unlike the shallow water equations, it may still be that only one of the central states is 0, while the other state may have a non-zero water height, but negative non-hydrostatic pressure instead. This is the case, if we have $p_L, p_R \neq 0$, but $c = 0$.

States where $C = 0$

Note that (2.0.10) loses its hyperbolicity, if $C = 0$. The condition $C = 0$ can be seen as "pressureless". We note that $C = 0$ does not automatically imply $h = 0$, once $c > 0$. Rather, it means that

$$h = -\frac{p + c}{g}. \tag{2.5.105}$$

Of course, we also need $h \geq 0$, and thus we obtain

$$p \leq -c. \tag{2.5.106}$$

And so, we have only $h = 0$ and $C = 0$, if $p = -c$.

We will not present a definite solution here, but instead just give some general observations for $c > 0$.

- If we get $C = 0$ for a border state (e.g. $C_{\mathbf{U}_L}(h_L) = 0$), we can only have a shock next to it, since otherwise we would get $C_{ML} < 0$ which violates the imposed positivity conditions.

- If we get $C = 0$ for a central state, e.g. $C_{ML} = 0$, we have a rarefaction, or $C_L = 0$ as well. This situation can for example occur, if $|u_R - u_L|$ is very large. This is one of the cases where $\Phi_{[\mathbf{U}]}(\phi_{\min}) > 0$, i.e. Theorem 2.33 does not give us a solution.

2.5.5 Computing a Solution

Finally, we give an algorithm for the solving Riemann problem without bathymetry, and given that Theorem 2.33 is fulfilled or we have a dry state. The result can be found in Algorithm 2.1. Effectively, it takes more space to describe the behavior in the wet-dry case. If $h_L, h_R > 0$, we just apply Theorem 2.33 and compute the output values, since we for now ignore the $C = 0$ case.

After that, we have found h_{ML} and h_{MR}, and subsequently know if we have a rarefaction wave or a shock for the genuinely nonlinear fields. In the following, we sketch shortly on how to compute the solution for a given $\xi = x/t$. The result can be found in Algorithm 2.2.

To do so, we need to define the rarefaction functions by

$$\rho_+(h) = u^+_{\mathbf{U}_R}(h) + \sqrt{gh + p_{\mathbf{U}_R}(h) + c^2}, \tag{2.5.107}$$

$$\rho_-(h) = u^-_{\mathbf{U}_L}(h) - \sqrt{gh + p_{\mathbf{U}_L}(h) + c^2}. \tag{2.5.108}$$

Here $u^\pm$ and p are the functions as we had them before, and ρ_+ is defined for $h > C^{-1}_{\mathbf{U}_R}(0)$, and ρ_- for $h > C^{-1}_{\mathbf{U}_L}(0)$, respectively. For $h \leq h_L$, we have that ρ_- is strictly monotonous, thus we can invert it. The same is true for $h \leq h_R$ and ρ_+. Thus, we have that the equation which describes the path of the rarefaction

$$\xi = \rho_-(h) \tag{2.5.109}$$

has a unique solution for $h_L \leq h > C^{-1}_{\mathbf{U}_L}(0)$, and

$$\xi = \rho_+(h) \tag{2.5.110}$$

has a unique solution for $h_R \leq h > C^{-1}_{\mathbf{U}_R}(0)$.

For a given ξ, we may then proceed as follows: we check if $\xi < u_M$ or $\xi > u_M$. Thus shows us which genuinely nonlinear field we need to evaluate. For $\xi < u_M$, we check r_1, and for $\xi > u_M$, we check r_3. (in effect, we may entirely ignore the values of the other side: if $\xi < u_M$, then we only need h_L and h_{ML}, but no h_{MR} or h_R) The next step depends on if we have a shock or a centered rarefaction.

- If we have a shock, we compute $s^\pm_{\mathbf{U}_0}$ depending on the sign we chose, and compare ξ against it.

- If we have a centered rarefaction, we work with ρ_+ or ρ_-. We first check ρ_+ and ρ_- against the border states, e.g. if we have $\xi < u_M$. Therefore, we check if $\xi \in (\rho_-(h_{ML}), \rho_-(h_L))$. If this is the case, we solve $\xi = \rho_-(h)$ for h. If $\xi \leq \rho_-(h_{ML})$, we output h_{ML}, and if $\xi \geq \rho_-(h_L)$, we output h_L.

For v and w, we simply decide by $\xi < u_M$ or $\xi > u_M$. If $\xi < u_M$, we return v_L and w_L, and if $\xi > u_M$, we return v_R and w_R.

This procedure also works for dry states, since then we have $h_L = h_{ML} = 0$, or $h_R = h_{MR} = 0$, so we cannot end up in the rarefaction curve, nor in the shock.

Data: Constants $c \geq 0$ and $g > 0$, as well as Ψ fulfilling Definition 2.7 and Assumptions 2.8 and 2.23.
Input: Left-hand and right-hand states $\mathbf{U}_L$, $\mathbf{U}_R$
Output: Central states $\mathbf{U}_{ML}$ and $\mathbf{U}_{MR}$.
if $h_L = h_R = 0$ **then**
  Set $h_{ML} = h_{MR} = 0$, $u_{ML} = u_{MR} = 0$, $p_{ML} = p_{MR} = 0$;
else if $h_L = 0$ **then**
  Set $h_{ML} = 0$, $p_{ML} = 0$;
  Compute $h_{MR} = \phi^{-1}_{\mathbf{U}_R}(0)$;
  Compute $u_{ML} = u_{MR}$ and p_{MR} from h_{MR};
else if $h_R = 0$ **then**
  Set $h_{MR} = 0$, $p_{MR} = 0$;
  Compute $h_{ML} = \phi^{-1}_{\mathbf{U}_L}(0)$;
  Compute $u_{ML} = u_{MR}$ and p_{ML} from h_{ML};
else
  Solve $\Phi_{[\mathbf{U}]}(\phi_M) = 0$ for ϕ_M;
  Compute $h_{ML} = \phi^{-1}_{\mathbf{U}_L}(\phi_M)$, and $h_{MR} = \phi^{-1}_{\mathbf{U}_R}(\phi_M)$;
  Compute $u_{ML} = u_{MR}$, and p_{ML}, p_{MR} from h_{ML} and h_{MR};
end
return $\mathbf{U}_{ML}$ *and* $\mathbf{U}_{MR}$

Algorithm 2.1: An algorithm for computing the intermediate states for the Riemann Problem (2.5.1) without bathymetry, and ignoring the $C = 0$ case.

2.5.6 Evaluation

Finally, we will show some plots which show that our Riemann solution behaves similarly to known Euler and shallow water Riemann problem solutions (cf. especially [24]). We programmed a Python script which reproduces the Riemann solution for the case which Corollary 2.21 covers, as well as the dry case. This essentially boiled down to implementing both Algorithms 2.1 and 2.2. The rarefaction integral for $u^{\pm}_{\mathbf{U}_0}$ is discretized using a 17-th order Gauss Legendre quadrature rule, to provide an exact-as-possible result. All equations which we did not resolve into a closed-form representation are solved numerically, using the scipy [45] `fsolve` method. This means, we use if for computing $\phi^{-1}_{\mathbf{U}_0}$ and solving $\Phi_{[\mathbf{U}]}(\phi_M) = 0$. In all cases, unless mentioned otherwise, we will use Ψ_C as shock parametrization, i.e. p_Ψ uses Ψ_C.

To begin with, we shall mention that we compared the values of the solution for $c = 0$ and $p_L = p_R = 0$ with the shallow water solver provided by [24], and we got the same

Data: Constants $c \geq 0$ and $g > 0$, as well as Ψ fulfilling Definition 2.7 and Assumptions 2.8 and 2.23.
Data: Intermediate states $\mathbf{U}_L$, $\mathbf{U}_{ML}$, $\mathbf{U}_{MR}$ $\mathbf{U}_R$.
Input: $\xi \in \mathbb{R}$
Output: The value $\mathbf{U}$ of Riemann Problem solution at position $\xi = x/t$.
if $\xi < u_M$ **then**
  if $h_L < h_{ML}$ **then**
    if $\xi < s^-_{\mathbf{U}_L}(h_{ML})$ **then**
      return $\mathbf{U}_L$
    else
      return $\mathbf{U}_{ML}$
    end
  else
    if $\xi \leq \rho_-(h_{ML})$ **then**
      return $\mathbf{U}_{ML}$
    else if $\xi \geq \rho_-(h_L)$ **then**
      return $\mathbf{U}_L$
    else
      Solve $\rho(h) = \xi$ for h^*;
      return $(h^*, u^-(h^*), p(h^*))^T$
    end
  end
else
  if $h_R < h_{MR}$ **then**
    if $\xi < s^+_{\mathbf{U}_R}(h_{MR})$ **then**
      return $\mathbf{U}_{MR}$
    else
      return $\mathbf{U}_R$
    end
  else
    if $\xi \geq \rho_+(h_{MR})$ **then**
      return $\mathbf{U}_{MR}$
    else if $\xi \leq \rho_+(h_R)$ **then**
      return $\mathbf{U}_R$
    else
      Solve $\rho_+ = \xi$ for h^*;
      return $(h^*, u^+(h^*), p(h^*))^T$
    end
  end
end

Algorithm 2.2: An algorithm for computing the solution to the Riemann Problem (2.5.1) at a given position $\xi = x/t$, and given the intermediate states computed using Algorithm 2.1.

values out of it as our solver returned them.

Firstly, we show the solution, for $g = 9.81$ and $c = 4$, for five problems. See Figure 2.4. The data for the five problems can be found in Table 2.1. One can clearly see the three fields in the dam-break problem: on the left, we have a rarefaction, then the central contact discontinuity, and finally a shock on the right-hand side. Especially interesting here is the non-hydrostatic pressure: after it sinks to a negative value, it jumps to a positive value during the contact discontinuity. For the symmetric expansion, we see that u is connected from both sides to a central state via a rarefaction. Note that we could only take $||u_R| - |u_L||$ to be small, otherwise we would get the $C = 0$ problem. The contact discontinuity has no effect in this case. The values of -2 and 2 lead to us close to the $C = 0$ state. For the collision, we may take arbitrarily large differences between u_R and u_L. The behavior here is just as in the shallow water case, i.e. we have two shocks. But also, the non-hydrostatic pressure increases. Once again, since the problem is completely symmetric, the contact discontinuity has no effect. The pressure exchange shows a qualitative behavior exactly like in the dam break, except that h and p are switched. The pressure exchange is an example which shows the difference to systems which do not depend on non-hydrostatic pressure, since h and u have the same boundary values on both sides—so we arrive at a constant solution for $c \to 0$. Finally, we get to the wet-dry front. Here, we can see the special case that we discussed: h starts at 0 (and u and p have no real meaning here, we could in fact just set them to any value and get the same result)

In addition to Figure 2.4, we also provide the computed values in Table 2.2. For completeness, we provide the values of $h_{ML}, h_{MR}, u_{ML} = u_M = u_{MR}, p_{ML}, p_{MR}$. In Table 2.3, we show the values of $\phi_L, \phi_{ML}, \phi_{MR}$, and ϕ_R. This also confirms that $\phi_{ML} = \phi_{MR}$ holds true.

Secondly, we compare the solution of the dam break problem for different values of c, to show the influence of c on the solution. Figure 2.5 shows the dam break problem for $c = 0, 2, 4$. In particular, $c = 0$ equals the shallow water dam break solution. The plot shows that increasing c penalizes changes in the non-hydrostatic pressure component stronger. This can be best seen for h: the larger c grows, the larger is the exchange between hydrostatic and non-hydrostatic pressure at the central contact discontinuity. Subsequently, h is changed less and less during the genuinely nonlinear fields—since the increase in c penalizes a non-hydrostatic pressure change during the genuinely nonlinear fields more. Unsurprisingly, p varies still stronger the larger c is, and u becomes smaller with growing c. This is all due to the penalization as well. The solution also expands quicker, the larger c is which corresponds well to the increasing eigenvalue.

For all of these problems, it does essentially not matter. For both Ψ_P and Ψ_C, the results are almost the same, since we only deal with little amounts of non-hydrostatic pressure. However, this changes once we have larger numbers, the different grows rather significantly, as seen in Figure 2.6 for an extreme example. As it is expected, since Ψ_P lets the non-hydrostatic pressure grow linearly and Ψ_C lets it grow logarithmically, we get that Ψ_P causes a significantly higher non-hydrostatic pressure in the shock, and as a result, a lower increase in water height. In addition to that, the shock wave propagates

	Name	h_L	h_R	u_L	u_R	p_L	p_R
(a)	Dam break	4	1	0	0	0	0
(b)	Symmetric expansion	1	1	−2	2	0	0
(c)	Symmetric collision	1	1	1	−1	0	0
(d)	Pressure exchange	1	1	0	0	10	0
(e)	Wet-Dry state	0	1	0	0	0	0

Table 2.1: The sample Riemann problems we consider, with their initial data.

ID	h_{ML}	h_{MR}	u_M	p_{ML}	p_{MR}
(a)	2.78885177	1.53431854	2.44393171	−5.77063045	6.84938135
(b)	0.64116568	0.64116568	−0.00000000	−7.11147814	−7.11147814
(c)	1.20620312	1.20620312	0.00000000	2.99964013	2.99964013
(d)	0.85737710	1.18376253	0.89530099	7.53796115	2.69916721
(e)	0.00000000	0.78589889	−1.14934695	0.00000000	−3.85483407

Table 2.2: The intermediate state locations for the problems mentioned in Table 2.1. For all problems, we set $c = 4$ and $g = 9.81$.

quicker.

2.6 Reformulations of the Equation System

We close this chapter by looking at some potential re-formulations of the equation system. Naturally, the weak solutions are not preserved by these transformations. However, they get very close to it.

ID	ϕ_L	ϕ_{ML}	ϕ_{MR}	ϕ_R
(a)	78.48000000	22.05615704	22.05615704	4.90500000
(b)	4.90500000	−2.54322245	−2.54322245	4.90500000
(c)	4.90500000	10.75458715	10.75458715	4.90500000
(d)	14.90500000	10.06851874	10.06851874	4.90500000
(e)	0.00000000	0.00000000	0.00000000	4.90500000

Table 2.3: The values of $\phi = hp_T$ during the Riemann solutions for the problems shown in Table 2.1. For all problems, we set $c = 4$ and $g = 9.81$.

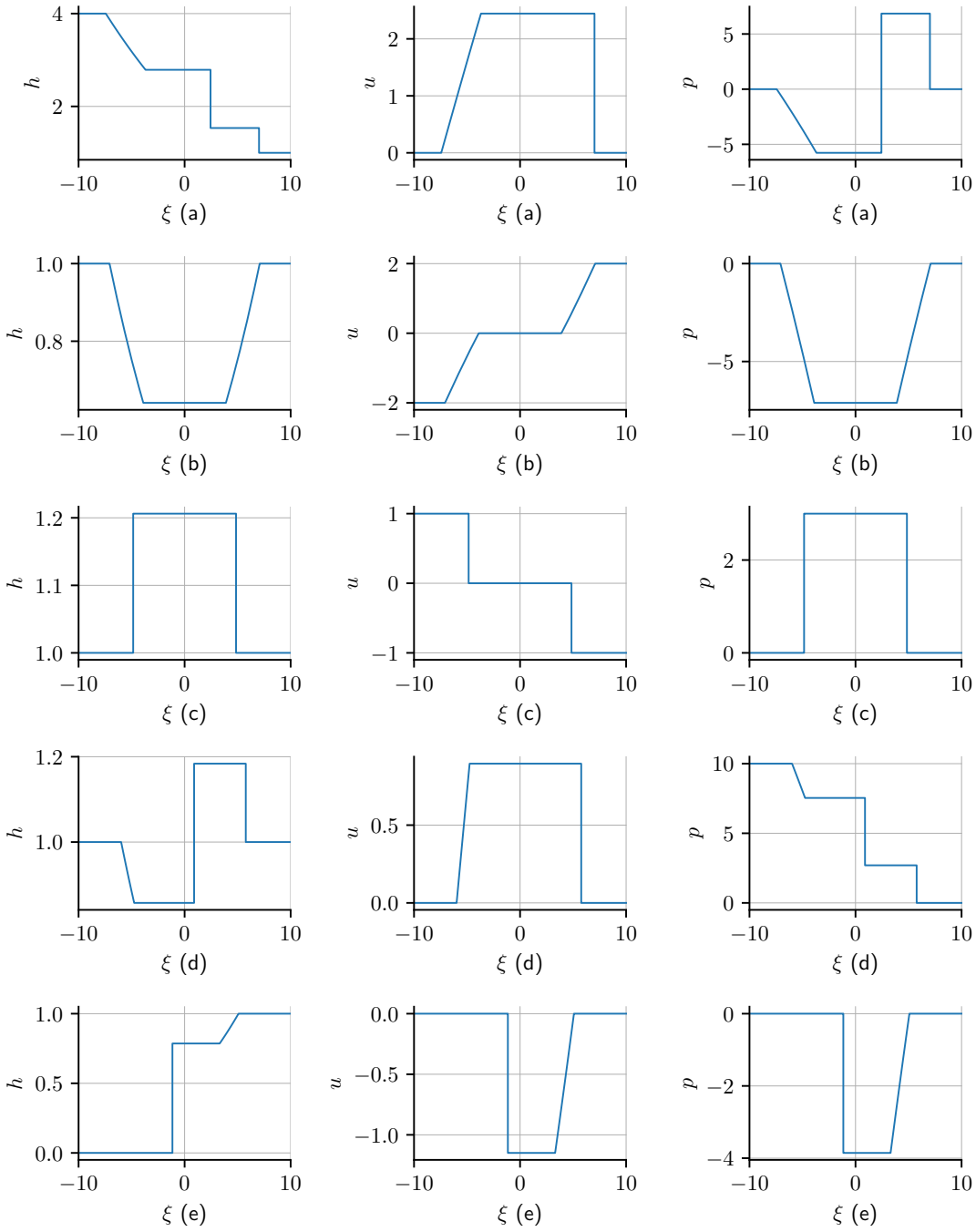


Figure 2.4: Solving Riemann problems for (2.0.10) with initial data from Table 2.1 (a)-(e). The problems are ordered by row, their number is written next to ξ. For all problems, we set $c = 4$ and $g = 9.81$. The values for u and p in (e) could be chosen arbitrarily on the left, so we set them to 0.

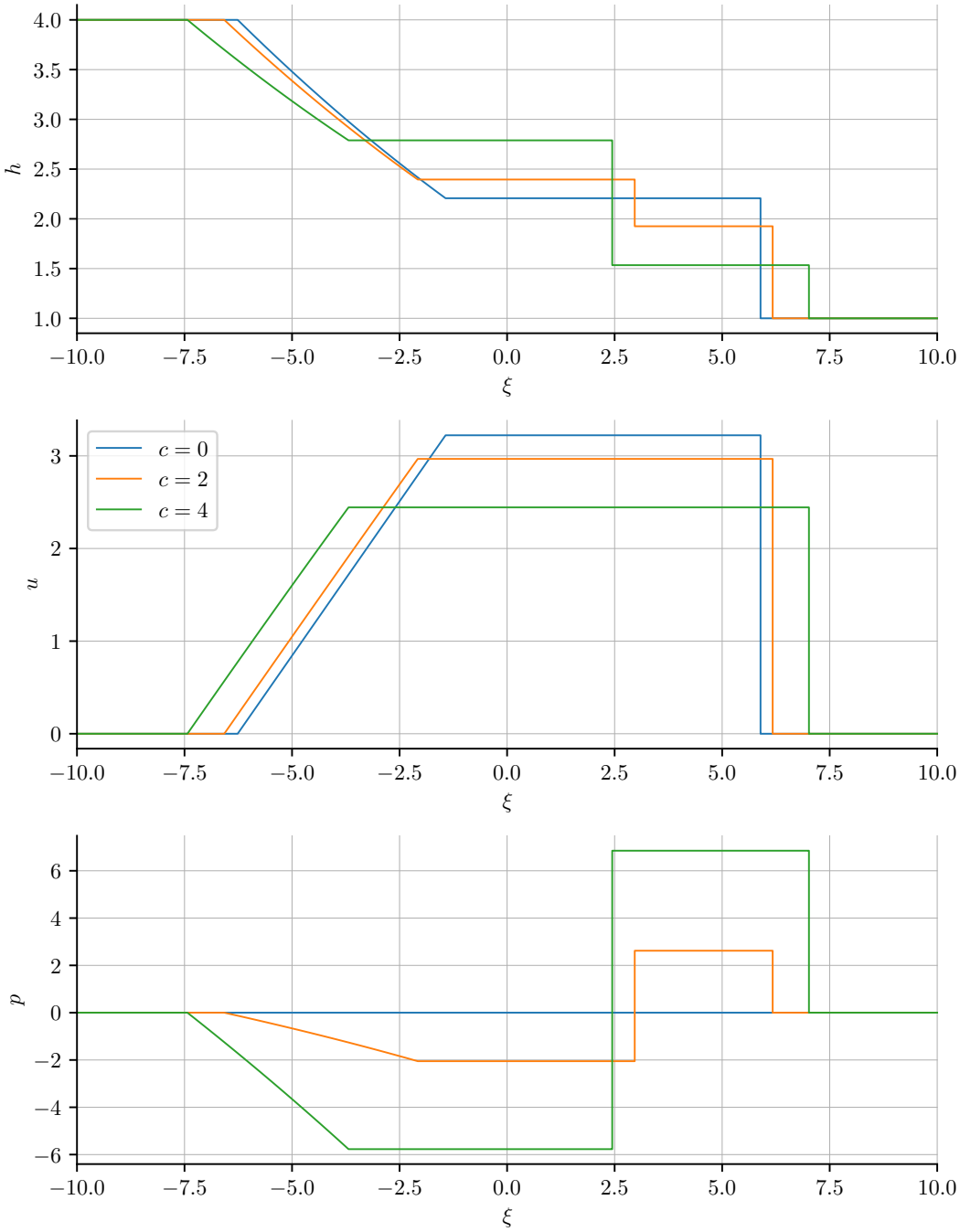


Figure 2.5: The dam break (see Table 2.1 (a)), for different values of $c = 0, 2, 4$. The case $c = 0$ corresponds to the situation with the shallow water equations (2.1.1).

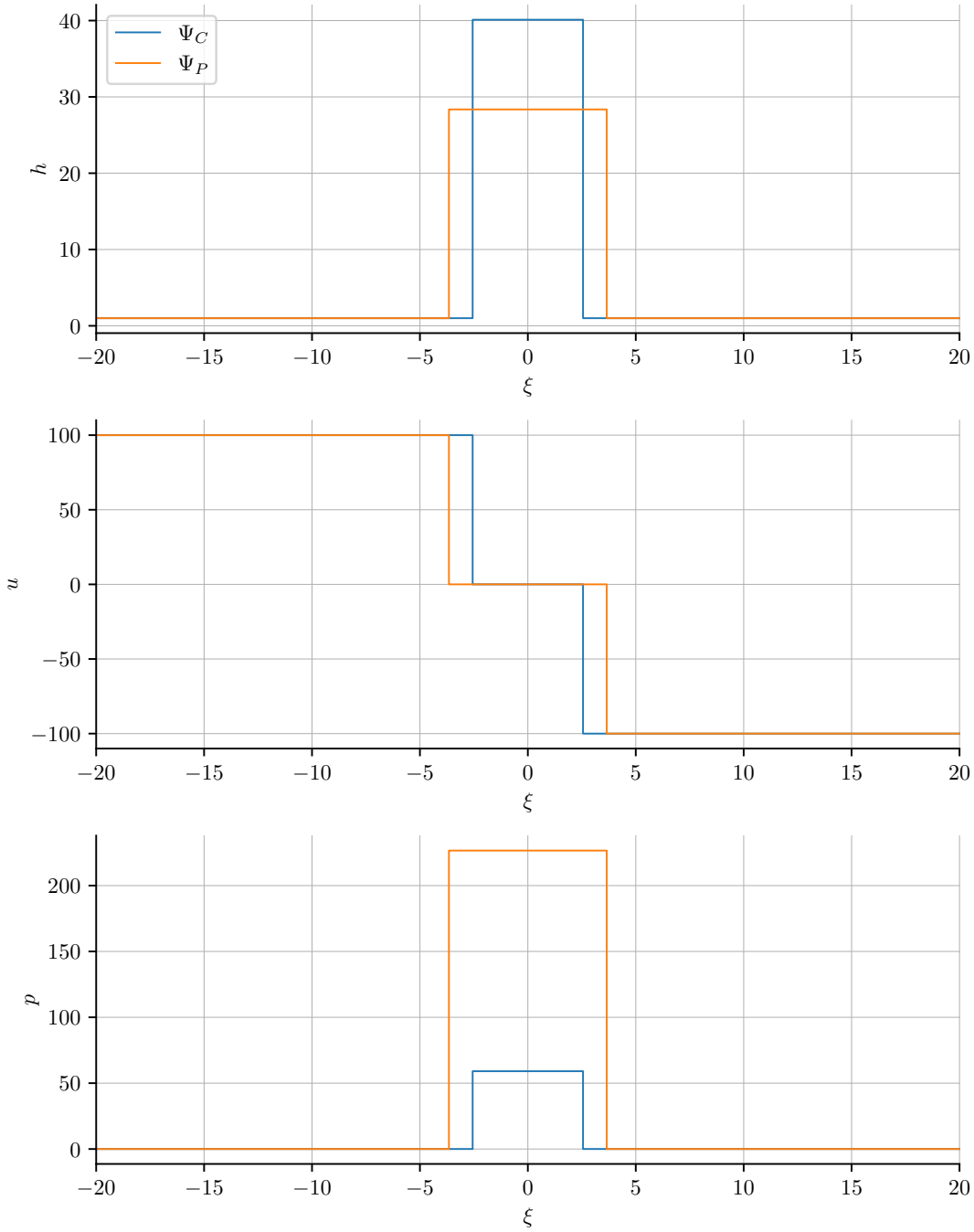


Figure 2.6: The symmetric collision with $u_L = 100$ and $u_R = -100$ (see Table 2.1 (c) for values with $u_L = 1$ and $u_R = -1$) for different families of paths, Ψ_C denotes the linear paths in conservative variables, and Ψ_P denotes linear paths in primitive variables.

2.6.1 Formulation in Primitive Variables

In order to re-write (2.0.10) in primitive variables, we just note that we can re-write for any variable v to

$$\partial_t(hv) = \partial_t hv + \partial_t vh = -\nabla \cdot (h\mathbf{u})v + \partial_t vh. \tag{2.6.1}$$

And also, we have

$$\nabla \cdot (h\mathbf{u}v) = \nabla \cdot (h\mathbf{u})v + h(\mathbf{u} \cdot \nabla)v. \tag{2.6.2}$$

Combining these two equations, we obtain

$$\partial_t(hv) + \nabla \cdot (h\mathbf{u}v) = h(\partial_t v + (\mathbf{u} \cdot \nabla)v). \tag{2.6.3}$$

Inserting this into (2.0.10) and dividing the resulting equations for $h\mathbf{u}$, hw, and hp by h, we finally get the system

$$\begin{aligned}
\partial_t h + \nabla \cdot (h\mathbf{u}) &= 0, \\
\partial_t \mathbf{u} + (\mathbf{u} \cdot \nabla)\mathbf{u} + g\nabla h + \nabla p + \frac{p\nabla h}{h} &= -\left(g + \gamma\frac{p}{h}\right)\nabla b - \frac{1}{h}\tau_b(h, \mathbf{u}), \\
\partial_t w + (\mathbf{u} \cdot \nabla)w &= \gamma\frac{p}{h} + \frac{1}{h}R_b(\mathbf{U}, \nabla\mathbf{U}), \\
\partial_t p + (\mathbf{u} \cdot \nabla)p + c^2(\nabla \cdot \mathbf{u}) &= \frac{1}{h}\left(2c^2(\mathbf{u} \cdot \nabla)b - 2c^2 w\right).
\end{aligned} \tag{2.6.4}$$

For the Green-Naghdi equations, i.e. $\gamma = 3/2$, and $c \to \infty$, this formulation can also be found in [25, eq. (1.45), (1.47)].

Unfortunately, the primitive formulation does not give us a division-free scheme, as it is the case for the shallow water equations. The idea would have been to potentially achieve a quadrature-free formulation which only uses tensor multiplication. However, since (2.6.4) contains a division operation, this is not possible.

2.6.2 Conservative Re-Formulation

Yet, we may also re-formulate the system slightly to remove the $(\mathbf{u} \cdot \nabla)h$ term, without changing the structure of the equation system too much. To begin with, we do this by ignoring bathymetry once again. We assume $h > 0$. For this, we re-formulate the equation to

$$\begin{aligned}
&(\mathbf{u} \cdot \nabla)h \\
&= \frac{h\mathbf{u}}{h} \cdot \nabla h \\
&= (h\mathbf{u}) \cdot \nabla(\log h) \\
&= \nabla \cdot (h\mathbf{u}\log h) - (\log h)\nabla \cdot (h\mathbf{u}).
\end{aligned} \tag{2.6.5}$$

Next, we insert the h-equation and use $\partial_x(x\log x - x) = \log x$ and get

$$\begin{aligned}
&\nabla \cdot (h\mathbf{u}\log h) - (\log h)\nabla \cdot (h\mathbf{u}) \\
&= \nabla \cdot (h\mathbf{u}\log h) + (\log h)\partial_t h \\
&= \nabla \cdot (h\mathbf{u}\log h) + \partial_t(h\log h - h).
\end{aligned} \tag{2.6.6}$$

This lets us re-write the fourth equation to

$$\partial_t(hp - c^2 h(\log h - 1)) + \nabla \cdot (h\mathbf{u}p + c^2 h\mathbf{u} - c^2 h\mathbf{u}\log h) = 2c^2\mathbf{u}\cdot\nabla b - 2c^2 w. \quad (2.6.7)$$

Introducing

$$\pi = p + c^2(1 - \log h) \quad (2.6.8)$$

gives us the equation

$$\partial_t(h\pi) + \nabla \cdot (h\mathbf{u}\pi) = 2c^2\mathbf{u}\cdot\nabla b - 2c^2 w. \quad (2.6.9)$$

so the fourth equation has been re-written as a conservative equation, given that we ignore the bathymetry. Note that while the above calculation was valid mathematically, to make it valid physically (at least in terms of dimensional analysis, that is), we would need to introduce a reference water height H. For example one could take the same water height as for $c = \alpha\sqrt{gH}$. Then, we write (cf. e.g. [31])

$$\frac{h\mathbf{u}}{h}\cdot\nabla h = \frac{h\mathbf{u}}{\frac{h}{H}}\cdot\nabla\frac{h}{H} = h\mathbf{u}\cdot\nabla\left(\log\frac{h}{H}\right), \quad (2.6.10)$$

and h/H is dimension-less, hence this term is well-defined in the dimensional sense.

We keep the Galilean properties by doing this re-write. In addition, we remove p entirely from the equation system, by re-writing the second and third equation in terms of π as well. This gives us

$$\begin{aligned}
\partial_t h + \nabla\cdot(h\mathbf{u}) &= 0,\\
\partial_t(h\mathbf{u}) + \nabla\cdot(h\mathbf{u}\mathbf{u}^T) + \nabla(\frac{g}{2}h^2 + h(\pi - \ell(h))) &= -(gh + \gamma(\pi - \ell(h)))\nabla b,\\
\partial_t(hw) + \nabla\cdot(h\mathbf{u}w) &= \gamma(\pi - c^2(1 - \log h)),\\
\partial_t(h\pi) + \nabla\cdot(h\mathbf{u}\pi) &= 2c^2(\mathbf{u}\cdot\nabla)b - 2c^2 w.
\end{aligned} \quad (2.6.11)$$

For brevity, we wrote $\ell(h) = c^2(1 + \log(h))$.

We do not lose the hyperbolicity by this re-formulation in any way: for $d = 2$, we get

$$\mathbf{A}_1^{\pi,*}(\mathbf{U}^\pi) = \begin{pmatrix} 0 & 1 & 0 & 0 & 0\\ c^2 log(h) + gh - u^2 & 2u & 0 & 0 & 1\\ -uv & v & u & 0 & 0\\ -uw & w & 0 & u & 0\\ -u\pi & \pi & 0 & 0 & u \end{pmatrix}, \quad (2.6.12)$$

which has the eigenvalues

$$\lambda_1^\pi = u - \sqrt{C^\pi}, \lambda_2^\pi = \lambda_v^\pi = \lambda_w^\pi = u, \lambda_3^\pi = u + \sqrt{C^\pi}. \quad (2.6.13)$$

Here, we have C^π defined as

$$C^\pi := c^2\log(h) + gh + \pi. \quad (2.6.14)$$

In particular, we have

$$\begin{aligned} C &= c^2 + gh + p \\ &= p + c^2(1 - \log(h)) + c^2 \log(h) + gh \\ &= \pi + c^2 \log(h) + gh = C^\pi. \end{aligned} \tag{2.6.15}$$

In addition, the system is still rotationally invariant, so Proposition 2.4 can be still applied.

It is natural that this conservative formulation yields different path-dependent weak solutions in general. Compared to (2.0.10), the system (2.6.11) does not depend on the family of paths anymore, as long as $\nabla b = 0$. However, it is comparably close to the original, non-conservative system. This can be also seen when comparing the Rankine-Hugoniot conditions of (2.6.11). We re-insert $p = \pi - c^2(1 - \log h)$, and ignore the equation concerning w. We write again $\hat{u} = u - s$, and $Q = h_R \hat{u}_R = h_L \hat{u}_L$. Then, the Rankine-Hugoniot conditions read

$$\begin{aligned} h_R \hat{u}_R = h_L \hat{u}_L = Q \\ Q\hat{u}_R + \frac{g}{2} h_R^2 + h_R p_R = Q\hat{u}_L + \frac{g}{2} h_L^2 + h_L p_L \\ Q(p_R + c^2 - c^2 \log h_R) = Q(p_L + c^2 - c^2 \log h_L). \end{aligned} \tag{2.6.16}$$

In the last equation, we may cross out Qc^2 on each side. Then, we get

$$Q(p_R - p_L) = Q(\log(h_R) - \log(h_L)). \tag{2.6.17}$$

This is exactly what we also got when we used Ψ_C for the Rankine-Hugoniot conditions (2.4.52) for our equation system (2.0.10).

It should be noted that once again, converting this formulation to primitive variables would still not give us a division-free scheme.

3 Numerical Discretization

Next, we describe the numerical discretization which we implemented. We follow again [15] in that we use the ADER-DG method with integrated finite volume limiter. But in contrast to [15], we implement both methods in a two-dimensional setting with triangles as elements. As a comparison [15], only squares are used, and most tests in [15, Section 5] are seemingly only one-dimensional.

3.1 Geometric Setting

To begin with, we describe the geometric setting, in which we implemented the equations.

Many of the following definitions about the spatial setting can be found in [10, Section 1.2].

3.1.1 Spatial Setting

We work in two space dimensions, so $d = 2$. As a domain, we consider a scaled and moved square $\Omega = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ where $x_{\min} - x_{\max} = y_{\min} - y_{\max}$.

As a reference element, we take the triangle T_{ref} which is spanned by the points $(0,0)^T$, $(1,0)^T$, and $(0,1)^T$. Thus, we have $T_{\text{ref}} = \text{int}(\text{conv}(\{(0,0)^T, (1,0)^T, (0,1)^T\}))$, where $\text{conv}(A)$ denotes the convex hull of the given set A, and $\text{int}(A)$ denotes the interior thereof. Then, we may define an element space

$$\mathcal{S} = \left\{ P_{h,R,b}(T_{\text{ref}}) \;\middle|\; h > 0, b \in \mathbb{R}^2, R \in SO(2) \right\}, \tag{3.1.1}$$

and the transformation $P_{h,R,b}$ is defined as

$$P_{h,R,b}(x) = (hR)x + b. \tag{3.1.2}$$

Note that we defined T_{ref} to be an open set, and in particular, it does not contain its faces. If it is clear in context that a transformation $P_{h,R,b}$ belongs to a triangle $T \in \mathcal{T}$, then we will not explicitly state that $T = P_{h,R,b}(T_{\text{ref}})$.

We will now consider a mesh $\mathcal{T}$ made out of the elements from $\mathcal{S}$ of Ω. That is we have, is also done in e.g. [10], that

$$\Omega = \bigcup_{T \in \mathcal{T}} \text{cl}(T). \tag{3.1.3}$$

Here, $\text{cl}(A)$ denotes the closure of the set A. We also require that $T_1, T_2 \in \mathcal{T}$ with $T_1 \neq T_2$ are non-overlapping, that is $T_1 \cap T_2 = \emptyset$. In short, $\mathcal{T}$ is a partition of Ω. In

addition, we require $\mathcal{T}$ to be a conforming mesh, so if two triangles $T_1 \neq T_2$ share part of an edge, they share the whole edge.

Given a cell $T \in \mathcal{T}$ and h, R, b from $P_{h,R,b}(T_{\text{ref}}) = T$, we define its outer normals $n^i(T)$ for $i = 1, 2, 3$. These denote the normals w.r.t. the faces

$$F_1(T) = P_{h,R,b}(\text{conv}\{(0,0)^T, (1,0)^T\}), \tag{3.1.4}$$
$$F_2(T) = P_{h,R,b}(\text{conv}\{(1,0)^T, (0,1)^T\}), \tag{3.1.5}$$
$$F_3(T) = P_{h,R,b}(\text{conv}\{(0,1)^T, (0,0)^T\}), \tag{3.1.6}$$

The $n^i(T)$ point away from T, and are orthogonal to $F_i(T)$, respectively.

The enumeration of the faces in this order is chosen so that we walk counter-clockwise around the boundary of T, beginning with the point b. If T is clear, then we are going to write $n^i = n^i(T)$ and $F_i = F_i(T)$. Furthermore, we may now compute the length of all $F_i(T)$. We obtain

$$|F_1(T)| = h, \quad |F_2(T)| = \sqrt{2}h, \quad |F_3(T)| = h. \tag{3.1.7}$$

In addition, we define $F_{\text{ref}} = \text{conv}\{(0,0)^T, (1,0)^T\})$ which is F_1 from T_{ref}.

3.1.2 Time Discretization

In addition to the space discretization, we also need to introduce a time discretization. For that, we introduce the reference time interval $I_{\text{ref}} = [0, 1]$. Suppose that we have time steps $t_0, t_1, ...$[1] We have the transformation

$$Q_{t_n,t_{n+1}}(t) = (t_{n+1} - t_n)t + t_n. \tag{3.1.8}$$

3.1.3 Choice of Basis

Before we discuss the steps of the method itself, we choose a basis in space and time. In our particular case, we will just define a basis for the reference space-time element $I_{\text{ref}} \times T_{\text{ref}}$ and then transform from all other elements from it. In addition, we choose the same basis for all variables h, hu, hv, hw, hp, and b. Note that we might get interesting results from e.g. choosing more compatible bases (see e.g. [10]) or other basis functions like splines, however this was out of the scope for this work.

Once again, it suffices to define a basis on the reference space element T_{ref} and the reference space-time element $I_{\text{ref}} \times T_{\text{ref}}$. As a convention, we will always mention the time interval before the space interval. We do the following: in space, we take the monomials in two dimensions. We write N_P for the space and time polynomial degree. Let

$$\mathbb{P}^1_{N_P} = \text{span}\left\{x^i \;\middle|\; 0 \leq i \leq N_P\right\} \tag{3.1.9}$$
$$\mathbb{P}^2_{N_P} = \text{span}\left\{x^i y^j \;\middle|\; 0 \leq i + j \leq N_P\right\}. \tag{3.1.10}$$

[1] We may not know all timestep values in advance.

In space, we choose an orthogonal polynomial basis with respect to the $L^2(T_{\text{ref}})$ scalar product, e.g. by orthogonalizing the monomial basis. We will also need the tensor product between $\mathbb{P}^1_{N_P}$and $\mathbb{P}^2_{N_P}$ which in this case becomes

$$\mathbb{P}^1_{N_P} \otimes \mathbb{P}^2_{N_P} = \operatorname{span}\left\{ t^k x^i y^j \;\middle|\; 0 \le k \le N, 0 \le i+j \le N_P \right\}. \tag{3.1.11}$$

Given these polynomial bases, we define the basis on the reference element. It is given as

$$B_C = \{w_1, ..., w_{b_C}\}, \tag{3.1.12}$$
$$B_P = \{v_1, ..., v_{b_P}\}, \tag{3.1.13}$$

so that $\operatorname{span}(B_C) = \mathbb{P}^2_{N_P}$, and $\operatorname{span}(B_P) = \mathbb{P}^1_{N_P} \otimes \mathbb{P}^2_{N_P}$. We choose the $w_1, ..., w_{b_C}$, so that they are orthonormal with respect to the scalar product on the triangle area. That means

$$\int_{T_{\text{ref}}} w_i(x) w_j(x)\, \mathrm{d}x = \delta_{ij}, \tag{3.1.14}$$

where δ_{ij} is the Kronecker delta. Likewise, we want

$$\int_{I_{\text{ref}} \times T_{\text{ref}}} v_i(t,x) v_j(t,x)\, \mathrm{d}(t,x) = \delta_{ij}. \tag{3.1.15}$$

Both of this is done e.g. by applying the Gram-Schmidt procedure. Given B_C and B_P on the reference element, we define for $T \in \mathcal{T}$ and $P_{h,R,b}$, $Q_{t_n,t_{n+1}}$ the transformed basis functions as

$$w_{i,T}(x) = w_i(P^{-1}_{h,R,b}(x)) \tag{3.1.16}$$
$$v_{i,T}(t,x) = v_i(Q^{-1}_{t_n,t_{n+1}}(t), P^{-1}_{h,R,b}(x)). \tag{3.1.17}$$

We get the element bases $B_C(T) = \{w_{1,T}, ..., w_{b_C,T}\}$ and $B_P(T) = \{v_{1,T}, ..., v_{b_P,T}\}$. In particular, $B_C(T_{\text{ref}}) = B_C$ and $B_P(T_{\text{ref}}) = B_P$.

Transformation to the Reference Element

Suppose now that we are given a cell $T \in \mathcal{T}$ and the time interval $I = [t_n, t_{n+1}]$. Then, we have parameters h, R, b associated with T, so that $T = P_{h,R,b}(T_{\text{ref}})$. Also, $I = Q_{t_n,t_{n+1}}(I_{\text{ref}})$. We write $\Delta t = t_{n+1} - t_n$. Then:

$$\det DQ_{t_n,t_{n+1}} = \Delta t \tag{3.1.18}$$

and since $|\det R| = 1$

$$|\det DP_{h,R,b}| = |\det hR| = h^2 \tag{3.1.19}$$

Also, we need to differentiate $v_T \in B_P$ in both time and space, and get

$$\begin{aligned} \partial_t v_T &= \frac{1}{\Delta t} v_T \\ \nabla v_T = (Dv_T)^T &= \left(\frac{1}{h} Dv_T R^T \right)^T = \frac{1}{h} R \nabla v_T. \end{aligned} \tag{3.1.20}$$

For $w_T \in B_C$, latter equation holds like-wise. In addition, $\partial_t w_T = 0$.

Defining the Function Spaces

Given all these definitions, we may define the function spaces V_P and V_C we will subsequently work with, similarly to [10], where these constructions are used in the space-only setting. We define for the corrector

$$V_C = \left\{ f \in L^2(\Omega) \;\middle|\; f|_T \in [\mathrm{span}(B_C(T))]^6 \; \forall T \in \mathcal{T}_{\mathrm{DG}} \right\}, \tag{3.1.21}$$

and for the predictor

$$V_P = \left\{ f \in L^2([0,\infty) \times \Omega) \;\middle|\; f|_T \in [\mathrm{span}(B_P(T))]^6 \; \forall T \in \mathcal{T}_{\mathrm{DG}} \right\}. \tag{3.1.22}$$

Essentially, the predictor basis is just the corrector basis, but a tensor product with polynomials in time. These definitions mean that on each element $T \in \mathcal{T}$, we have continuous functions, but between elements, this does not need to be the case. But this way, we are able to describe a function on all of Ω. We choose the same basis for all variables, hence we get a vector of six functions.

Furthermore, given a $\mathbf{U} \in V_C$ or V_P, some edge F and an outer normal n, and two adjacent triangles T_+, T_- to F. We want that n points away onto T_+ from F, we may define the boundary projections

$$\mathbf{U}^+(y) = \lim_{x \in T_+, x \to y} \mathbf{U}(x), \tag{3.1.23}$$

$$\mathbf{U}^-(y) = \lim_{x \in T_-, x \to y} \mathbf{U}(x). \tag{3.1.24}$$

In essence, $\mathbf{U}^-$ is the projection of $\mathbf{U}$ on T_- on F, and $\mathbf{U}^+$ is the projection of $\mathbf{U}$ on T_+ onto F. For our case of polynomial basis functions, this amounts to applying continuity.

3.2 Finite Volume Discretization

To begin with, we implement (2.0.10) using a finite volume scheme. [42, 21] That is, we take (2.2.1) and integrate it over some $T \in \mathcal{T}$.

Suppose that $\mathbf{U}$ is some (unknown) function. We get the equation

$$\partial_t \int_T \mathbf{U} \, \mathrm{d}x = \int_T \nabla \cdot \mathbf{F}(\mathbf{U}) \, \mathrm{d}x + \int_T \mathbf{B} \cdot \nabla \mathbf{U} \, \mathrm{d}x + \int_T \mathbf{S}(\mathbf{U}, \nabla \mathbf{U}) \, \mathrm{d}x. \tag{3.2.1}$$

Next, split this into the first two terms which involve $\mathbf{F}$ and $\mathbf{B}$, and the source term $\mathbf{S}$, using Godunov/Lie-Trotter splitting [44, 30]. Thus, we consider the two parts separately, and merge them afterwards.

3.2.1 Finite Volume Part

The following part is inspired partially by [21, Chapter 4]. For the first term, we introduce the general flux term between two cells as $\hat{\mathbf{D}}(\mathbf{U}^-, \mathbf{U}^+) \cdot n$ for the outer normal n,[2]

[2] See [35] for more details and the construction. In contrast to the conservative case, the term $\hat{\mathbf{D}}$ yields us a path-dependent result.

so that we obtain the semi-discrete equation

$$\partial_t \int_T \mathbf{U}^f \, \mathrm{d}x = \int_{\partial T} \hat{\mathbf{D}}(\mathbf{U}^{f,-}, \mathbf{U}^{f,+}) \cdot n \, \mathrm{d}S. \tag{3.2.2}$$

Next, we introduce the average of $\mathbf{U}^f$ over T which we write as $\bar{U}^f$, and it is a (constant) vector. In addition, we define the values from the neighbors over edge F_i of $\mathbf{U}^f$ and write $\bar{U}^{f,i}$ for them (and also use that they are constant in time). Thus, the boundary integral simplifies to

$$\begin{aligned} &\int_{\partial T} \hat{\mathbf{D}}(\mathbf{U}^{f,-}, \mathbf{U}^{f,+}) \cdot n \, \mathrm{d}S \\ &\approx \sum_{i=1}^{3} |F_i| \mathbf{D}(\bar{U}^f, \bar{U}^{f,i}) \cdot n^i \\ &= h \left(\mathbf{D}(\bar{U}^f, \bar{U}^{f,1}) \cdot n^1 + \sqrt{2} \mathbf{D}(\bar{U}^f, \bar{U}^{f,2}) \cdot n^2 + \mathbf{D}(\bar{U}^f, \bar{U}^{f,3}) \cdot n^3 \right). \end{aligned} \tag{3.2.3}$$

Here, $\mathbf{D}$ is the numerical flux. The time derivative of the integral over the whole triangle becomes

$$\partial_t \int_T \mathbf{U} \, \mathrm{d}x = |T| \partial_t \bar{U}. \tag{3.2.4}$$

Next, we discretize this equation in time using the explicit Euler scheme. Then, we obtain

$$\bar{U}^f_{n+1} = \bar{U}^f_n - \frac{2\Delta t}{h} \left(D(\bar{U}^f_n, \bar{U}^{f,1}_n) \cdot n^1 + \sqrt{2} D(\bar{U}^f_n, \bar{U}^{f,2}_n) \cdot n^2 + D(\bar{U}^f_n, \bar{U}^{f,3}_n) \cdot n^3 \right). \tag{3.2.5}$$

3.2.2 Source Term

Now, we deal with the source term. For the scope of this work, we ignore the wave-breaking term $R_b(\mathbf{U}, \nabla \mathbf{U})$ in the finite volume discretization.[3] Then, we can write in a simplified manner

$$\partial_t \mathbf{U} = \mathbf{S}(\mathbf{U}). \tag{3.2.6}$$

This is a first-order ordinary differential equation for each cell, and we look for an implicit Euler discretization for it. Thus, we need to solve

$$U^{n+1,s} = U^{n,s} + \Delta t \mathbf{S}(U^{n+1,s}, 0). \tag{3.2.7}$$

[3] It would be very well possible to use finite differences to discretize it. However, this has not been implemented so far.

for $\mathbf{U}^s_{n+1}$. In addition, $\mathbf{S}$ now does not depend on b in any way. Solving (3.2.7) using a CAS gives us

$$\mathbf{S}_h^{-1}(\mathbf{U}) = \begin{pmatrix} h \\ (2(\Delta t)gn_m^2)^{-1}\text{sign}(hu)\left(-h^{\frac{7}{3}} + h^{\frac{7}{6}}\sqrt{h^{\frac{7}{3}} + \text{sign}(hu)4(\Delta t)gn_m^2 hu}\right) \\ (2(\Delta t)gn_m^2)^{-1}\text{sign}(hv)\left(-h^{\frac{7}{3}} + h^{\frac{7}{6}}\sqrt{h^{\frac{7}{3}} + \text{sign}(hv)4(\Delta t)gn_m^2 hv}\right) \\ (2(\Delta t)^2\gamma c^2 + h^2)^{-1}\left(h((\Delta t)\gamma hp + h^2 w)\right) \\ (2(\Delta t)^2\gamma c^2 + h^2)^{-1}\left(h(2(\Delta t)c^2 hw + h^2 p)\right) \\ b \end{pmatrix}. \tag{3.2.8}$$

In total, we get

$$U^{n+1,s} = \mathbf{S}_h^{-1}(U^{n,s}). \tag{3.2.9}$$

3.2.3 Combining Flux and Source Term

After we discretized both the flux and the source term, we now combine them using Lie-Trotter splitting, also called Godunov splitting [44, 30]. That is, given the matrix $U^n \in \mathbb{R}^{N_T \times 6}$ as input, interpret it as $\bar{U}^f$, solve for $\bar{U}^f_{n+1}$, and subsequently use this as U^s_n to obtain $U^s_{n+1} = U^{n+1}$. Here N_T denotes the number of triangles. In particular, this gives us for each row U^n_i (i.e. each cell) that

$$U_i^{n+1,\text{fv}} = \mathbf{D}(U_i^n, U_i^{1,n}) \cdot n^{i,1} + \sqrt{2}\mathbf{D}(U_i^n, U_i^{2,n}) \cdot n^{i,2} + \mathbf{D}(U_i^n, U_i^{3,n}) \cdot n^{i,3}, \tag{3.2.10}$$

and

$$U_i^{n+1} = \mathbf{S}_h^{-1}\left(U_i^n - \frac{2\Delta t}{h} U_i^{n+1,\text{fv}}\right). \tag{3.2.11}$$

Here, $n^{i,k}$ are the normals of the cell i, and $U_i^{k,n}$ is the neighbor at face k.

It remains to discuss the timestep. As natural for such a splitting scheme, the timestep is bounded by the maximum timesteps of both parts. Yet, since we used an implicit Euler scheme for the source, and all eigenvalues of $D\mathbf{S}$ lie in the stability region of the implicit Euler scheme, or at the boundary thereof. Thus, the scheme for $\mathbf{S}$ is stable. During all of this, we ignore the wave-breaking term R_b. For the fluxes, we need the condition

$$\Delta t < \frac{\sqrt{2}h}{4|\lambda_{\max}(U^{n+1})|}. \tag{3.2.12}$$

3.3 The Numerical Flux

It is left to specify what the numerical flux term $\mathbf{D}(\mathbf{U}^-, \mathbf{U}^+) \cdot n$ is supposed to be. In this work, we use the following scheme which describes many approximate fluxes (see [15] or [6]). For more background on the theory of the path-dependent numerical fluxes, see [35]. In Table 3.1, an overview over the numerical fluxes we consider here can be

Handle	Name	$\Theta(n, \mathbf{U})$	Φ	Ψ
`llfh`	h-preserving Rusanov Flux	$\Theta_{R,h}$	Ψ_P	-
`llfb`	η-preserving Rusanov Flux (default)	$\Theta_{R,\eta}$	Ψ_P	-
`hll`	HLL Flux	$\alpha_0 I + \alpha_1(\mathbf{A}_{1/2} \cdot n)$	Ψ_P	-
`roe`	Roe Flux	$\lvert\mathbf{A}_{1/2} \cdot n\rvert$	Ψ_R	-
`dot`	DOT Flux	$\lvert\mathbf{A}(\mathbf{U}) \cdot n\rvert$	Ψ_P	Ψ_P

Table 3.1: A list of the implemented fluxes, and their names in the implementation. All fluxes are available for the DG as well as the finite volume limiter part. The matrix $\Theta(n, \mathbf{U})$ specifies the correction term which essentially defines the behavior of the flux. Φ is the path that is used to parametrize the non-conservative part, and Ψ is the path which parametrizes $\Theta(n, \mathbf{U})$. If the choice of path does not matter here, we write "-".

found. Given two families of paths Ψ and Φ, we define the numerical flux from $\mathbf{U}_R$ to $\mathbf{U}_L$ as

$$\begin{aligned}&\mathbf{D}(\mathbf{U}_L, \mathbf{U}_R) \cdot n\\ =&\frac{1}{2}\left((\mathbf{F}(\mathbf{U}_L) \cdot n) + (\mathbf{F}(\mathbf{U}_R) \cdot n) + \int_{\Psi[\mathbf{U}_L, \mathbf{U}_R]} (\mathbf{B}(\mathbf{U}) \cdot n)\, \mathrm{d}\mathbf{U}\right)\\ &- \frac{1}{2}\int_{\Phi[\mathbf{U}_L, \mathbf{U}_R]} \Theta(n, \mathbf{U})\, \mathrm{d}\mathbf{U}.\end{aligned} \tag{3.3.1}$$

It can be alternatively written as

$$\begin{aligned}\mathbf{D}(\mathbf{U}_L, \mathbf{U}_R) \cdot n =&(\mathbf{F}(\mathbf{U}_L) \cdot n) + \frac{1}{2}\int_{\Psi[\mathbf{U}_L, \mathbf{U}_R]} (\mathbf{A}(\mathbf{U}) \cdot n)\, \mathrm{d}\mathbf{U}\\ &- \frac{1}{2}\int_{\Phi[\mathbf{U}_L, \mathbf{U}_R]} \Theta(n, \mathbf{U})\, \mathrm{d}\mathbf{U}.\end{aligned} \tag{3.3.2}$$

Using rotational invariance, we are able to write

$$\begin{aligned}&\mathbf{D}(\mathbf{U}_L, \mathbf{U}_R) \cdot n\\ =&\frac{1}{2}T^T\left(\mathbf{F}_1(T\mathbf{U}_L) + \mathbf{F}_1(T\mathbf{U}_R) + \int_{\Psi[\mathbf{U}_L, \mathbf{U}_R]} \mathbf{B}_1(T\mathbf{U})T\, \mathrm{d}\mathbf{U}\right)\\ &- \frac{1}{2}T^T\int_{\Phi[\mathbf{U}_L, \mathbf{U}_R]} T\Theta(n, \mathbf{U})\, \mathrm{d}\mathbf{U}.\end{aligned} \tag{3.3.3}$$

If $\Theta(n, \cdot)$ is rotationally invariant, depends on the exact choice of flux. However, for all numerical fluxes we consider here, it is true: there is $\hat{\Theta}(\mathbf{U})$, so that $T^T\hat{\Theta}(T\mathbf{U})T = \Theta(n, \mathbf{U})$.

3.3.1 Examining the Approximate Flux

Most of the time, we want to compute $\mathbf{D}(\mathbf{U}_L, \mathbf{U}_R) \cdot n$ and $\mathbf{D}(\mathbf{U}_R, \mathbf{U}_L) \cdot (-n)$ at the same time. The latter is computed as

$$\begin{aligned} &\mathbf{D}(\mathbf{U}_R, \mathbf{U}_L) \cdot (-n) \\ =&\frac{1}{2}\left(-(\mathbf{F}(\mathbf{U}_R) \cdot n) - (\mathbf{F}(\mathbf{U}_L) \cdot n) - \int_{\Psi[\mathbf{U}_R, \mathbf{U}_L]} (\mathbf{B}(\mathbf{U}) \cdot n)\, \mathrm{d}\mathbf{U}\right) \\ &+ \frac{1}{2}\int_{\Phi[\mathbf{U}_R, \mathbf{U}_L]} \Theta(n, \mathbf{U})\, \mathrm{d}\mathbf{U}. \end{aligned} \tag{3.3.4}$$

If we now assume for Ψ and Φ that we have

$$\Psi[\mathbf{U}_L, \mathbf{U}_R] = -\Psi[\mathbf{U}_R, \mathbf{U}_L], \tag{3.3.5}$$

we receive

$$\begin{aligned} &\mathbf{D}(\mathbf{U}_R, \mathbf{U}_L) \cdot (-n) \\ =&-\frac{1}{2}\left((\mathbf{F}(\mathbf{U}_R) \cdot n) + (\mathbf{F}(\mathbf{U}_L) \cdot n) - \int_{\Psi[\mathbf{U}_L, \mathbf{U}_R]} (\mathbf{B}(\mathbf{U}) \cdot n)\, \mathrm{d}\mathbf{U}\right) \\ &+ \frac{1}{2}\int_{\Phi[\mathbf{U}_L, \mathbf{U}_R]} \Theta(n, \mathbf{U})\, \mathrm{d}\mathbf{U}. \end{aligned} \tag{3.3.6}$$

Hence, $\mathbf{D}$ can be computed in both directions by simply computing the parts containing $\mathbf{F}$, $\mathbf{B}$, and Θ separately, and then adding them up with different signs—instead of having to compute $\mathbf{D}(\mathbf{U}_L, \mathbf{U}_R) \cdot n$ and $\mathbf{D}(\mathbf{U}_R, \mathbf{U}_L) \cdot (-n)$ separately. With rotational invariance, it can be written as

$$\begin{aligned} &\mathbf{D}(\mathbf{U}_R, \mathbf{U}_L) \cdot (-n) \\ =&-\frac{1}{2}T^T\left(\mathbf{F}_1(T\mathbf{U}_R) + \mathbf{F}_1(T\mathbf{U}_L) - \int_{\Psi[\mathbf{U}_L, \mathbf{U}_R]} T^T\mathbf{B}_1(T\mathbf{U})T\, \mathrm{d}\mathbf{U}\right) \\ &+ \frac{1}{2}T^T\int_{\Phi[\mathbf{U}_L, \mathbf{U}_R]} T\Theta(n, \mathbf{U})\, \mathrm{d}\mathbf{U}. \end{aligned} \tag{3.3.7}$$

3.3.2 Roe Averages

Before we begin discussing the numerical fluxes, we look at Roe averages and their extension to the non-conservative case, as found in [36]. Since the equations naturally become rather long, introduce the jump notation $[x] = x_R - x_L$ for any variable x. We begin by looking at the equation system without bathymetry (cf. [6, eq. (2.8)]), and after

applying the rotational invariance. Thus, we consider $\mathbf{A}_1^*$ again and get

$$\begin{aligned}
[hu] &= [hu] \\
(g\hat{h} - \hat{u}^2)[h] + 2\hat{u}[hu] + [hp] &= [hu^2] + \frac{g}{2}[h^2] + [hp] \\
-\hat{u}\hat{w}[h] + \hat{w}[hu] + \hat{u}[hw] &= [huw] \\
-\hat{u}\hat{v}[h] + \hat{v}[hu] + \hat{u}[hv] &= [huv] \\
-\hat{u}(c^2 + \hat{p})[h] + (c^2 + \hat{p})[hu] + \hat{u}[hp] &= c^2[hu] + [hup] - c^2 \int_{\Phi[\mathbf{U}_L, \mathbf{U}_R]} u \, \mathrm{d}h.
\end{aligned} \tag{3.3.8}$$

We follow [24], i.e. we solve these equations for $\hat{h}$, $\hat{u}$, $\hat{v}$, $\hat{w}$, and $\hat{p}$. If we manage to do so (and we do that), we fulfill [36, Definition 3] immediately (if we remove the rows concerning $\hat{v}$, $\hat{w}$), if we remove the third and fourth row to get a strictly hyperbolic system as usual. In [24], this is done for the shallow water equations using the variation of constants. This means that we want the Roe-averaged variables to hold for all possible values the constants of our system can take. For our case, the constants are g, γ, and c. In addition to that, we need to choose the family of paths Φ.

A Simple Roe Matrix Computation

We start by looking at the second equation, since the first equation does not give us any information. Here, we note that the term depending on p_R and p_L is redundant, and we are left with

$$(g\hat{h} - \hat{u}^2)(h_R - h_L) + 2\hat{u}(h_R u_R - h_L u_L) = h_R u_R^2 - h_L u_L^2 + \frac{g}{2}(h_R^2 - h_L^2). \tag{3.3.9}$$

This however is just the same equation as in the shallow water case, so we will simply take the corresponding Roe averages. So, as they are mentioned in [24], the Roe averages for $\hat{h}$ and $\hat{u}$ read

$$\hat{h} = \frac{h_L + h_R}{2}, \tag{3.3.10}$$

$$\hat{u} = \frac{\sqrt{h_R}u_R + \sqrt{h_L}u_L}{\sqrt{h_R} + \sqrt{h_L}}. \tag{3.3.11}$$

Next, we look at the third equation. Solving it for $\hat{w}$ gives us

$$\hat{w} = \frac{w_R h_R(u_R - \hat{u}) - w_L h_L(u_L - \hat{u})}{h_R(u_R - \hat{u}) - h_L(u_L - \hat{u})}. \tag{3.3.12}$$

We compute

$$h_R(u_R - \hat{u}) = h_R\sqrt{h_L}\frac{u_R - u_L}{\sqrt{h_R} + \sqrt{h_L}} = \sqrt{h_R}\sqrt{h_R h_L}\frac{u_R - u_L}{\sqrt{h_R} + \sqrt{h_L}}, \tag{3.3.13}$$

and

$$h_L(u_L - \hat{u}) = \sqrt{h_R}\sqrt{h_R h_L}\frac{u_L - u_R}{\sqrt{h_R} + \sqrt{h_L}}. \tag{3.3.14}$$

Cancelling out (3.3.13) and (3.3.14) from (3.3.12) gives us

$$\hat{w} = \frac{\sqrt{h_R} w_R + \sqrt{h_L} w_L}{\sqrt{h_R} + \sqrt{h_L}}, \tag{3.3.15}$$

and so in the same way for $\hat{v}$

$$\hat{v} = \frac{\sqrt{h_R} v_R + \sqrt{h_L} v_L}{\sqrt{h_R} + \sqrt{h_L}}. \tag{3.3.16}$$

For $\hat{p}$, we may remove $c^2(h_R u_R - h_L u_L)$ from both sides, and ignore the remaining parts which depend on c^2. Then, we apply the same procedure as before to get

$$\hat{p} = \frac{\sqrt{h_R} p_R + \sqrt{h_L} p_L}{\sqrt{h_R} + \sqrt{h_L}}. \tag{3.3.17}$$

For all of these variables, it holds that if $\mathbf{U}_R \to \mathbf{U}_L$, then $\hat{\mathbf{U}} \to \mathbf{U}_L$ as well.

The Non-Conservative Part

Next, we look at the non-conservative part, that is, all remaining parts which depend on c^2. It reads

$$\int_{\Phi[\mathbf{U}_L, \mathbf{U}_R]} u - \hat{u} \, \mathrm{d}h = 0. \tag{3.3.18}$$

So now, we need to choose a family of paths. Thus, we heuristically choose linear paths for h and b. Then, we may pull out $\Phi_h'[\mathbf{U}_L, \mathbf{U}_R]$ from the integral and obtain the equation

$$\hat{u} = \int_0^1 \Phi_u[\mathbf{U}_L, \mathbf{U}_R](s) \, \mathrm{d}s. \tag{3.3.19}$$

In addition, we need $\Phi_u(0) = u_L$, and $\Phi_u(1) = u_R$. Thus, we may choose a second-degree polynomial for Φ_u. We get the approach

$$\Phi_u(t) = \alpha_0 + \alpha_1 t + \alpha_2 t^2, \tag{3.3.20}$$

and we have the three unknown coefficients $\alpha_0, \alpha_1, \alpha_2$. Then, the third condition becomes

$$\hat{u} = \int_0^1 \Phi_u[\mathbf{U}_L, \mathbf{U}_R](s) \, \mathrm{d}s = \alpha_0 + \frac{1}{2}\alpha_1 + \frac{1}{3}\alpha_2. \tag{3.3.21}$$

So, we can write the three conditions on the coefficients into an equation system and get

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1/2 & 1/3 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \end{pmatrix} = \begin{pmatrix} u_L \\ \hat{u} \\ u_R \end{pmatrix}. \tag{3.3.22}$$

Solving it yields $\alpha_0 = u_L$, $\alpha_1 = 2(3\hat{u} - 2u_L - u_R)$, and $\alpha_2 = 3(u_L + u_R - 2\hat{u})$.

Extension to Bathymetry

Following [36, eq. (4.57) etc.], it suffices to look at the part which depends on bathymetry only. We get the system

$$\begin{aligned}(g\hat{h} + \gamma\hat{p})[b] &= g\int_{\Phi[\mathbf{U}_L,\mathbf{U}_R]} h \,\mathrm{d}b + \gamma\int_{\Phi[\mathbf{U}_L,\mathbf{U}_R]} p \,\mathrm{d}b \\ -2c^2\hat{u}[b] &= -2c^2\int_{\Phi[\mathbf{U}_L,\mathbf{U}_R]} u \,\mathrm{d}b.\end{aligned} \tag{3.3.23}$$

This justifies our previous choice of linear paths for b, since we may pull out the path derivative $\Phi_b'[\mathbf{U}_L, \mathbf{U}_R] = [b]$. This simplifies everything to

$$\begin{aligned}g\hat{h} + \gamma\hat{p} &= g\int_0^1 \Phi_h[\mathbf{U}_L,\mathbf{U}_R](s)\,\mathrm{d}s + \gamma\int_0^1 \Phi_p[\mathbf{U}_L,\mathbf{U}_R](s)\,\mathrm{d}s, \\ -2c^2\hat{u} &= -2c^2\int_0^1 \Phi_u[\mathbf{U}_L,\mathbf{U}_R](s)\,\mathrm{d}s.\end{aligned} \tag{3.3.24}$$

We can even take the same paths for h and u as before. Thus, our equation system reduces to

$$\hat{p} = \int_0^1 \Phi_p[\mathbf{U}_L,\mathbf{U}_R](s)\,\mathrm{d}s. \tag{3.3.25}$$

Thus, we simply take the same path which we use to connect u to connect p as well. In order to get a full family of paths, we may choose any paths we use to connect v and w as we like. Since $\hat{v}$ and $\hat{w}$ are in structure similar to $\hat{u}$, we choose the same paths we used for u here as well. In summary, we may define what we will here call the Roe path family Ψ_R. We define

$$Q[\mathbf{U}_L, \hat{\mathbf{U}}, \mathbf{U}_R](t) = \mathbf{U}_L + 2(3\hat{\mathbf{U}} - 2\mathbf{U}_L - \mathbf{U}_R)t + 3(\mathbf{U}_L + \mathbf{U}_R - 2\hat{\mathbf{U}})t^2. \tag{3.3.26}$$

Then, we can write

$$\Psi_R[\mathbf{U}_L,\mathbf{U}_R] = \begin{pmatrix} P[h_L,h_R] \\ Q[u_L,\hat{u},u_R] \\ Q[v_L,\hat{v},v_R] \\ Q[w_L,\hat{w},w_R] \\ Q[p_L,\hat{p},p_R] \\ P[b_L,b_R] \end{pmatrix}. \tag{3.3.27}$$

This fulfills Corollary 2.11, since Ψ_R is merely composed of polynomials. Hence it fulfills Definition 2.7 and Assumption 2.8 as well for $\Omega = \mathbb{R}^+ \times \mathbb{R}^5$.

The Roe Matrix

Since we could explicitly specify variables $\hat{h}$, $\hat{u}$, $\hat{v}$, $\hat{w}$, and $\hat{p}$, we get a Roe matrix with the same structure as $\mathbf{A}_1$. Using rotational invariance by Proposition 2.1, we set

$$\mathbf{A}_{1/2}[\mathbf{U}_L,\mathbf{U}_R]\cdot n = T^T\mathbf{A}_1(T\hat{\mathbf{U}})T. \tag{3.3.28}$$

Here, $\hat{\mathbf{U}} = (\hat{h}, \hat{h}\hat{u}, \hat{h}\hat{v}, \hat{h}\hat{w}, \hat{h}\hat{p}, \hat{b})^T$. For completeness, we set $\hat{b} = (b_R + b_L)/2$. By construction, $\hat{\mathbf{U}}$ we obtain that computing $\hat{\mathbf{U}}$ from $T\mathbf{U}_L$ and $T\mathbf{U}_R$ gives the same result as computing $T\hat{\mathbf{U}}$ from $\mathbf{U}_L$ and $\mathbf{U}_R$. It is also imminent that $\mathbf{A}_1(\hat{\mathbf{U}})$ posseses four real eigenvalues ($\hat{u}$ three times, as well as 0, and $\hat{u} \pm \sqrt{g\hat{h} + \hat{p} + c^2}$ each one time), unless we have $\hat{u}^2 = \hat{C}$ or $\hat{C} = 0$. In the definition of [36], we therefore only have a Roe matrix, if we ignore the rows concerning v and w and obtain a strictly hyperbolic system. This is the same situation as with the Riemann Problem. If $\mathbf{U}_L$ and $\mathbf{U}_R$ are clear, we will leave them away and just write $\mathbf{A}_{1/2}$.

Another method from which a bathymetry-respecting Roe matrix can be constructed is discussed in [36].

3.3.3 Path-Independent Fluxes

We begin with the case where Θ is constant in (3.3.1). Subsequently, we get

$$\int_{\Phi[\mathbf{U}_L, \mathbf{U}_R]} \Theta \, \mathrm{d}\mathbf{U} = \Theta \int_0^1 \Phi'[\mathbf{U}_L, \mathbf{U}_R](t) \, \mathrm{d}t = \Theta(\mathbf{U}_R - \mathbf{U}_L). \tag{3.3.29}$$

Rusanov Flux

The simplest case when using constant Θ, is to just take as in [15],

$$\Theta = \Theta_{R,h} := \begin{pmatrix} s & 0 & 0 & 0 & 0 & 0 \\ 0 & s & 0 & 0 & 0 & 0 \\ 0 & 0 & s & 0 & 0 & 0 \\ 0 & 0 & 0 & s & 0 & 0 \\ 0 & 0 & 0 & 0 & s & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{3.3.30}$$

The last row is left constant, since b should stay constant throughout the flux. s is defined as

$$s := \max_{\mathbf{U} \in [\mathbf{U}_L, \mathbf{U}_R]} \|\mathbf{A}_1(\mathbf{U})\|_2. \tag{3.3.31}$$

This can be simplified even further by parametrizing $\mathbf{U}$. This means that we use that for $t \in [0, 1]$ we have

$$s = \max_{t \in [0,1]} \|\mathbf{A}_1(\Psi_P(t))\|_2. \tag{3.3.32}$$

Since

$$\|\mathbf{A}_1(\Psi_P(t))\|_2 = |u| + \sqrt{gh + p + c^2} \tag{3.3.33}$$

is a monotone function and Ψ_P is linear and thus convex, we get that we maximum value must lie at $t = 0$ or $t = 1$. So, it remains to check

$$s = \max_{\mathbf{U} \in \{\mathbf{U}_L, \mathbf{U}_R\}} \left(|u| + \sqrt{gh + p + c^2} \right). \tag{3.3.34}$$

We need to make a small modification so that the Resting-Lake property is fulfilled for the Rusanov flux. With $\Theta_{R,h}$, it is not the case, as it can be seen by the following example. Suppose that we have $g > 0$, $\eta_L = \eta_R$, $u_L = u_R = 0$ and $v_L = v_R = 0$, but $b_L < b_R$. Then $h_L > h_R$, i.e. the first component of $\mathbf{D}$ becomes

$$h_L u_L + h_R u_R - s(h_R - h_L) = s(h_L - h_R) > 0, \tag{3.3.35}$$

since we have $h_L > h_R \geq 0$, and thus $s > \sqrt{g h_L} > 0$. Subsequently, we may have a large flux, where no flux should be.

This problem depends entirely on the correction term

$$\Theta(\mathbf{U}_R - \mathbf{U}_L). \tag{3.3.36}$$

Therefore, we modify it: in the first component, we replace h_R by η_R and h_L by η_L. Then, under the same setting as before, we get

$$u_L + u_R - s(\eta_R - \eta_L) = 0 \tag{3.3.37}$$

by construction. This is as if we would replace in (2.2.1) the vector $\mathbf{U}$ by a vector $\tilde{\mathbf{U}}$ which reads

$$\tilde{\mathbf{U}} = \begin{pmatrix} \eta \\ hu \\ hv \\ hw \\ hp \\ b \end{pmatrix} \tag{3.3.38}$$

and change $\mathbf{F}$ and $\mathbf{B}$ to give the same results We have $\partial_t \eta = \partial_t (h + b) = \partial_t h$, since b is constant over time. As a result, we keep the same PDE as we had before. In summary, we get the following Θ:

$$\Theta = \Theta_{R,\eta} := \begin{pmatrix} s & 0 & 0 & 0 & 0 & s \\ 0 & s & 0 & 0 & 0 & 0 \\ 0 & 0 & s & 0 & 0 & 0 \\ 0 & 0 & 0 & s & 0 & 0 \\ 0 & 0 & 0 & 0 & s & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{3.3.39}$$

Note that due to this fix, we lose the preservation of the vacuum property. Thus, it makes sense to employ the modified Rusanov flux for the DG part only due to its simplicity. Secondly, we will always use the limiter, once we get to a dry state.

As in both $\Theta_{R,h}$ and $\Theta_{R,\eta}$ we have a scaled identity matrix in the rows for hu and hv, we get that both matrices yield a rotationally invariant Θ.

Roe Flux

We can also use the Roe matrix which computed in Section 3.3.2 as a flux. The Roe flux reads, as discussed in [36] or [6] as

$$\Theta = \left|\mathbf{A}_{1/2} \cdot n\right|. \tag{3.3.40}$$

Here, the absolute value of a diagonalizable matrix A is defined as

$$|A| = R\,|D|\,R^{-1}, \tag{3.3.41}$$

where $A = RDR^{-1}$ is an eigenvector decomposition of A, and D is a diagonal matrix. $|D|$ is the element-wise absolute value of D. In particular, this does not prevent rotational invariance:

$$\left|\mathbf{A}_{1/2} \cdot n\right| = \left|T^T \mathbf{A}_1(T\hat{\mathbf{U}})T\right| = T^T \left|\mathbf{A}_1(T\hat{\mathbf{U}})\right| T, \tag{3.3.42}$$

using the same argumentation as when proving the hyperbolicity in Proposition 2.4.

For the Roe flux, and only for the Roe flux, we also choose Ψ_R for the path which parametrizes the integral over $\mathbf{B}\cdot n$ in $\mathbf{D}$. All other fluxes from Table 3.1 use Ψ_P instead.

PVM Methods

A generalization of the two previous methods, introduced in [6], are the so-called PVM methods, where we take $\Theta = P(\mathbf{A}_{1/2} \cdot n)$ where P is a polynomial in the Roe matrix. The coefficients are also problem-dependent and may be completely nonlinear.

For the Rusanov flux, it is obvious that it can be represented in such a way, and the Roe flux (or rather: the absolute value of the matrix) can be written as a polynomial of degree $N-1$ (where N is so that $(\mathbf{A}_{1/2} \cdot n) \in \mathbb{R}^{N\times N}$).

Another flux which can be written in the PVM format is the HLL method. We get

$$P(A) = \alpha_0 I_{5,6} + \alpha_1 A, \tag{3.3.43}$$

where

$$\alpha_0 = \frac{S_R\,|S_L| - S_L\,|S_R|}{S_R - S_L} \tag{3.3.44}$$

$$\alpha_1 = \frac{|S_R| - |S_L|}{S_R - S_L}, \tag{3.3.45}$$

and

$$S_L = \min\{\lambda_1(\mathbf{A}(\mathbf{U}_L)\cdot n), \lambda_1(\mathbf{A}_{1/2}\cdot n)\} \tag{3.3.46}$$

$$S_R = \max\{\lambda_3(\mathbf{A}(\mathbf{U}_R)\cdot n), \lambda_3(\mathbf{A}_{1/2}\cdot n)\}. \tag{3.3.47}$$

We implemented the HLL flux as well. It is also rotationally invariant, since it is a linear combination of two rotationally invariant matrices. The matrix $I_{5,6}$ is defined as

$$I_{5,6} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{3.3.48}$$

3.3.4 DOT Flux

The DOT flux [5, 13][4] is written as

$$\Theta(\mathbf{U}) = |\mathbf{A}(\mathbf{U}) \cdot n| \,. \tag{3.3.49}$$

It may look similar to the Roe flux, yet the matrix varies w.r.t. $\mathbf{U}$ this time.[5]This makes the exact computation of the integral more difficult, even in the case of Ψ_P: we would need to determine the points where all the eigenvalues switch sign.

We implemented the most simple version of the DOT flux which is also suggested in [13]. That is, we use linear paths in primitive variables, i.e. Ψ_P, and we use a three-point Gauss-Legendre quadrature to approximate it.

Just as we have done with the Roe flux, we obtain that the DOT flux is rotationally invariant as well.

3.4 Boundary Conditions

We allow four different boundary configurations, mainly based on empirical behavior. For all numerical methods which we consider, we only need to deal with the boundary over the numerical flux. This means that we are given $\mathbf{U}_L$ (or $\mathbf{U}_R$) at a face $F \subseteq \partial\Omega$, and we need to return $\mathbf{U}_R$ (or $\mathbf{U}_L$). Write $\mathbf{U}_L = (h_L, (h\mathbf{u})_L, (hw)_L, (hp)_L, b_L)^T$, and $\mathbf{U}_R = (h_R, (h\mathbf{u})_R, (hw)_R, (hp)_R, b_R)^T$, and we are some position (t_b, x_b). Then, we consider the folloing choices which are also written into Table 3.2.

- A given target solution: we are given a $\mathbf{U}_{\text{bnd}}(t, x)$ for $x \in \partial\Omega$. Ideally, $\mathbf{U}_{\text{bnd}}(t, x)$ should be the solution to the given initial conditions, or at least close to it. Then, we set $\mathbf{U}_R = \mathbf{U}_{\text{bnd}}(t_b, x_b)$.
- Absorbing boundary conditions, or resting lake boundary conditions. This means that we set $h\mathbf{u}$, hw, and hp to zero, and $h = -b$, and so we get

$$\mathbf{U}_R = (-b_L, 0, 0, 0, b_L)^T. \tag{3.4.1}$$

[4] In [15] also called the "generalized Osher-Solomon" flux. In fact, the name DOT seems to be introduced by [5] and stands for "Dumbser-Osher-Toro".

[5] One may even conjecture, if it is useful to consider a generalization of PVM methods by replacing the Roe average matrix by the path integral. e.g. for the HLL flux above, we would replace $\mathbf{A}_{1/2} \cdot n$ in S_L and S_R, as well as in P itself by the parameter $\mathbf{A} \cdot n$ which varies during the path.

Handle	h_{out}	$(hu)_{\text{out}}$	$(hv)_{\text{out}}$	$(hw)_{\text{out}}$	$(hp)_{\text{out}}$
`scenario`	h_{bnd}	$(hu)_{\text{bnd}}$	$(hv)_{\text{bnd}}$	$(hw)_{\text{bnd}}$	$(hp)_{\text{bnd}}$
`absorbing`	$-b_{\text{in}}$	0	0	0	0
`keep_height`	h_{in}	0	0	0	0
`keep_no_velocity`	h_{in}	0	0	$(hw)_{\text{in}}$	$(hp)_{\text{in}}$
`keep_all`	h_{in}	$(hu)_{\text{in}}$	$(hv)_{\text{in}}$	$(hw)_{\text{in}}$	$(hp)_{\text{in}}$

Table 3.2: A list of the available boundary conditions in our implementation. Here, $\mathbf{U}_{\text{in}}$ is the data from the adjacent cell, $\mathbf{U}_{\text{bnd}}$ is the prescribed boundary condition, and $\mathbf{U}_{\text{out}}$ is the boundary value.

- Height-preserving boundary conditions. We keep the height value h, but all other values are set to zero. Thus, $h\mathbf{u}$, hw, hp are set to zero. Thus, we get $\mathbf{U}_R = (h_L, 0, 0, 0, b_L)^T$.
- All-but-velocity-preserving boundary conditions. We keep the boundary values as they are, and only set $h\mathbf{u}$ to zero. Therefore, we have

$$\mathbf{U}_R = (h_L, 0, (hw)_L, (hp)_L, b_L)^T. \tag{3.4.2}$$

While we could also attempt to set $\mathbf{U}_R = \mathbf{U}_L$, this has shown to lead to unstable solutions in practice. We conjecture that this is related to the equation for (hp), see e.g. [9].

Given $\mathbf{U}_L$ and $\mathbf{U}_R$, we then compute the flux $\mathbf{D}(\mathbf{U}_L, \mathbf{U}_R) \cdot n$. In particular, we do not need to evaluate the flux into the other direction—or if we do, we discard the other value.

3.5 The ADER-DG Method

Next, we discuss the main method which we are going to use. It is the ADER-DG method, similarly to as done in [15]. The ADER-DG method is a space-time DG method. This means that it uses the Discontinuous Galerkin formalism not only for space discretization, but also for the time evolution. As a result, we have space-time cells—which is why we introduced the space-time bases above in the first place. This contrasts with the regular, well-known method of lines approach where we only use the DG method in space to get a semi-discrete equation which is then solved with a "normal" ODE solver, or a variant thereof. For an analysis of the semi-discrete DG methods, we refer to [10]. We will not handle them any further here, although the finite volume implementation described in the previous section can be seen as a DG method as well.

There have been numerous papers which describe the method itself. To give a small overview over two papers that were most useful for applying the method, we suggest the following papers: the paper [11] introduces the method itself for $\mathbf{A} \equiv \text{const}$, and analyzes the method. This paper also includes a von Neumann stability analysis. In

addition, the paper [47] offers a good introduction to the discrete maximum principle limiter.[6]

The ADER-DG method can be split into two main steps, plus an additional finite volume limiter. See Algorithm 3.1 for a high-level description of a single step in the procedure.

Input: Timestep Δt, cell-local previous solution $\mathbf{U}_n$, cell state S
Output: The next timestep solution $\mathbf{U}_{n+1}$.
if *Cell state S is DG* **then**
  Predictor: Solve (3.5.3) with $\mathbf{U}_n$ and Δt, get $\mathbf{Q}_n$;
  Corrector: Compute (3.5.21), get $\mathbf{U}_{n+1}$;
end
if *If S is FV, an error occurred or $\mathbf{U}_{n+1}$ violates Definition 3.1* **then**
  Limiter: Solve (3.2.11) with the projected averages;
end

Algorithm 3.1: A step of the ADER-DG method with limiter.

In total, we need the following ingredients to define an ADER-DG method:

- A mesh $\mathcal{T}_{\mathrm{DG}}$ which was defined in Section 3.1, and a subdivided mesh $\mathcal{T}_{\mathrm{FV}}$ for the limiter. We need $\mathcal{T}_{\mathrm{FV}}$ to be subdivision of $\mathcal{T}_{\mathrm{DG}}$, i.e. each cell in $\mathcal{T}_{\mathrm{FV}}$ is contained in exactly one cell in $\mathcal{T}_{\mathrm{DG}}$.
- A predictor function space V_P in space and time defined on $\mathcal{T}_{\mathrm{DG}}$.
- A corrector function space V_C in space only, also defined on $\mathcal{T}_{\mathrm{DG}}$.

We will now explain the two steps of the ADER-DG method, as well as the inclusion of the finite volume limiter afterwards.

3.5.1 Predictor Step

For the predictor step, we assume that we know the solution for the previous time step $\mathbf{U}_n(x) \in V_C$. The goal is to determine a predictor function $\mathbf{Q}(t,x) \in V_P$ which should follow the local behavior of the PDE, i.e. of (2.2.1). For this, we multiply our equation with a test function $\mathbf{V} \in V_P$, then we integrate the equation in space and time. For brevity, we will write point-wise multiplication between vector-valued functions by writing nothing in-between them. So the following means the multiplication of the equation with the test function $\mathbf{V}$ in each component. So, we have

$$\int_{[t^n,t^{n+1}]\times T} \mathbf{V}(t,x) \odot (\partial_t \mathbf{Q} + \nabla \cdot \mathbf{F}(\mathbf{Q}) + \mathbf{B}(\mathbf{Q}) \cdot \nabla \mathbf{Q} - S(\mathbf{Q}, \nabla \mathbf{Q}))\, \mathrm{d}(t,x) = 0. \quad (3.5.1)$$

[6] The DMP check is actually introduced in [14], however, the value for δ suggested there is not the one used in later works.

Next, we use Fubini's Theorem (once again, separately on each component) and integrate the right-hand side by parts in time, and replace $\mathbf{Q}(0,x)$ by $\mathbf{U}_n(x)$, and obtain

$$\begin{aligned}
&\int_{[t^n,t^{n+1}]\times T} \mathbf{V}(t,x) \odot \partial_t \mathbf{Q}(t,x) \,\mathrm{d}(t,x) \\
&= \int_T \mathbf{V}(1,x) \odot \mathbf{Q}(1,x) \,\mathrm{d}x - \int_T \mathbf{V}(0,x) \odot \mathbf{U}_n(x) \,\mathrm{d}x \\
&- \int_{[t^n,t^{n+1}]\times T} \partial_t \mathbf{V}(t,x) \odot \mathbf{Q} \,\mathrm{d}(t,x).
\end{aligned} \tag{3.5.2}$$

Inserting this into equation (3.5.1) and re-ordering some terms gives us

$$\begin{aligned}
&\int_T \mathbf{V}(1,x) \odot \mathbf{Q}(1,x) \,\mathrm{d}x - \int_{[t^n,t^{n+1}]\times T} \partial_t \mathbf{V}(t,x) \odot \mathbf{Q}(t,x) \,\mathrm{d}(t,x) \\
&= \int_T \mathbf{V}(0,x) \odot \mathbf{U}_n(x) \,\mathrm{d}x \\
&- \int_{[t^n,t^{n+1}]\times T} \mathbf{V}(t,x) \odot (\nabla \cdot \mathbf{F}(\mathbf{Q}) + \mathbf{B}(\mathbf{Q}) \cdot \nabla \mathbf{Q} - S(\mathbf{Q}, \nabla \mathbf{Q})) \,\mathrm{d}(t,x).
\end{aligned} \tag{3.5.3}$$

We now solve (3.5.3) for $\mathbf{Q}$ which will be done used a fixed-point iteration. To make this more clear, note that the right-hand side is linear in $\mathbf{V}$, and the left-hand side is bilinear (in $\mathbf{V}$ and $\mathbf{Q}$). Hence, if we fix $\mathbf{Q}$ on the right-hand side, this problem ultimately has the structure

$$a(\mathbf{Q}, \mathbf{V}) = f[\mathbf{U}_n, \mathbf{Q}](\mathbf{V}) \tag{3.5.4}$$

Thus, if we are able to invert a, we can try to solve for $\mathbf{Q}$ by using a fixed-point iteration.

We note that the predictor is an entirely local operation and no communication with other elements is required. Yet, since we need to solve a fixed-point iteration, it is in practice a rather compute-intensive operation.

Transformation to the Reference Element

Inserting the transformation to the reference element (3.1.20) into the predictor equation (3.5.3), we get

$$\begin{aligned}
&h^2 \int_{T_{\mathrm{ref}}} \mathbf{V}(1,x) \odot \mathbf{Q}(1,x) \,\mathrm{d}x - (\Delta t) h^2 \int_{I_{\mathrm{ref}}\times T_{\mathrm{ref}}} \frac{1}{\Delta t} \partial_t \mathbf{V}(t,x) \odot \mathbf{Q}(t,x) \,\mathrm{d}(t,x) \\
=&h^2 \int_{T_{\mathrm{ref}}} \mathbf{V}(0,x) \odot \mathbf{U}_n(x) \,\mathrm{d}x \\
&- (\Delta t) h^2 \int_{I_{\mathrm{ref}}\times T_{\mathrm{ref}}} \mathbf{V}(t,x) \odot \left(\frac{1}{h} \mathbf{A}(\mathbf{Q}) \cdot (R \nabla \mathbf{Q}) - S(\mathbf{Q}, \frac{1}{h} R \nabla \mathbf{Q}) \right) \mathrm{d}(t,x).
\end{aligned} \tag{3.5.5}$$

Simplifying the equation (including removing the h^2 term) gives

$$\begin{aligned}
&\int_{T_{\text{ref}}} \mathbf{V}(1,x) \odot \mathbf{Q}(1,x) \,\mathrm{d}x - \int_{I_{\text{ref}} \times T_{\text{ref}}} \partial_t \mathbf{V}(t,x) \odot \mathbf{Q}(t,x) \,\mathrm{d}(t,x) \\
&= \int_{T_{\text{ref}}} \mathbf{V}(0,x) \odot \mathbf{U}_n(x) \,\mathrm{d}x \\
&\quad - (\Delta t) \int_{I_{\text{ref}} \times T_{\text{ref}}} \mathbf{V}(t,x) \odot \left(\frac{1}{h} \mathbf{A}(\mathbf{Q}) \cdot (R \nabla \mathbf{Q}) - S(\mathbf{Q}, \frac{1}{h} R \nabla \mathbf{Q}) \right) \mathrm{d}(t,x).
\end{aligned} \tag{3.5.6}$$

Most notably, the left-hand side does not depend on Δt.

Here, the application of $R \in SO(2)$ onto $\nabla \mathbf{Q}$ is defined as

$$R \nabla \mathbf{Q} = \begin{pmatrix} R_{11} \partial_x \mathbf{Q} + R_{12} \partial_y \mathbf{Q} \\ R_{21} \partial_x \mathbf{Q} + R_{22} \partial_y \mathbf{Q} \end{pmatrix}. \tag{3.5.7}$$

Matrix Formulation

Next, we want to formulate (3.5.6) using the previously-defined bases B_P and B_C. Therefore, we need to introduce a coefficient vector for each of the variables h, hu, hv, hw, hp, and b. Since they all have the same basis, we can also write their coefficients conveniently into a matrix. Suppose we have $U^n \in \mathbb{R}^{b_C \times 6}$ as matrix of basis coefficients. We now want to write (3.5.6) in matrix form, so that we may compute the predictor update coefficient matrix $Q^n \in \mathbb{R}^{b_P \times 6}$. Conveniently, we can handle each line of the equation (3.5.6) separately.

Firstly, we introduce the matrix $\mathcal{R} \in \mathbb{R}^{b_P \times b_P}$ for the left-hand side of the predictor equation. This is for the first line in (3.5.6). That is, we have

$$\mathcal{R}_{ij} = \int_{T_{\text{ref}}} v_i(1,x) v_j(1,x) \,\mathrm{d}x - \int_{I_{\text{ref}} \times T_{\text{ref}}} \partial_t v_i(t,x) v_j(t,x) \,\mathrm{d}(t,x). \tag{3.5.8}$$

Naturally, we need $\mathcal{R}$ to be invertible. In practice, this is always the case for our choice of V_P. For the second line of (3.5.6) which the first term of the right-hand side, we introduce the matrix $U^{n,\text{init}} \in \mathbb{R}^{b_P}$ defined as

$$(U^{n,\text{init}})_{ij} = U^n_{ij} \int_{T_{\text{ref}}} v_i(0,x) v_j(0,x) \,\mathrm{d}x \tag{3.5.9}$$

which is essentially an extension of U^n to the space-time cell. We do not introduce a separate symbol for the mass matrix of B_P here, since we do not need it anywhere other than here.

The nonlinear part, i.e. the last line of (3.5.6) needs more discussion. We discretize it by choosing a grid of quadrature points for $I_{\text{ref}} \times T_{\text{ref}}$. Let d_C be the number of quadrature points, and $p^C \in \mathbb{R}^{d_C}$ be the vector of quadrature points in space and time. That means that $p^C_i \in I_{\text{ref}} \times T_{\text{ref}}$. Then let $q^C \in \mathbb{R}^{d_C}$ be the vector of weights of the quadrature rule, and so $q^C_i \in \mathbb{R}^+$. Next, we introduce the collocation matrix $\mathcal{C} \in \mathbb{R}^{d_C \times b_P}$, defined as

$$\mathcal{C}_{ij} = v_j(p^C_i) \tag{3.5.10}$$

as well as the collocation matrices $\mathcal{C}^x, \mathcal{C}^y \in \mathbb{R}^{d_C \times b_P}$ for the derivatives

$$\mathcal{C}^x_{ij} = \frac{1}{h}(R_{11}\partial_x v_j(p_i^C) + R_{12}\partial_y v_j(p_i^C)) \tag{3.5.11}$$

$$\mathcal{C}^y_{ij} = \frac{1}{h}(R_{21}\partial_x v_j(p_i^C) + R_{22}\partial_y v_j(p_i^C)), \tag{3.5.12}$$

and the weighted transposed collocation matrix $\mathcal{D} \in \mathbb{R}^{b_P \times d_C}$ as

$$\mathcal{D}_{ij} = v_i(p_j^C) q_j^C. \tag{3.5.13}$$

Thus, we get for a quadrature rule of sufficiently high order that

$$(\mathcal{D}\mathcal{C})_{ij} = \int_{I_{\text{ref}} \times T_{\text{ref}}} v_i(t,x) v_j(t,x) \, \mathrm{d}(t,x). \tag{3.5.14}$$

And we also write

$$(\mathcal{D} f(\mathcal{C}y))_i \approx \int_{I_{\text{ref}} \times T_{\text{ref}}} v_i(x) f\left(\sum_j y_j v_j(x)\right) \mathrm{d}(t,x) \tag{3.5.15}$$

for any function $f : \mathbb{R} \to \mathbb{R}$. Here, we mean by $f(\mathcal{C}x)$ that f is applied on each quadrature point separately. For the order of the quadrature d_C, we will have to integrate over an expression of the form $\mathbf{V}\mathbf{A}(\mathbf{Q})\nabla\mathbf{Q}$, where $\mathbf{V}$ is a polynomial of degree N_P. Since we take the derivative, $\nabla\mathbf{Q}$ has degree $(N_P - 1)$, and $\mathbf{A}(\mathbf{Q}) \cdot n$ only contains primitive variables other than h. Therefore, $\mathbf{A}(\mathbf{Q}) \cdot n$ only contains rational functions, with the same degree in denominator and numerator. Thus, we choose for the scope of this work $d_C = (2N_P + 1)$, since we cannot compute $\mathbf{A}(\mathbf{Q}) \cdot n$ exactly. Other orders, with at least $(2N_P - 1)$ would be possible as well.

In total, we get the following fixed-point iteration

$$\begin{aligned} &Q^{n,i+1} \\ &= \mathcal{R}^{-1} U^{n,\text{init}} \\ &+ (\Delta t)\mathcal{R}^{-1}\mathcal{D}\left(\mathbf{A}_1(\mathcal{C}Q^{n,i})\mathcal{C}^x Q^{n,i} + \mathbf{A}_2(\mathcal{C}Q^{n,i})\mathcal{C}^y Q^{n,i}\right) \\ &+ \mathcal{R}^{-1}\mathbf{S}(\mathcal{C}Q^{n,i}, (\mathcal{C}^x Q^{n,i}, \mathcal{C}^y Q^{n,i})). \end{aligned} \tag{3.5.16}$$

Here, $\mathbf{A}_1$ and $\mathbf{A}_2$ come from expanding $(\mathbf{A}(\mathbf{Q}) \cdot (R\nabla\mathbf{Q}))$. Once again, the term $\mathbf{A}_1(\mathcal{C}Q^{n,i})$ is understood as computing the values at the quadrature points for the coefficients from $Q^{n,i}$, and then sending each quadrature point through $\mathbf{A}_1$. The other terms are understood similarly. We abort with iterating by (3.5.16), once we reach

$$\left\| Q^{n,i} - Q^{n,i+1} \right\|_F < \epsilon_c \tag{3.5.17}$$

for some i. Then Q^n is set to $Q^{n,i}$. Here, $\epsilon_c > 0$ is given as a parameter. If we do not get to (3.5.17) after some iterations (which is also a parameter which can be chosen freely),

we declare the predictor to be divergent. The actual norm we use in (3.5.17) does not matter, so for the ease of use, we choose the Frobenius norm. As initial guess $Q^{n,0}$, we simply take $U^{n,\text{init}}$.

The convergence of the predictor iteration (3.5.16) is not much discussed in the papers to our knowledge. Only [4] includes some proof ideas for the convergence of the predictor, however the proof given there only works if $\mathbf{F}$ is defined on all of $\mathbb{R}^N$, and it is globally Lipschitz continuous. Furthermore, [11] argues about the convergence, for the linear case.

3.5.2 Corrector Step

For the corrector step, we again take the given $\mathbf{U}_n(x) \in V_C$, as well as a predictor solution $\mathbf{Q}(t,x) \in V_P$. We now want to determine the solution for the next step, i.e. we have the unknown function $\mathbf{U}_{n+1}(x) \in V_C$. This time, we a test function $\mathbf{W} \in V_C$. This will be again symbolized by the symbol $\odot$. Once again, we will write the function arguments (t,x), and (x) only to reduce ambiguity, when this is needed. Next, we again integrate in space and time and get again

$$\int_{[t^n,t^{n+1}]\times T} \mathbf{W}(x) \odot (\partial_t \mathbf{Q} + \nabla \cdot \mathbf{F}(\mathbf{Q}) + \mathbf{B}(\mathbf{Q}) \cdot \nabla \mathbf{Q} - S(\mathbf{Q}, \nabla \mathbf{Q}))\, \mathrm{d}(t,x) = 0. \quad (3.5.18)$$

Next, we integrate by parts in time for each element. We insert, again, $\mathbf{U}_n(x)$ for $\mathbf{Q}(t^n,x)$, and now also $\mathbf{U}_{n+1}(x)$ for $\mathbf{Q}(t^{n+1},x)$. This gives us

$$\begin{aligned} &\int_{[t^n,t^{n+1}]\times T} \mathbf{W}(x) \odot \partial_t \mathbf{Q}(t,x)\, \mathrm{d}(t,x) \\ &= \int_T \mathbf{W}(x) \odot \mathbf{U}_{n+1}(x)\, \mathrm{d}x \\ &- \int_T \mathbf{W}(x) \odot \mathbf{U}_n(x)\, \mathrm{d}x. \end{aligned} \quad (3.5.19)$$

Note that in comparison to the predictor, the integral over $\partial_t \mathbf{W}(x)\mathbf{Q}(t,x)$ disappears. This is due to the fact that $\mathbf{W}$ does not depend on t, hence $\partial_t \mathbf{W} \equiv 0$. In addition to that, we now take $\mathbf{F}$ and integrate it by parts in space (compare Gauss divergence theorem). For the now needed boundary integral, we use the previously-introduced numerical flux $\mathbf{D}$, and get

$$\begin{aligned} &\int_{[t^n,t^{n+1}]\times T} \mathbf{W}(x) \odot (\nabla \cdot \mathbf{F}(\mathbf{Q}))\, \mathrm{d}(t,x) \\ &= \int_{[t^n,t^{n+1}]\times \partial T} \mathbf{W}(x) \odot (\mathbf{D}(\mathbf{Q}^-, \mathbf{Q}^+) \cdot n)\, \mathrm{d}(t,x) \\ &- \int_{[t^n,t^{n+1}]\times T} \mathbf{F}(\mathbf{Q}) \cdot \nabla \mathbf{W}(x)\, \mathrm{d}(t,x). \end{aligned} \quad (3.5.20)$$

In total, by inserting (3.5.19) and (3.5.20) into (3.5.18), we obtain the equation

$$\begin{aligned}
&\int_T \mathbf{W}(x) \odot \mathbf{U}_{n+1}(x)\,\mathrm{d}x \\
=&\int_T \mathbf{W}(x) \odot \mathbf{U}_n(x)\,\mathrm{d}x \\
&- \int_{[t^n,t^{n+1}]\times\partial T} \mathbf{W}(x) \odot (\mathbf{D}(\mathbf{Q}^-,\mathbf{Q}^+)\cdot n)\,\mathrm{d}(t,x) \\
&+ \int_{[t^n,t^{n+1}]\times T} \mathbf{F}(\mathbf{Q})\cdot\nabla\mathbf{W}(x)\,\mathrm{d}(t,x) \\
&- \int_{[t^n,t^{n+1}]\times T} \mathbf{W}(x) \odot (\mathbf{B}(\mathbf{Q})\cdot\nabla\mathbf{Q} + S(\mathbf{Q},\nabla\mathbf{Q}))\,\mathrm{d}(t,x).
\end{aligned} \tag{3.5.21}$$

While this equation is more complicated than the equation for the predictor (3.5.3), we only need it to run once to compute $\mathbf{U}_{n+1}$. However, it is not an element-local operation anymore, since we need the boundary values from neighboring cells or the boundary conditions.

Transformation to the Reference Element

Next, we insert the transformations (3.1.20) into the corrector equation (3.5.21) to get

$$\begin{aligned}
&h^2\int_{T_{\mathrm{ref}}} \mathbf{W}\odot\mathbf{U}_{n+1}\,\mathrm{d}x \\
=&h^2\int_{T_{\mathrm{ref}}} \mathbf{W}\odot\mathbf{U}_n\,\mathrm{d}x \\
&-(\Delta t)h\int_{[0,1]\times\partial T_{\mathrm{ref}}} \mathbf{W}(x)\odot(\mathbf{D}(\mathbf{Q}^-,\mathbf{Q}^+)\cdot n)\,\mathrm{d}(t,x) \\
&+(\Delta t)h^2\int_{[0,1]\times T_{\mathrm{ref}}} \mathbf{F}(\mathbf{Q})\cdot\frac{1}{h}R\nabla\mathbf{W}(x)\,\mathrm{d}(t,x) \\
&-(\Delta t)h^2\int_{[0,1]\times T_{\mathrm{ref}}} \mathbf{W}\odot\left(\frac{1}{h}\mathbf{B}(\mathbf{Q})\cdot R\nabla\mathbf{Q} - S(\mathbf{Q},\frac{1}{h}R\nabla\mathbf{Q})\right)\mathrm{d}(t,x)
\end{aligned} \tag{3.5.22}$$

It should be noted that the face integral (i.e. line 3) only gets the scaling factor h instead of h^2, since the faces are all one-dimensional. Next, we again divide by h^2 to obtain

$$\begin{aligned}
&\int_{T_{\text{ref}}} \mathbf{W} \odot \mathbf{U}_{n+1} \, \mathrm{d}x \\
=&\int_{T_{\text{ref}}} \mathbf{W} \odot \mathbf{U}_n \, \mathrm{d}x \\
&- \frac{\Delta t}{h} \int_{[0,1]\times\partial T_{\text{ref}}} \mathbf{W}(x) \odot (\mathbf{D}(\mathbf{Q}^-, \mathbf{Q}^+) \cdot n) \, \mathrm{d}(t,x) \\
&+ \frac{\Delta t}{h} \int_{[0,1]\times T_{\text{ref}}} \mathbf{F}(\mathbf{Q}) \cdot R\nabla\mathbf{W}(x) \, \mathrm{d}(t,x) \\
&- (\Delta t) \int_{[0,1]\times T_{\text{ref}}} \mathbf{W} \odot \left(\frac{1}{h}\mathbf{B}(\mathbf{Q}) \cdot R\nabla\mathbf{Q} - S(\mathbf{Q}, \frac{1}{h} R\nabla\mathbf{Q}) \right) \mathrm{d}(t,x).
\end{aligned} \tag{3.5.23}$$

Matrix Formulation

In order to convert to the corrector equation (3.5.23) into a matrix formulation, we proceed as we did for the predictor. Let U^n be the coefficients of the previous timestep, and Q^n the result from the predictor iteration (3.5.16), and let U^{n+1} be the coefficient vector we are looking for. For convenience, we decompose U^{n+1} into

$$U^{n+1} = U^n + U^{n+1,\text{local}} + U^{n+1,\text{edge}}, \tag{3.5.24}$$

where $U^{n+1,\text{edge}}$ shall contain the discretized face integral, and $U^{n+1,\text{local}}$ the rest. In comparison to (3.5.23), U^{n+1} corresponds to the first line, U^n to the second line, $U^{n+1,\text{edge}}$ to the third line, and $U^{n+1,\text{local}}$ to the fourth and fifth line.

We introduce the spatial mass matrix $\mathcal{M} \in \mathbb{R}^{b_C \times b_C}$ defined element-wise as

$$\mathcal{M}_{ij} = \int_{T_{\text{ref}}} w_i w_j \, \mathrm{d}x. \tag{3.5.25}$$

With our choice of basis, $\mathcal{M}$ has shown to be always invertible in practice. With $\mathcal{M}$, we are able to write

$$(\mathcal{M}U^n)_{ij} = U^n_{ij} \int_{T_{\text{ref}}} w_i w_j \, \mathrm{d}x, \tag{3.5.26}$$

$$(\mathcal{M}U^{n+1})_{ij} = U^{n+1}_{ij} \int_{T_{\text{ref}}} w_i w_j \, \mathrm{d}x. \tag{3.5.27}$$

In addition, we re-use the collocation matrices $\mathcal{C}$ and $\mathcal{C}^x$, and $\mathcal{C}^y$. This is because we are inserting the predictor into the equation again. But this time, we need space-only weighted and transposed collocation matrices $\mathcal{E}, \mathcal{E}^x, \mathcal{E}^y \in \mathbb{R}^{b_C \times d_C}$. Therefore, we define

for the same quadrature grid (p^C, q^C) the matrices

$$\mathcal{E}_{ij} = w_i(p_j^C)q_j^C, \tag{3.5.28}$$

$$\mathcal{E}^x_{ij} = \frac{1}{h}(R_{11}\partial_x w_i(p_j^C) + R_{12}\partial_y w_i(p_j^C))q_j^C, \tag{3.5.29}$$

$$\mathcal{E}^y_{ij} = \frac{1}{h}(R_{21}\partial_x w_i(p_j^C) + R_{22}\partial_y w_i(p_j^C))q_j^C. \tag{3.5.30}$$

We have $w_i \in B_C$ being functions, and q_j^C being constants. Note that again, we included the rotation matrices here already. Thus, we obtain for lines four and five from (3.5.23)

$$\begin{aligned} U^{n+1,\text{local}} =& (\Delta t)\mathcal{M}^{-1}\left(\mathcal{E}^x\partial_x \mathbf{F}_1(\mathcal{C}Q^n) + \mathcal{E}^y\partial_y \mathbf{F}_2(\mathcal{C}Q^n)\right) \\ &- (\Delta t)\mathcal{M}^{-1}\mathcal{E}\left(\mathbf{B}_1(\mathcal{C}Q^n)\mathcal{C}^x Q^n + \mathbf{B}_2(\mathcal{C}Q^n)\mathcal{C}^y Q^n\right) \\ &+ (\Delta t)\mathcal{M}^{-1}\mathbf{S}(\mathcal{C}Q^n, (\mathcal{C}^x Q^n, \mathcal{C}^y Q^n)). \end{aligned} \tag{3.5.31}$$

Once again, the application of a matrix on $\mathbf{F}_i$, $\mathbf{B}_i$, and $\mathbf{S}$ is understood as the application on each quadrature point separately. That is, the functions are applied on each row.

It remains define $U^{n+1,\text{edge}}$. For this, we need to discretize the face integrals. Thus, we require a new quadrature rule with d_F points over $I_{\text{ref}} \times F_{\text{ref}}$ which we write (p^F, q^F). We choose a Gauss-Legendre quadrature of degree $2N$ in space and time, and transform (p^F, q^F) to each face $F_{1,\text{ref}}, F_{2,\text{ref}}, F_{3,\text{ref}}$, to get $(p^{F_k}, q^{F_k}) \in \mathbb{R}^{d_F} \times \mathbb{R}^{d_F}$ which is a quadrature for F_k, and $k = 1, 2, 3$. In particular, we get that $w_i^{F_1} = \sqrt{2}w_i^{F_2} = w_i^{F_3}$ for all i, since as the hypotenuse, $F_{2,\text{ref}}$ is longer than $F_{1,\text{ref}}$ and $F_{3,\text{ref}}$. We order the points additionally in counter-clockwise order around the triangle. In particular, p^{F_1} lets its points go from $(0,0)^T$ to $(1,0)^T$, for p^{F_2}, they go from $(1,0)^T$ to $(0,1)^T$, and for p^{F_3}, they go from $(0,1)^T$ to $(0,0)^T$. Then, we introduce matrices $\mathcal{F}^k$ and $\mathcal{G}^k$ for each face (i.e. $k = 1, 2, 3$), so that

$$\mathcal{F}^k_{ij} = w_j(p_i^{F_k}), \tag{3.5.32}$$

$$\mathcal{G}^k_{ij} = w_i(p_j^{F_k})q_j^{F_k}. \tag{3.5.33}$$

While $q_j^{F_k}$ denoted the quadrature weights, $w_i, w_j \in B_C$ denoted functions from the basis. Suppose for now that we have a cell T that does not touch the boundary. Let $\hat{Q}^{n,1}$, $\hat{Q}^{n,2}$, and $\hat{Q}^{n,3}$ be the predictor results from the neighboring cells over the edges F_1, F_2 and F_3, respectively. Then, the face integral (row three from (3.5.23)) can be written as

$$U^{n+1,\text{edge}} = -\frac{\Delta t}{h}\mathcal{M}^{-1}\sum_{k=1,2,3}\mathcal{G}^k(\mathbf{D}(\mathcal{F}^k Q^n, \mathcal{F}^k\hat{Q}^{n,k})\cdot n^k). \tag{3.5.34}$$

Again, this notation is understood as that $\mathbf{D}$ is evaluated for each quadrature point, i.e. for each row of $\mathcal{F}^k Q^n$ and $\mathcal{F}^k\hat{Q}^{n,k}$, respectively. Three peculiarities have to be observed.

- If a cell touches the boundary at face k, we replace $\mathcal{F}^k\hat{Q}^k$ by the chosen boundary values $\hat{Q}^k_{\text{bnd}}$ which can be determined per quadrature point, as considered in Section 3.4.

- Suppose we have two cells Q^1 and Q^2 which share a face F with matrices $\mathcal{F}^{Q^1}$ and $\mathcal{F}^{Q^2}$, then the respective quadrature point evaluations of $\mathcal{F}^{Q^1}Q^1$ and $\mathcal{F}^{Q^2}Q^2$ are not the same. In fact, they have their spatial order exactly reversed, e.g. if the points from the side on Q^1 run from $(0,0)^T$ to $(1,0)^T$, they run from $(1,0)^T$ to $(0,0)^T$ on the other side. This is the case, because if we go through Q^1 and Q^2 in counter-clockwise order, we go through $\mathcal{F}$ in opposite directions, and thus, the quadrature points (which we have ordered before), are reversed as well. This is also the case for the finite volume patches.

- In its current formulation, we compute $\mathbf{D}$ at each quadrature point. This way, we can experiment with the effect of different numerical fluxes in the DG formulation, and it is better to test the correctness of the model itself. However, we note that it may be in the long run preferrable to switch to a different method which evaluates and integrates over $\mathbf{D}$ before-hand on each cell and uses some approximation for Θ, as it is e.g. used in [37].

3.5.3 Limiter

Since the ADER-DG method fails at wet-dry boundaries, we also include a finite volume-style limiter (again following [15]). This means that we subdivide each cell $T \in \mathcal{T}_{\mathrm{DG}}$ into smaller cells on which we run the finite volume method described in Section 3.2, and we obtain a mesh $\mathcal{T}_{\mathrm{FV}}$ which contains all subcells which will from now on be referred to as patches. Thus, we require that for $\tau \in \mathcal{T}_{\mathrm{FV}}$, there is a $T \in \mathcal{T}_{\mathrm{DG}}$, so that $\tau \subseteq T$. In addition, $\mathcal{T}_{\mathrm{FV}}$ should be conforming as well.

This can be seen as if we take V_C to just cover all constant functions, and V_P to be constant in space. This amounts to that we could in the corrector equation simply remove all references to $\mathbf{W}$ (which now just represented a constant function), as well as all derivatives of $\mathbf{W}$ or $\mathbf{Q}$. Also, we remove the predictor iteration, and assume a constant evolution in time. The only part which may not be constant in this formulation is the flux which potentially arrives from a valid DG cell (since a finite volume and a DG cell might lie next to each other).

Projection to the Limiter

Given a cell $T \in \mathcal{T}_{\mathrm{DG}}$, denote by $\tau_{1,T}, ..., \tau_{k,T} \in \mathcal{T}_{\mathrm{FV}}$ its patches. This means that we have $\tau_{i,T} \in T$ for all $i = 1, ..., k$. It is important that we have $k \geq |B_C|$, so that the projection to the limiter and back is possible without a loss. We subsequently use the L^2 projection to obtain a matrix $\mathcal{P}_{\mathrm{lim}}$ which is defined as

$$(\mathcal{P}_{\mathrm{lim}})_{ij} = \frac{1}{|\tau_{i,T}|} \int_{\tau_{i,T}} w_{j,T}(x) \, \mathrm{d}x. \tag{3.5.35}$$

Here, $|\tau_{i,T}|$ denotes the volume of $\tau_{i,T}$, and $w_{j,T}$ is the basis function w_j transformed to T. The normalization mass matrix normally seen in the L^2 projection has just become the $1/|\tau_i|$, since the closures of $\tau_1, ..., \tau_k$ forms a partition of T. For the projection from

$\tau_{1,T}, ..., \tau_{k,T}$ back to T, we apply the Moore-Penrose inverse $\mathcal{P}_{\text{lim}}^{\dagger}$. Since we required $k \geq \dim(V_P)$, we get $\mathcal{P}_{\text{lim}}^{\dagger}\mathcal{P}_{\text{lim}} = I$ by construction.

For simplicity, we choose the same set of patches for all cells (although this is not required by the formulation we just considered). Then $\mathcal{P}_{\text{lim}}$ is the same matrix for all cells. To see this, consider the reference patches $\tau_{1,\text{ref}}, ..., \tau_{k,\text{ref}}$. Naturally, we get $\tau_{i,T} = P_{h,R,b}(\tau_{i,\text{ref}})$ for all $i = 1, ..., k$. Subsequently, we obtain

$$|\tau_{i,T}| = h^2 |\tau_{i,\text{ref}}|, \tag{3.5.36}$$

as well as

$$\int_{\tau_{i,T}} w_{j,T}(x)\, \mathrm{d}x = h^2 \int_{\tau_{i,\text{ref}}} w_{j,T}\left(\frac{1}{h}Rx + b\right) \mathrm{d}x = h^2 \int_{\tau_{i,\text{ref}}} w_j(x)\, \mathrm{d}x. \tag{3.5.37}$$

This shows that $\mathcal{P}_{\text{lim}}$ is the same matrix for all cells (and all variables).

We will write $\bar{U}_{i,T,n}$ for the value in patch cell $\tau_{i,T}$ and timestep n.

When to Activate the Limiter

For the limiter, we have several situations when it is activated (as done in e.g. [37]):

1. We encounter positivity-violating and wet-dry situations, i.e. when we have some $\tau_{i,T}$ where $h < 0$ or $C < 0$.
2. The predictor iteration does not converge on T.
3. If the two previous situations are encountered for a neighboring cell. This is checked, after the predictor is run, and it is done in order to facilitate the transition between DG and finite volume cells.
4. The Discrete Maximum Principle (Definition 3.1) is violated [12].

The last step is included for scenarios, when for example a shock runs through T. For this, we employ an a posteriori finite volume limiter as described in [12]. It is executed after we completed the corrector which gives us U^{n+1}. Then, we run the limiter which checks a variant of Definition 3.1 (defined below). If U^{n+1} passes, we go on to the next step. If U^{n+1} fails to pass, we resort to the finite volume limiter, but we take the averages computed from U^n instead of U^{n+1}.

The limiter itself checks the discrete maximum principle.

Definition 3.1 (Discrete Maximum Principle [12])**.** $\mathbf{U}_{n+1}$ *fulfills the discrete maximum principle, if for a fixed* δ*, it holds for all* $x \in T$ *for each triangle* T

$$-\delta + \min_{y \in \mathcal{V}(T)} \mathbf{U}_n(y) \leq \mathbf{U}_{n+1}(x) \leq \delta + \max_{y \in \mathcal{V}(T)} \mathbf{U}_n(y). \tag{3.5.38}$$

The minimum and maximum is taken over each component (i.e. h, hu etc.) of $\mathbf{U}_h$ *separately.*

In [12], δ is chosen as

$$\delta = \max\left\{\delta_0, \epsilon_0\left(\max_{y\in\mathcal{V}(T)} \mathbf{U}_n(y) - \min_{y\in\mathcal{V}(T)} \mathbf{U}_n(y)\right)\right\}. \tag{3.5.39}$$

Here $\delta_0 > 0$ and $\epsilon_0 > 0$ are two given parameters which are set to $\delta_0 = 10^{-4}$ and $\epsilon_0 = 10^{-3}$ in [12]. In addition, $\mathcal{V}(T)$ denotes the Voronoi region around T which is the area of all triangles which share at least one vertex with T. We can write it as

$$\mathcal{V}(T) = \left\{x \in \Omega \;\middle|\; x \in \mathrm{cl}(\hat{T}), \mathrm{cl}(\hat{T}) \cap \mathrm{cl}(T) \neq \emptyset, \hat{T} \in \mathcal{T}\right\}. \tag{3.5.40}$$

In practice, it is difficult to check Definition (3.1) over $\mathbf{U}_n$ and $\mathbf{U}_{n+1}$. Hence, we will resort to checking it for all $\bar{U}_{i,T,n}$ only. Thus, we obtain

$$-\delta + \min_{(i,T')\in\mathcal{V}'(T)} \bar{U}_{i,T',n} \leq \min_{i=1,\dots,k} \bar{U}_{i,T,n+1}, \tag{3.5.41}$$

as well as

$$\delta + \max_{(i,T')\in\mathcal{V}'(T)} \bar{U}_{i,T',n} \geq \max_{i=1,\dots,k} \bar{U}_{i,T,n+1}. \tag{3.5.42}$$

Here, $\mathcal{V}'(T)$ is the Voronoi region of the average cells.

After the limiter has been run, we check, if a cell is dry (or $C < 0$). If it is not dry, we run the predictor iteration in the next step. If it is dry, then we ignore the predictor.

3.5.4 Projection Initial Conditions

For the initial conditions, we interpolate over an equidistant grid. That is, we take the points $x^{i,j}$ for $0 \leq i + j \leq N$, defined as

$$x^{i,j} = \frac{1}{N}\begin{pmatrix} i \\ j \end{pmatrix} \tag{3.5.43}$$

on the reference cell. We want there to be exactly as many points as basis functions. Suppose then that we flatten the index (i, j) to $k = 1, ..., b_C$. Then, we once again compute a collocation matrix $\mathcal{P}_{\text{init}} \in \mathbb{R}^{b_C \times b_C}$ as

$$(\mathcal{P}_{\text{init}})_{ik} = w_i(x^k). \tag{3.5.44}$$

Once again, this matrix yields the same functions w_i for all triangles, given that we consider the points $P_{h,R,b}(x^k)$ for $T \in \mathcal{T}_{\text{DG}}$.

3.5.5 Timestepping

Finally, we decide about the timestepping for the ADER-DG and the finite volume method. For a given $T \in \mathcal{T}_{\text{DG}}$, as mentioned in papers like [12, eq. (17)], one can simply take the estimate

$$\Delta t < \frac{h}{(2N+1)d\,\|\lambda_{\max}(\mathbf{U}_{n+1})\|_{\infty,T}}, \tag{3.5.45}$$

where h is the edge length of the triangle, $d = 2$ is the dimension in space, N_P is the polynomial degree in space, and $\|\lambda_{\max}(\mathbf{U}_{n+1})\|_{\infty,T}$ is the maximum eigenvalue of $\mathbf{A}$ of the computed $\mathbf{U}_{n+1}$ on T. In particular, $\|\cdot\|_{\infty,T}$ denotes the maximum norm on T. Since it is difficult to actually compute $\|\lambda_{\max}(\mathbf{U}_{n+1})\|_{\infty,T}$, we resort to computing the maximum eigenvalue over all finite volume patches, as it is done in [37]. This means that we compute for $T \in \mathcal{T}_{\mathrm{DG}}$ the value

$$\|\lambda_{\max}(\mathbf{U}_{n+1})\|_{\infty,T} \approx \max_{\tau_i \in \mathcal{T}_{\mathrm{FV}}, \tau_i \subseteq T} \left|\lambda_{\max}(\bar{U}_{i,T,n+1})\right|. \tag{3.5.46}$$

This however does not work well, if $\mathbf{U}_{n+1}$ has a high total variation. In this case, we adjust the CFL condition, and we will not handle better approximations here (e.g. evaluation e.g. at some nodes on the triangle).

Source-Involved Timestep Estimates

In practice, this timestep has shown to be insufficient at least for small mesh sizes. The reason for this is, as we conjecture, the influence of $\mathbf{S}$. In particular, the last component reads $-2c^2 w$. Thus, if we have a large c, this could cause the solution to diverge. [7] While it is very well possible to just hide this using the CFL condition, we also derived a heuristic formulation. That is, we choose the timestep

$$\Delta t < t_{\mathrm{DG}}(T) := \frac{1}{\|\mathcal{K}^{-1}\|_2 \left(h^{-1} \|\mathcal{Z}\|_2 \left(\max_{\tau \in \mathcal{T}_{\mathrm{FV}}, \tau \subseteq T} \left|\lambda_{\max}(\bar{U}_{i,T,n+1})\right|\right) + \max\{2c^2, \gamma\}\right)}. \tag{3.5.47}$$

Here $\mathcal{K}$ is the same matrix as in the discretization of the predictor which is defined in (3.5.8). The matrix $\mathcal{Z}$ is given by

$$(\mathcal{Z})_{ij} = \sqrt{2} \int_{I_{\mathrm{ref}} \times T_{\mathrm{ref}}} v_i \partial_x v_j \, \mathrm{d}(t, x). \tag{3.5.48}$$

Here, $v_i, v_j \in B_P$. It is used as a rough estimate for the matrix which relates the basis functions from b_P to their derivatives. We get that $\left\|\mathcal{K}^{-1}\right\|_2 \|\mathcal{Z}\|_2$ is roughly as big as $(2(2N+1))$ for triangles. For example, for $N_P = 4$, we obtained $\left\|\mathcal{K}^{-1}\right\|_2 \|\mathcal{Z}\|_2 \approx 19.7$ compared to 18. The value $\max\{2c^2, \gamma\}$ is meant as a rough approximation to $\mathbf{S}$. It should be noted that for $h \to 0$, the dependence on the new term disappears or at least becomes irrelevant.

The idea for this stems from the conjecture that the Picard-Lindelöf can be used to prove the convergence of the predictor step, generalizing the result from [4], so that $\mathbf{A}$ does not need to be defined on the whole space. In addition, we would be able to remove the dependence on a Lipschitz constant, and obtain a sufficient estimate for the convergence of a single predictor step. If we simplify and approximate this sufficient estimate, we conjecture that we can arrive at (3.5.47). In practice, this timestep has shown to work in most cases, if the water is sufficiently deep and the solution sufficiently smooth.

[7] Large in the sense of $\alpha = 5$, $g = 9.81$, and $H = 1$, i.e. $c \approx 15.66$.

Combining the Timesteps

If the limiter is activated, we just take the finite volume timestep (and discard the DG timestep estimate). In particular, we get

$$\Delta t < t_{\mathrm{FV}}(T) := \frac{h}{2\sqrt{2}\left(\max_{\tau_i \in \mathcal{T}_{\mathrm{FV}}, \tau_i \subseteq T} \left|\lambda_{\max}(\bar{U}_{i,T,n+1})\right|\right)}. \tag{3.5.49}$$

In total, we can define the timestep estimate for a cell $T \in \mathcal{T}_{\mathrm{DG}}$ as

$$t_{\mathrm{est}}(T) = \begin{cases} t_{\mathrm{FV}}(T) & T \text{ is limited} \\ t_{\mathrm{DG}}(T) & T \text{ is unlimited.} \end{cases} \tag{3.5.50}$$

And so, we obtain the estimate over the whole grid $\mathcal{T}_{\mathrm{DG}}$ by

$$t_{\mathrm{est}}(\mathcal{T}_{\mathrm{DG}}) = \min_{T \in \mathcal{T}_{\mathrm{DG}}} t_{\mathrm{est}}(T). \tag{3.5.51}$$

Thus, with $C_{\mathrm{CFL}} \in (0, 1)$ being the CFL number, we get the timestep estimate for the next step as

$$\Delta t = C_{\mathrm{CFL}} t_{\mathrm{est}}(\mathcal{T}_{\mathrm{DG}}). \tag{3.5.52}$$

4 Implementation in sam(oa)²

After we described the numerical method we want to use, we now present our implementation of the scheme in the sam(oa)² software [32]. Its repository can be found at [40]. Our contributions lie in the `dswe` branch of the repository. The algorithms which we implemented were already described in the last chapter. The purpose of this chapter is to detail how we implemented the equations.

4.1 An Introduction to sam(oa)²

sam(oa)² is an acronym for "Space-Filling Curves and Adaptive Meshes for Oceanic And Other Applications" [32]. It is mostly written in FORTRAN90 or higher, and it implements a two-dimensional, structured grid made out of triangles (from now on termed "cells"). The grid is based on a space-filling curve [1] which can be refined at will. In particular, cells can be split or merged whenever needed. In addition, has integrated support for MPI and OpenMP, and load balancing with respect to the mesh refinement. The main operation which sam(oa)² is the traversal over such grid spanned by the space-filling curve. Therefore, we see and process a single cell, a single edge, or a single vertex (also termed node) at a time and need to formulate our algorithms in this way. This section describes briefly how to work with sam(oa)².

4.1.1 General Structure of sam(oa)²

sam(oa)² uses scons [19] as build system. The `src/` folder contains the source code. In `src/` itself, the basic files and tools for traversals are included. The indivdual implementations of the different applications (e.g. PDEs like the heat equation, the shallow water equation etc.) are placed into their own folders inside `src/`. For example, the finite volume implementation of the shallow water equations is placed in `src/SWE`.

To compile a particular equation (termed "scenario" is sam(oa)²), we have to supply its name to scons using the `scenario` parameter. For the shallow water equations, we subsequently have to call scons with `scenario=swe`. As for the `master` branch (and also on the `dswe` branch), there are currently only two `SConscript` files for all scenarios together. In addition, the build system currently compiles all files for all scenarios, even though they are not needed for all others.

Additional scripts are placed into the `scripts/` folder. Furthermore, the `.gitignore` script reserves the folders `bin/` (which is used by default by scons to build), and `output/` for output files.

Running sam(oa)² with MPI and OpenMP follows standard procedures,[1] and it is also described in [41].

Include Process

We will shortly describe the include process which is used in sam(oa)². Firstly, each file includes the `Compilation_control.f90` file which defines only C-preprocessor style macros. Subsequently, macros or files containing macros have to be included into `Compilation_control.f90`. Secondly, data types relevant for the traversals like data which should reside on nodes, edges, or cells are included via the file `SFC_data_types.f90`. It is also C-preprocessor style includes[2] `Compilation_control.f90` into `SFC_data_types`. Thus, each scenario includes a FORTRAN module which implements its specific data types. The modules which define the data types must not include files which contain traversals as to avoid a cyclic dependency. Lastly, we have all other code files. These C-preprocessor style include `Compilation_control.f90` and contain a `use Samoa` statement inside every module. The `Samoa` module (defined in the file `src/Samoa/Samoa`) includes the `SFC_data_types` module, i.e. a `use SFC_data_types` statement.

Program Execution Process

The FORTRAN main program is defined in the file `src/SFC_main.f90`. First, it reads the given input arguments from the `src/Config.f90` file. Then, it calls the subroutine `sfc_generic()` from `src/SFC_traversal.f90` which finally calls the scenario-specific code.

4.1.2 The Mesh

Next, we describe the mesh that is used in sam(oa)². It is built out of the triangles which we used in the last chapter, and finer meshes are created by splitting a triangle in two parts which equals an additional refinement in the space-filling curve. We define the subdivision factor N is a parameter which shows how often the space-filling curve is applied. Figure 4.1 shows the lowest-order subdivisions which sam(oa)² produces, without further adaptive mesh refinement. If we look at the even subdivisions, we can see that for $(N+2)$, each square (made out of two triangles) has been replaced by four squares itself, thus halving the side lengths. The same is true for the odd subdivisions, except that squares are made out of four triangles there. We can therefore give the following formula to obtain the number of cells n on one side, based on the subdivisions. It reads

$$n(N) = \begin{cases} 2^{\frac{N}{2}} & N \text{ is even} \\ 2^{\frac{N-1}{2}} & N \text{ is odd} \end{cases}. \tag{4.1.1}$$

[1] That is: all OpenMP parameters are available, and to use MPI, we simply need to run sam(oa)² using `mpirun` with the desired parameters.

[2] That means: `#include "Compilation_control.90"`

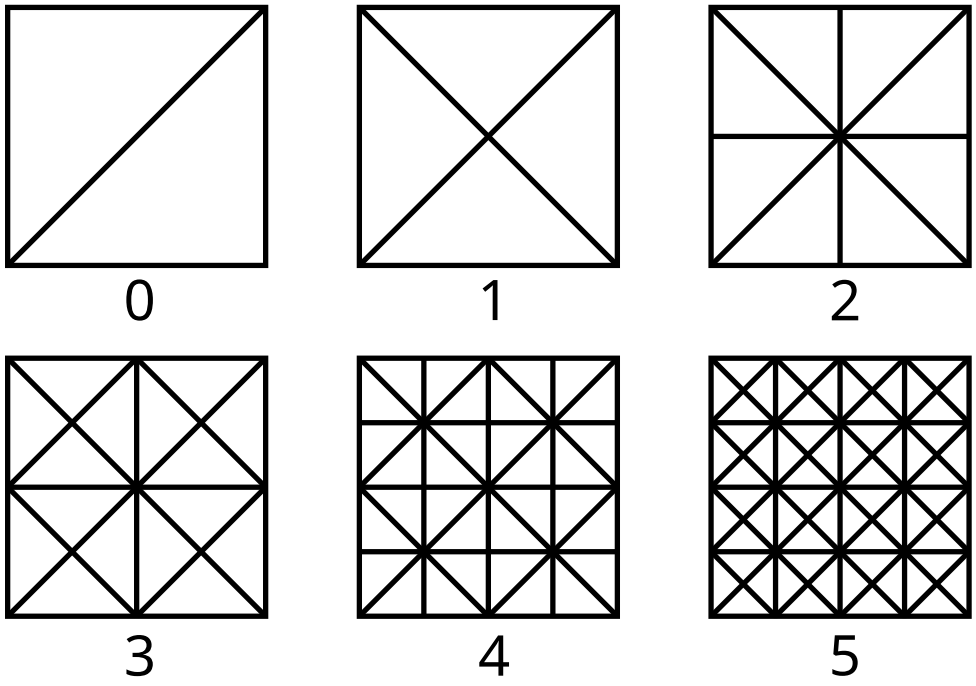


Figure 4.1: The mesh subdivision in sam(oa)² for $N = 0$ to $N = 5$.

In addition, we see that in each step (from even to odd and from odd to even), the cathetes become hypotenuses, and the hypotenuses are split in half. Thus, if $h(N)$ is the maximum edge length for subdivision N, and given $h(0) = h_0$, then we can use the recursion $h(N+1) = h(N)/\sqrt{2}$, and obtain that

$$h(N) = h_0 2^{-\frac{N}{2}} \tag{4.1.2}$$

is the mesh diameter (the smallest edge length), if we ignore the adaptive mesh refinement.

4.1.3 The SFC Traversals

A traversal over a grid is implemented in FORTRAN module which contains a `use Samoa` statement. In addition, it has:

1. Data structures of permanent and temporary data. These are usually mostly defined in a file which is included in `SFC_data_types`, but they may also be defined directly in the model instead. The needed data structures have the prefix `num_`, and most of the time, only `num_traversal_data` is defined anew for each traversal.

2. Macro directives which define the name of the resulting traversal type using the macro `_GT_NAME`, as well as macros which enable extra features as needed (e.g. `_GT_EDGES` to include the handling of edges), and the names of functions which handle the visiting of a node, an edge, or a cell (e.g. `_GT_CELL_UPDATE_OP` to update the data of each cell during a traversal).

3. Finally, the C-preprocessor style inclusion of the file `SFC_generic_traversal_ringbuffer.f90` for a static traversal or `SFC_generic_adaption_traversal.f90` for an adaptive traversal.

For the algorithms in sam(oa)², we refer to [32].

The following two paragraphs give a short overview over the traversals, but they are by no means exhaustive. In addition, they focus on the current state of implementation on the `master` and `dswe` branch only.[3]

Static Traversals

The traversal over cells is enabled by default. To enable the handling of edges, we additionally need to define `_GT_EDGES`, and for nodes, we need to define `_GT_NODES`.

Interaction between nodes, edges, and cells is done via the operators `_GT_CELL_UPDATE_OP`, and `_GT_ELEMENT_OP`. `_GT_ELEMENT_OP` has access to the data of all nodes, edges, and cells, and can update them all as well. `_GT_CELL_UPDATE_OP` has only access to the cell, and to skeleton update data. The skeleton update data is only available if `_GT_SKELETON` is defined, and it works as follows: first of all, `_GT_CELL_TO_EDGE_OP` is called for each edge (before and after `_GT_CELL_UPDATE_OP`) to set the `num_cell_rep` data structure. Then, the skeleton operators `_GT_SKELETON_OP` and `_GT_BND_SKELETON_OP` are called on each edge with the projections from `_GT_CELL_TO_EDGE_OP`, and they compute a result in form of the data structure `num_cell_update`. This is then passed to the `_GT_CELL_UPDATE_OP` which has access to the cell update data structures from all three edges.

For cells, edges, and nodes, we have additionally the hooks `FIRST_TOUCH`, and `LAST_TOUCH`.[4] These are, as their names suggest, called on the first occasion of meeting the respective simplex, and on the last occasion of meeting the respective simplex during the traversal. For MPI handling, both nodes and edges have extra hooks for the MPI data type (`MPI_TYPE_OP`), writing the data to MPI (`WRITE_OP`), and combining data from two different MPI results (`MERGE_OP`).

Furthermore, there are hooks for operations which happen before and after the complete traversal.

[3] For the `dswe` branch, a commit towards near the completion of this thesis is found at the hash `b41d46e4288c2902198d689e3ffe2728604b0509`.

[4] There is also `REDUCE` which is in all cases called directly after the `LAST_TOUCH` hook with the same arguments. Thus, it can be treated as equivalent to the `LAST_TOUCH` hook, and does not need to be considered.

Adaption Traversals

The adaption traversal allows refining or coarsen the grid as needed. To determine what to do for which cell is told by a refinement parameter which is given for each cell, and it can be set during the traversals. Depending on this parameter, one of the three extra refinement hooks is called. If the refinement parameter is 0, then `_GT_TRANSFER_OP` is called for this cell which means that the cell will be simply transferred to the new grid. In case the refinement parameter is negative, then `_GT_COARSEN_OP` is called, and the cell is merged with one of its neighbors. If the refinement parameter is positive, the operation `_GT_REFINE_OP` is called for each new subcell. Each cell can only be split into at maximum four parts or merged with one of its neighbors in a single adaptive traversal. For further refinement or coarsening, additional traversals are needed.

4.2 Implementation of the Governing Equations

Finally, we describe our implementation for the system (2.0.10) which we generally refer to as "DSWE" (short for "Dispersive Shallow Water Equations"). We heavily use the matrices we defined in the previous chapter (Chapter 3).

The implementation is located in the `dswe` branch, and there it can be found in the `src/ADERDG` and the `src/ADERDG/DSWE` folder. The sam(oa)^2 scenario is called `aderdg_dswe`.

In general, our implementation is constructed so that we can add any other shallow-water-like equation system, without having to modify any sam(oa)^2-specific traversal logic. For this, we encapsuled all equation system-specific logic into subroutines which are placed inside the `src/ADERDG/DSWE` folder for the H-BMSS-γ system. Generic traversal and sam(oa)^2-specific logic which is the same for all equations is placed into the `src/ADERDG` folder directly.

Thus, to implement a system, we need to specify some macros, data types, and functions only, but we do not need to deal with any traversal-related issues. The most important macros as the number of variables (`_ADERDG_Q_VARS`), the number of constant variables (`_ADERDG_AUX_VARS`). In our case, we set these parameters to 5 and 1; `Q` gets h, hu, hv, hw, hp, and `AUX` gets b.

Note however that in our implementation, no matrix logic has been abstracted away so far. Therefore, any new equation system that is added has to implement the predictor iteration completely from scratch, unless we re-use existing code. This is because we could not exclude the case that we might want to change the basis for one of the variables in the future.

In the following, all methods that are prefixes with `ADERDG_Impl_` can be found in `src/ADERDG/DSWE/Impl`.

4.2.1 Scenarios

We have a scenario file in `src/DSWE/DSWE_Scenario.f90` which contains the settings for the code which we are going to run. In general, the scenario is hard-coded into the

executable at compile time. It can be used for both initial conditions, and also prescribed scenario boundary conditions (cf. Table 3.2) which can be different for each point and time. The scenario is supplied over the compile-time parameter `swe_scenario`. A list of scenarios can be found in Table 5.1 in the next chapter (Chapter 5).

4.2.2 ADER-DG Implementation

Conceptually, our implementation is similar to the existing ADER-DG implementation for the shallow water equations [37]. That is we use two traversals, one for the predictor, and one for the corrector.

ADER-DG Loop

We begin by describing only the ADER-DG part for the traversals. We use two traversals per step, and an additional traversal for refinement.[5] After the first traversal, the projection on the edges are available everywhere by the skeleton operators. After the second traversal, the timestep for the next iteration is reduced globally.

Predictor Traversal The first traversal (`src/ADERDG/ADERDG_predictor.f90`) computes the predictor, and prepares the local part of the corrector. This means that on each cell, we give U^n, and get $U^{n+1,\text{local}}$ back. For that, internally the predictor iteration (3.5.16) is run. We copied the matrix evaluations from (3.5.16) directly into the FORTRAN code, using the `matmul` operation. The nonlinear quadrature point evaluation was done using a function which was marked as `elemental`, i.e. it is applied element-wise to a vector we input, and we are able to input vectors of any size. The computation of $U^{n+1,\text{local}}$ from Q^n also utilizes these ideas. In addition, this traversal computes the projections on the edges $\mathcal{F}^k Q^n$ for $k = 1, 2, 3$ and each element. All these computations happen in the `ADERDG_Impl_Predictor` subroutine which is called in the `_GT_ELEMENT_OP` hook. In addition, the values $\mathcal{F}^k Q^n$ are sent to the edges by the `_GT_CELL_TO_EDGE_OP` hook. As a result, the predictor is in practice the compute-intensive operation, as it was hinted at in the last Chapter.

The wave-breaking switch is only enabled and disabled by checking all quadrature points of the derivatives of the basis functions. In addition, its state is only changed at the beginning of the predictor iteration. This is to prevent that the iteration diverges, since the switch is enabled and disabled multiple times during the iteration.

Corrector Traversal The second traversal (`src/ADERDG/ADERDG_corrector.f90`) exchanges the boundary quadrature values, and completes the corrector. In the end, we obtain U^{n+1}.

For this, the skeleton hooks are used. Since it has for each edge F the quadrature points evaluated from both sides, write Q^+ and Q^-, we compute $\mathbf{D}(Q^-, Q^+) \cdot n$ and

[5] The refinement traversal can also be merged into the predictor traversal, see [37]. There are even more compact schemes, see e.g. [7], or [2].

$\mathbf{D}(Q^+, Q^-) \cdot (-n)$ for each quadrature point. This is done in `ADERDG_Impl_inner_edge`. Before and after $\mathbf{D}$ is applied, one of Q^+ and Q^- has to be reversed in space. If F lies on the boundary, we insert one of the boundary conditions which is mentioned in Table 3.2. Which condition to choose is supplied as the command line argument `dswe_boundary_condition`. The choice is then propagated over C-preprocessor style macros to `ADERDG_Impl_boundary_edge` which also computes one of $\mathbf{D}(Q^+, Q^-) \cdot n$ and $\mathbf{D}(Q^-, Q^+) \cdot (-n)$ after inserting the boundary value. We do not compute both fluxes here anymore. The flux which is used here can be decided over the command line argument `dswe_dg_flux_solver` which accepts all arguments as presented in Table 3.1.

Finally, the corrector is completed over the `_GT_CELL_UPDATE_OP` hook which gets the fluxes which were computed over the quadrature points in the skeleton part. The corrector is completed in `ADERDG_Impl_corrector_complete` which essentially applies $\mathcal{M}^{-1}\mathcal{G}^k$ to all computed flux quadrature points to obtain $U^{n+1,\text{edge}}$. By that, we finally compute $U^{n+1} = U^n + U^{n+1,\text{local}} + U^{n+1,\text{edge}}$.

The corrector traversal also computes the timestep per cell and takes the minimum over all cells of it, so that it can be used in the next predictor step.

Adapt Traversal Additionally, we have implemented a refinement traversal (located in `src/ADERDG/ADERDG_adapt.f90`) for the system. It requires the implementation of the split subroutine `ADERDG_Impl_adapt_split_cell`, and merge subroutine `ADERDG_Impl_adapt_merge_cells`. In addition, `ADERDG_Impl_corrector_complete` can specify whether we should refine or coarsen a cell. However, for our current implementation, this traversal is currently not much in use.

Limiter Implementation

Next, we describe the addition to the limiter. Our implementation uses the generic finite volume implementation from [17] as a basis and utilizes a lot of code from it for the finite volume part. The interplay between finite volume and DG part is mostly taken from [37], and slightly abstracted.

Finite Volume Procedure Most of the finite volume step is handled in the corrector traversal. To begin with, we have the averages $\bar{U}_{i,T,n}$ from the previous time step. Using the `_GT_CELL_TO_EDGE_OP`, the indices i which border another cell are sent to the boundary. In the skeleton operators, the fluxes are *not* computed there already, so we can handle all flux computations in the `_GT_CELL_UPDATE_OP`. Only the boundary conditions are set in the skeleton operators, over the subroutine `ADERDG_Impl_FV_boundary` which makes use of the same boundary condition as the DG part.

The actual finite volume computations happen in the subroutines `ADERDG_Impl_FV_compute_flux` which computes the numerical fluxes for a specified number of edges. It can be controlled over the parameter `_ADERDG_FV_FLUX_CHUNK_SIZE`. This is the idea from [17], to chunk the flux computation in order to improve performance. After that, the source term is added in the method `ADERDG_Impl_FV_update_cells` which is called

for each finite volume cell. In addition, this cell also checks the positivity conditions (i.e. $h \geq 0$ and $C \geq 0$) and resets cells which violate them. This means, if $h < 0$, then all variables are set to 0, and if $C < 0$, then $p = -c^2 - gh$. The positivity checks are marked as `elemental`.

Projection between DG and Finite Volume Representations The projection from DG and to finite volume and vice-versa is handled over the methods `ADERDG_Impl_proj_from_subpatch`, `ADERDG_Impl_proj_to_subpatch`, and `ADERDG_Impl_proj_to_subpatch_permanent`. The latter is for the projection of b, while the former projects all other variables. Internally, they only apply $\mathcal{P}_{\text{lim}}$ or its Moore-Penrose inverse. In addition, `ADERDG_Impl_proj_to_subpatch` checks the positivity conditions and resets any cells which violate them.

DMP Checking After U^{n+1} has been computed, the discrete maximum principle is checked using $\bar{U}_{i,T,n+1}$. These values have then just been computed, and $\bar{U}_{i,T,n}$ from the previous step.

To compute a minimum and maximum over the Voronoi region, we do a reduction the observable values using the hooks for the nodes. This is because for $T \in \mathcal{T}_{\text{DG}}$, we have that $\mathcal{V}(T)$ contains exactly all triangles which share at least a vertex with T. In particular, we set initial values $-\infty$ for the maximum computation and ∞ for the minimum computation in `_GT_NODE_FIRST_TOUCH`. Subsequently, we compare the minimum and maximum against new values from a cell over the `_GT_ELEMENT_OP` operator. Over `_GT_NODE_MERGE`, two node values from different MPI implementations are combined.

Input and Output

The input is handled in a traversal on its own (`src/ADERDG/ADERDG_init.f90`). It is called in conjunction with the adapt traversal. Thus, we first refine the mesh, and then set the initial data. This is done through the methods `ADERDG_Impl_init` and `ADERDG_Impl_init_permanent`. The `ADERDG_Impl_init_permanent` subroutine sets up the bathymetry and is re-used in the adaptive traversal, and the method `ADERDG_Impl_init` sets up all other variables. Support for ASAGI exists in theory as it has jus been ported from `src/swe`, but it has not been used or tested so far.

As output, the implementation currently only supports VTK XML output, for which it has its own traversal. In general, we allow an arbitrary number of points and an arbitrary cell type to be used in the output which is given through `ADERDG_Impl_output_reference`. For our equations in particular, we implemented an arbitrary high-order output, but supporting the VTK cell type `VTK_LAGRANGE_TRIANGLE` (cell type 69). Effectively, it is a generalization of the `VTK_TRIANGLE` and `VTK_QUADRATIC_TRIANGLE` outputs. This requires a specific ordering of the points which we programmed in our Python script.

The output subroutine `ADERDG_Impl_output_labels` specifies the output labels and the data per cell. The subroutine `ADERDG_Impl_output_preprocess_data` converts the

cell data to output data the data per cell, respectively. We output all variables, and if the wave-breaking switch is active or not.

4.2.3 Generating Matrices

The matrices are generated in Python with the help of the sagemath toolkit [38]. As for the quadrature rules, we used the rules which the toolkit quadpy [39] suggested to us.

The function bases B_C and B_P were created from orthogonalizing monomials.

Given the bases, we have three basic matrix computation methods. The first one is the computation of a collocation matrix, potentially of the derivatives. That is, we for an operator D (e.g. $D = \mathrm{Id}$, $D = \partial_x$ or similar), the matrix M defined by

$$M_{ij} = w_j D(\phi_i)(p_j). \tag{4.2.1}$$

Here, w is a weight vector which can also be set to 1 everywhere to get an unweighted collocation matrix. (ϕ_i) is a set of basis functions. Using this structure, we are able to compute $\mathcal{C}$, $\mathcal{C}^x$, $\mathcal{C}^y$ (both for $R = I$), $\mathcal{D}$, $\mathcal{E}$, $\mathcal{E}^x$ and $\mathcal{E}^y$, as well as also $\mathcal{F}^k$ and $\mathcal{G}^k$. Also $\mathcal{P}_I$ can be computed in this way, and the same is true for the output matrix. The second method is for the computation of mass matrices, again potentially over derivatives. That is, we compute for two bases (ϕ_i) and (ψ_i) over some area A the matrix N as

$$N_{ij} = \int_A \phi_i(x)\psi_j(x)\,\mathrm{d}x. \tag{4.2.2}$$

This lets us compute $\mathcal{M}$, $\mathcal{K}$, and the matrix to obtain $U^{n,\mathrm{init}}$. Finally, as a third method, we have the limiter projection matrix $\mathcal{P}_{\mathrm{lim}}$ which needs an additional custom implementation.

In addition to the matrices, we decided to generate the adjacency data for the patches inside Python as well. This means that we generate the adjacency data for $\tau_{1,T}, ..., \tau_{k,T}$ for $T \in \mathcal{T}_{\mathrm{DG}}$. This was previously computed by [3] in FORTRAN in the file `src/Tools_patch.f90`. However, since we need to compute $\mathcal{P}_{\mathrm{lim}}$, it was less error-prone to precompute the adjacency data inside the Python file as well. We obtain arrays with the normals of the patches in T_{ref}, of the patch cells which are located at each edge, and which patches lie at the boundary. These then are relayed over the functions `ADERDG_Impl_subpatch_data` and `ADERDG_Impl_subpatch_border` which replace the functions from the `patch_geometry` class, i.e. the functions `get_edge` and `get_boundary_cell`, respectively.

4.2.4 Implementation of Numerical Fluxes and Quadrature Evaluation

Finally, we discuss the implementation of the numerical fluxes and the nonlinear parts of predictor and corrector.

To begin with, we implemented the model in Python. For that, we implemented the functions **F**, **B**, and **S** in Python using the sympy [33] and the sagemath [38] toolkit. From them, we could compute **A** automatically.

Next, we computed basic functions, as S_h^{-1}, the implicit Euler evaluation in the finite volume step, the matrix $|\mathbf{A}_1|$, and the Rusanov flux using sagemath and sympy, and generated FORTRAN code from it.[6] Further functions, like the HLL, DOT or Roe flux could be easily written manually with these basic kernels. All functions were marked as `elemental` in FORTRAN which allows us to insert vectors into them, and FORTRAN executes the function on each element in the input. For all numerical flux kernels, we made use of the rotational invariance, since the application of T and T^T could also be written into an elemental function. For the nonlinear parts in the predictor and corrector, we also generated code and marked the resulting functions as `elemental`.

All the PDE and flux-specific kernels can be found in the `src/DSWE` folder. This is, so that at least the fluxes can be re-used later for other potential Discontinuous Galerkin implementations.

4.3 Further Extensions

Besides the implementation of the equation system, we also created some small modifications and extensions to sam(oa)². To begin with, the `master` branch did not run with the gfortran compiler correctly any more due to some smaller syntax inconsistencies. After fixing these small errors, it did work again. And so, we could run all our tests using gfortran in the end.

Furthermore, we wrote a small VTK reader in Python which is able to read the `pvtu` and `vtu` files. It uses the existing `vtk` package, and wraps the data arrays which describe cells and points into Python arrays.

The scripts used in Chapter 2 to compute the Riemann Problem and the solitary wave are placed as Python scripts into `scripts/DSWE`.

[6] In the beginning, we tried to generate even more code, involving matrices using sympy—with the goal to be able to auto-generate the predictor as a whole from symbolic formulas. However, sympy did not seem to be yet ready for such operations which involve symbolic matrix multiplications inside code generation. We already started work on a small code generator which we used for generating the matrix data, but it cannot yet replace the sympy code generation, nor supports matrices.

5 Evaluation

Given our implementation as described in the previous two chapters (Chapters 3 and 4), we will now proceed to show some examples which we computed with our program. These should not only confirm the correctness of the program itself, but also of our calculations of the solitary wave and the Riemann problem solution from Chapter 2. All tests shown in this chapter are run in sam(oa)^2 only, and compared with our (reference) solutions which we computed in Python in Chapter 2.

Most of the time, we will show one-dimensional solutions which were extracted using our VTK reader which we programmed. It looks for cells which lie exactly on the $y = 0$-axis and interpolates through them with an N_P polynomial, where N_P is the polynomial degree which we used for the simulation. Since we lie at the boundary between two cells, we take the average of them. If we have symmetric initial conditions, this gives us an exact representation in the case of an even subdivision scheme. But it still introduces a slight error due to rotation, if we use an uneven subdivision level.

We always run the `aderdg_dswe` sam(oa)^2 scenario. For a list of `aderdg_dswe` scenarios, see Table 5.1. Mainly, the scenarios are concerned with the reproduction of the solitary wave and the Riemann Problem. In all cases, we will refer to the fluxes by their name from Table 3.1. Therefore, we write `llfb` for the Rusanov flux which preserves the Resting-Lake condition, `dot` for the DOT flux, and similar. The boundary conditions are also mentioned by their names from Table 3.2.

5.1 Test Setup

For all tests we showcase here, we used a simple laptop. Furthermore, the goal was to simply reproduce the equation system, not to test if it scales. The laptop has an AMD Ryzen 4800H CPU with 8 cores and 2 threads per core, making 16 logical cores in total. In addition, it has 32 GiB of RAM. As operating system, it uses Linux with the kernel in version `5.16.10-arch1-1`. More specifically, it is the Arch Linux distribution. As compiler, `gfortran` in version 11.2.0. was used.

Handle	Description
`LINEAR_DAM_BREAK`	Riemann Problem, Table 2.1 (a)
`PULLAWAY_STRONG`	Riemann Problem, Table 2.1 (b)
`COLLISION`	Riemann Problem, Table 2.1 (c)
`PRESSURE_EXCHANGE`	Riemann Problem, Table 2.1 (d)
`WETDRYFRONT`	Riemann Problem, Table 2.1 (e)
`SQUARE_DAM_BREAK`	Table 2.1 (a) in a square
`RESTING_LAKE_EMPTY`	Empty resting lake, $q = -1, r = 0$
`RESTING_LAKE_STILL`	Still resting lake, $q = 2, r = 0$
`RESTING_LAKE_EXCITED`	Still resting lake, $q = 2, r = 1$
`RESTING_LAKE_ISLAND`	Still resting lake, $q = 0.4, r = 0$
`SOLITARY_WAVE`	Solitary wave with $A = 0.01, \alpha = 5$
`SOLITARY_WAVE2`	Solitary wave with $A = 0.2, \alpha = 3$
`SOLITARY_WAVE_ON_BEACH`	Solitary wave $\alpha \to \infty$ against beach

Table 5.1: A list of all implemented scenarios which were tested. They were set over the compile-time parameter `swe_scenario`.

5.2 Well-Balancedness Tests

First of all, we test if our scheme fulfills the two steady-state solutions from Definition 2.2. For this, we use the following scenario:

$$b(x) = \frac{1}{4}\left(e^{-40\|x\|^2} - q\right), \tag{5.2.1}$$

$$h(0, x) = \max\{0, -b\} + \frac{r}{4}\sin(\|x\|^2). \tag{5.2.2}$$

Here, $q \in \mathbb{R}$ is a parameter which controls how submerged the bottom topology is. For $q > 1$, we have only water, for $q \in (0, 1]$, we have water and land, and for $q \leq 0$, we only have land. All other parameters are set to zero, therefore $u(0, \cdot) \equiv 0, v(0, \cdot) \equiv 0, w(0, \cdot) \equiv 0$, and $p(0, \cdot) \equiv 0$. The parameter $r \in \mathbb{R}$ adds some optional excitation for $r \neq 0$. This is order to test the stability of slight perturbations of the steady states.

5.2.1 Vacuum Scenario

The first test is with $q = -1, r = 0$, i.e. we test the case where we only have dry land. This is scenario `RESTING_LAKE_EMPTY` from Table 5.1. As expected, the simulation skips all cells as a result—yielding a perfectly dry solution for all fluxes. We note that this is not the case, if we did not skip all cells automatically—the `llfb` flux would generate water in this case. We simulated for $10s$.

Since the system was solved perfectly, we will not show any plots for this case.

5.2.2 Resting Lake Scenario

In the second test, we set $q = 2$ and $r = 0$, i.ewe look at a fully-submerged resting-lake scenario without excitation. This is scenario `RESTING_LAKE_STILL` from Table 5.1. We use subdivision factor 10 which yields 2048 cells, absorbing boundary conditions, and $c = a\sqrt{gH}$ with $a = 3$ and $H = 1$. We run the simulation for $100s$ and use a CFL of 0.5. As shown in Figure 5.1, the non-hydrostatic pressure p, as well as the water level η oscillate at around 10^{-14} and 10^{-15} respectively. However, w and $\|\mathbf{u}\|_2$ seem to increase over the course of time, both with a similar slope. Looking at the output at the end of the simulation, the plots for u, v, and w show a more or less random pattern (respecting rotational symmetry). The limiter never active during the entire simulation, and the DMP check was deactivated.

In order to further verify the Resting Lake behavior, we ran the same simulation with degree 5 polynomials in space and time, as well as subdivision level 6. The result after $100s$ is shown in Figure 5.2, and it shows very similar results to the simulation with polynomials of degree 2 in Figure 5.1. The only small differences are that w seems to be oscillating a bit in the maximum and minimum values, and h and p show to initially oscillate a little more, until they get into an equilibrium state. However, we noticed here a phenomenon which existed throughout all of our tests, that is that we have a larger error at the corners of the elements. This problem especially existed for the non-hydrostatic pressure component. In Figure 5.3, the variable hp is shown at $0.7s$. At all corners, we have small deviations from the equilibrium at 0, and the error depends on if the vertex has eight or four neighbors. After more timesteps, the oscillations became smaller.

5.2.3 Almost Resting Lake Scenario

Thirdly, we set $q = 2$ and $r = 1$, i.e. we look again at fully-submerged resting lake scenario, but this time, the water is not completely still. This is `RESTING_LAKE_EXCITED` from Table 5.1. This should test that the solution still stays bounded compared to the original. Otherwise, we use the same settings as for the resting lake scenario, except that we only run the simulation for $10s$. As it can be seen in Figure 5.4, the simulation stays bounded, and does not necessarily cause an increase in any variable: instead they simply all oscillate. Looking at the plot (Figure 5.5) shows that most of the excitation has an effect at the boundary. This is not surprising, since we force `absorbing` boundary conditions. However, we have a interesting effect, namely the peaks near the boundary in Figure 5.5. The peaks are probably again caused by the triangle topology, as the error is mostly at the vertices of the triangles again.

5.2.4 Resting-Lake with Island

Lastly, we set $q = 0.4$, so that we have an island in a resting lake. This is to test the interplay between the finite volume limiter and the DG simulation, as well as the bathymetry.

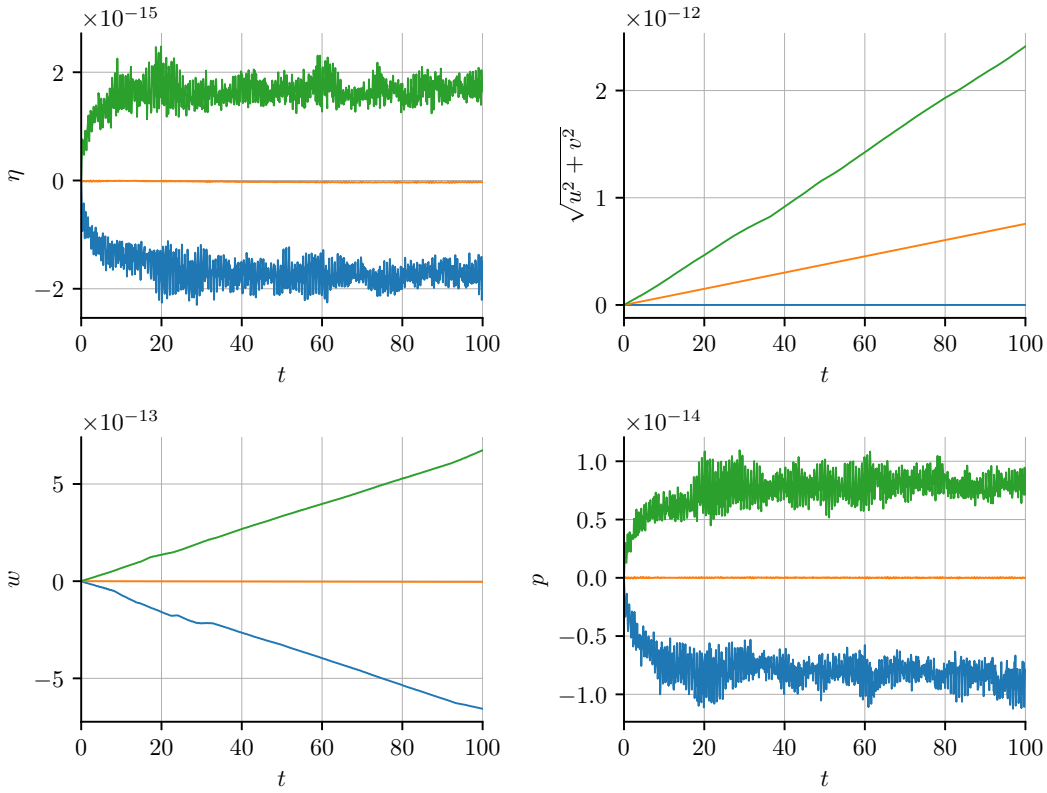


Figure 5.1: Minimum, average and maximum value of η, $\|u\|_2$, w, and p during the resting lake simulation.

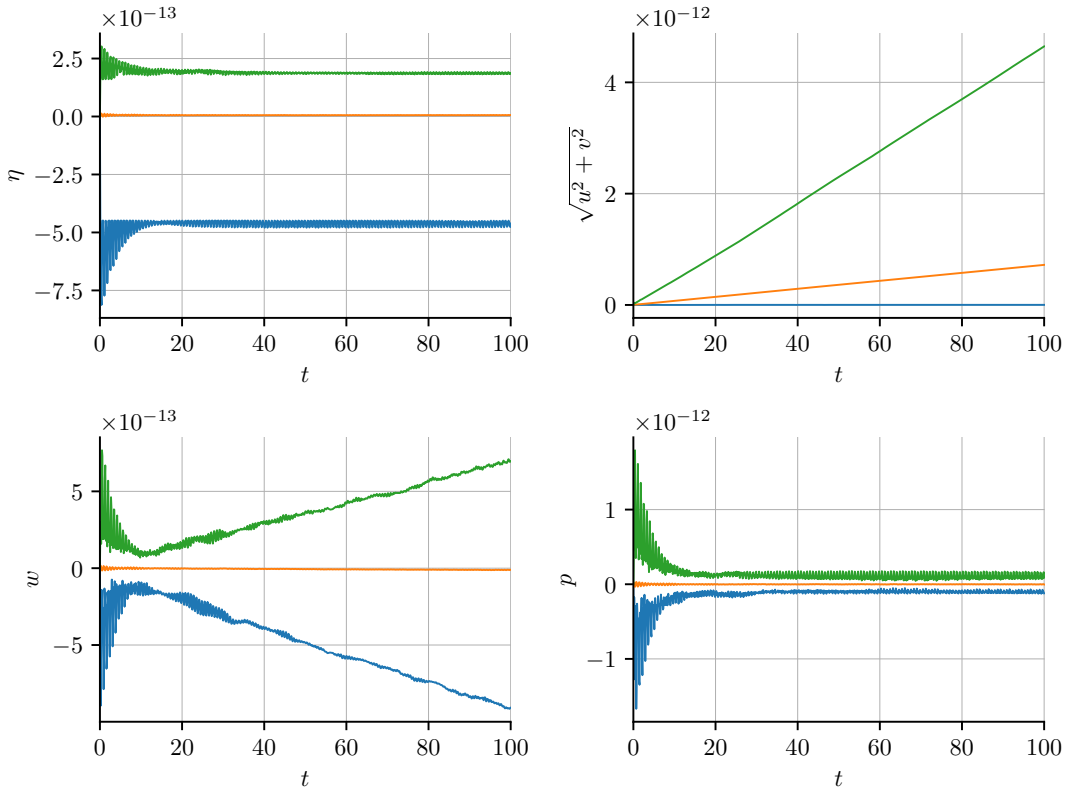


Figure 5.2: Minimum, average and maximum value of η, $\|u\|_2$, w, and p during the resting lake simulation.

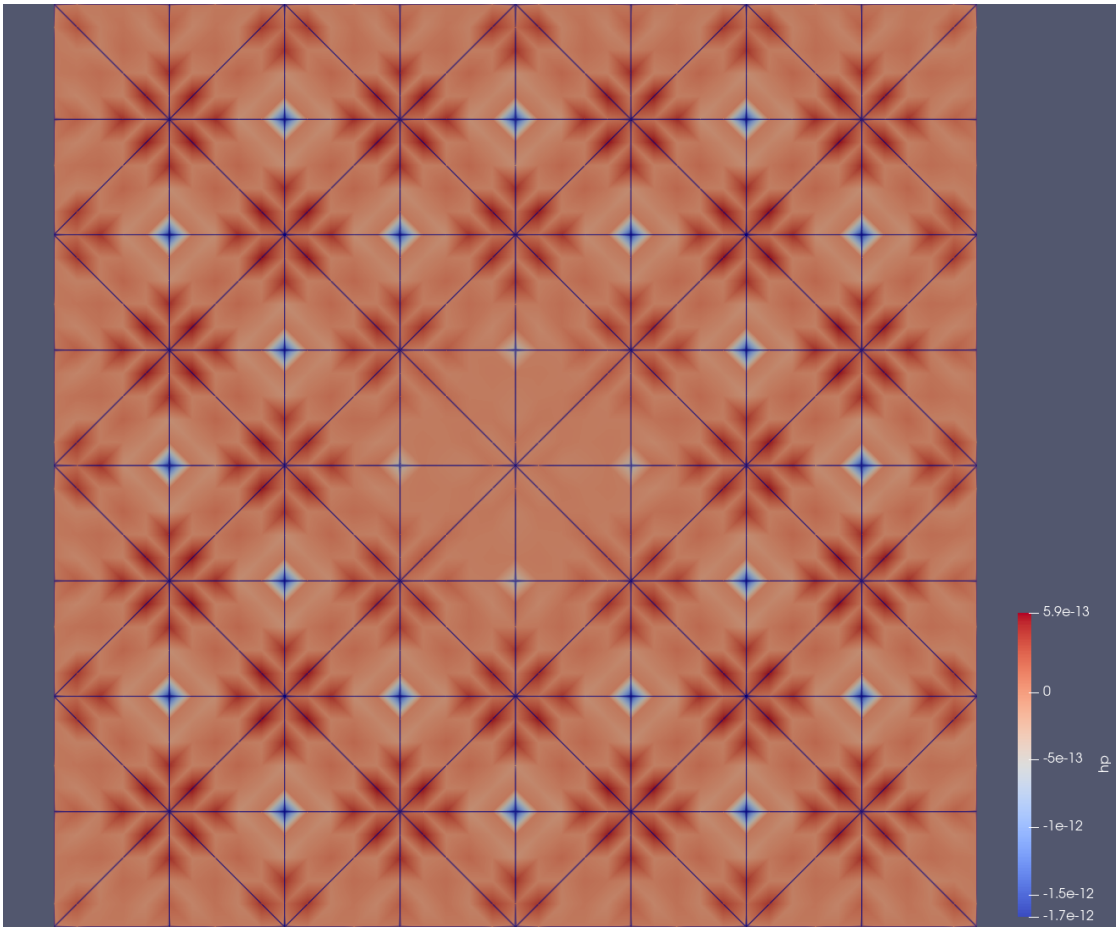


Figure 5.3: The behavior of hp on triangles. Blue means small ($\geq -10^{-12}$) negative values, and red means small ($\leq 10^{-12}$) positive values. The snapshot was taken at $0.7s$ in the simulation.

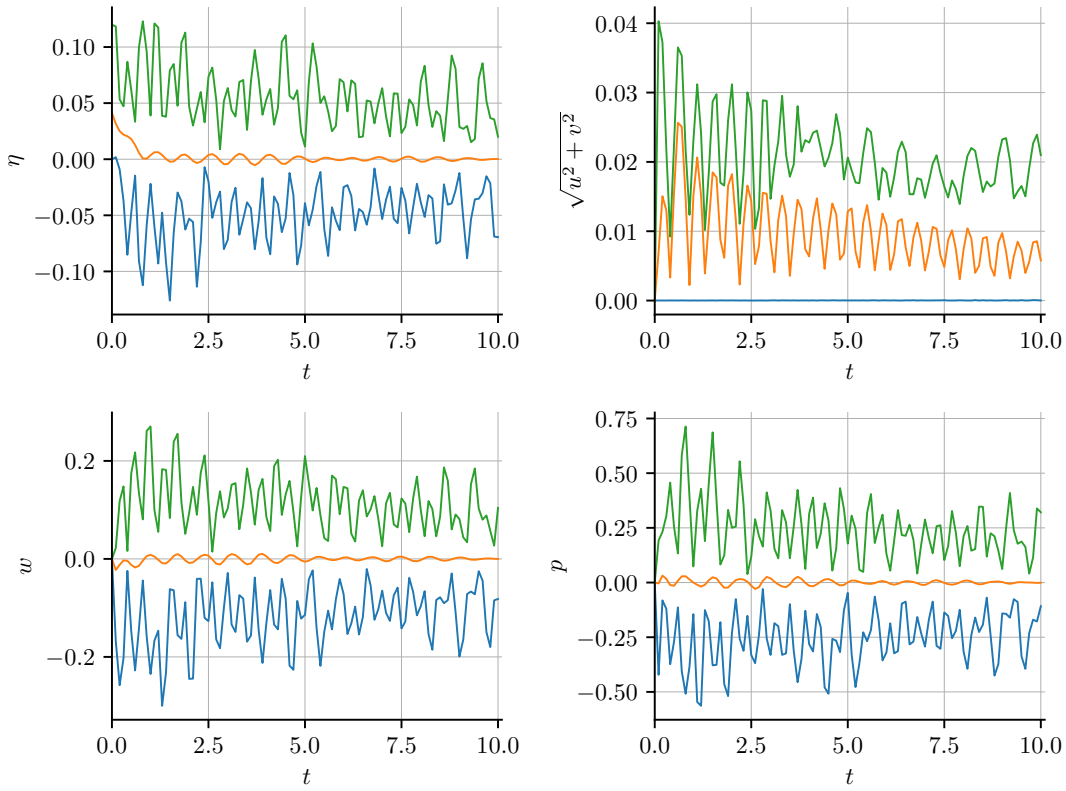


Figure 5.4: Minimum, average and maximum value of η, $\|u\|_2$, w, and p during the excited resting lake simulation.

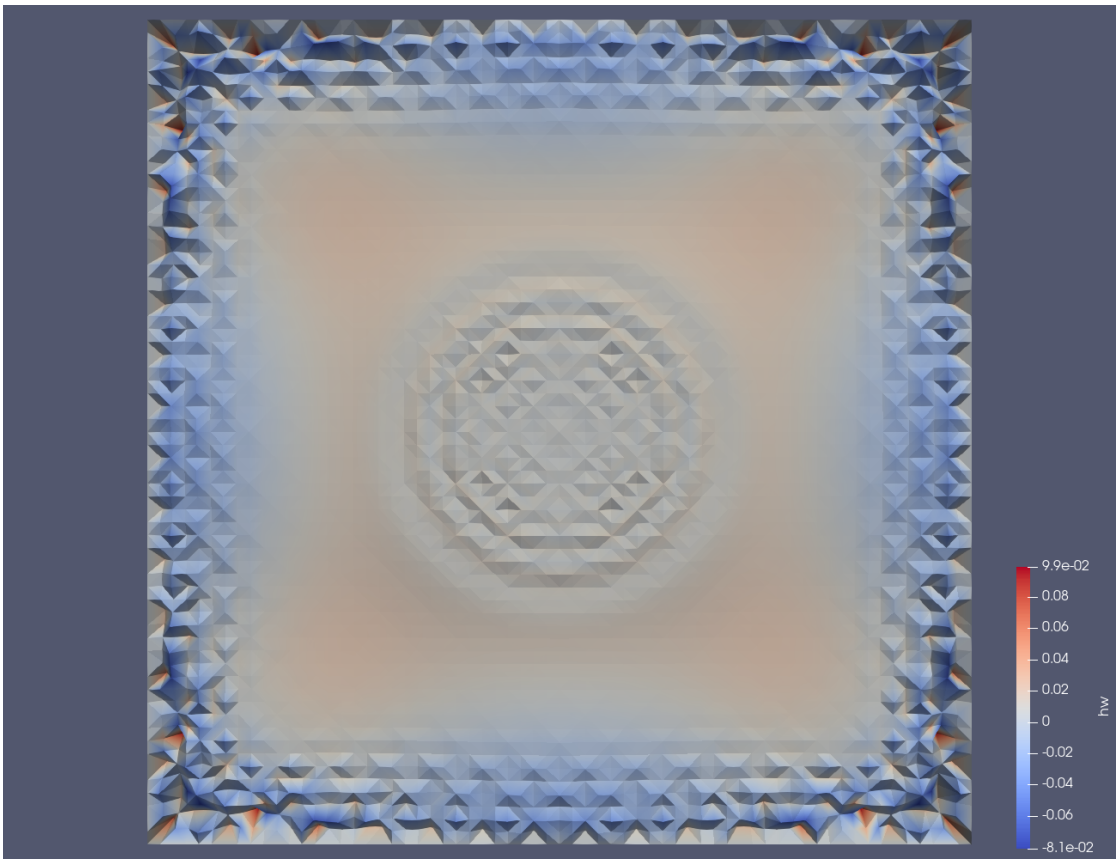


Figure 5.5: A plot of the water level at $10s$ into the almost resting lake simulation. The color indicates hw.

This test does unfortunately not work yet. The reason is that, no matter if we pick `dot` or `roe` as a flux, the pressure accumulates at the wet-dry fronts and subsequently forces an abrupt change in water height. We conjecture that this is due to the choice of linear or quadratic paths for these numerical fluxes, as seen in Table 3.1. To begin with, we have a projection error after the first timestep.

For all the following tests, we used a CFL of 0.8.

5.3 Solitary Wave

Next, after we have looked at the Resting Lake scenario, we begin with reproducing our two solutions for the equation system (2.0.10). That is, we start with the solitary wave. The goal here is not only to reproduce it, but also to do a convergence test with it in sam(oa)^2.

5.3.1 Running the Solitary Wave

The first test is to simply simulate the solitary wave for about 40 seconds. The scenario in sam(oa)^2 is called `SOLITARY_WAVE2`.

We run the solitary wave computed in Section 2.3 with $A = 0.2$, $H = 1$, $\alpha = 3$, and $\gamma = 2$, and compute it to an accuracy of $\Delta\xi = 10^{-3}$, and linear interpolation to approximate the function between. The domain is $160m$ long and wide, and the wave starts at $10m$ from the left-hand side. Then, it moves to the right, until it is exactly $10m$ away from the right-hand side—which is where we measure. This is done in order to capture the highest point at a place, where it sits exactly between two elements. As boundary conditions, we use the quasi-exact computed wave from Section 2.3, i.e. `scenario` boundary conditions. We use 14 as subdivision factor, i.e. we have 2^7 cells per side which gives us $\Delta x = 1.25m$.

This whole test is similar to what is done [15, Section 5.1.2]. In comparison to there, we use a different c, and we could not use periodic boundary conditions, since sam(oa)^2 does not support them.

See Figure 5.6 for an overview, where we plot on the left-hand side the solitary wave, as computed, overlayed over the reference solution. On the right-hand side, the difference between the two solutions can be seen. In total, the computed solution is very close to our computed quasi-exact compute wave from Section 2.3. The next thing we notice is the small visible error in w and p on the right-hand side. This is most likely due to an approximation error during the projection of the solitary wave to the grid in the beginning. However, such an error is also shown in [15, Figure 7] for p and slightly visible for [] the only difference is that it is present on the right-hand side. While the error is slightly higher in Figure 5.6 than what is shown in [15, Figure 7], this is most likely due to us using $\Delta x > 1m$. In [15, Section 5.1.2], it is $\Delta x = 1m$, and so, each element seems to cover a much smaller interval compared to our simulation. Furthermore, the simulation in [15] appears to happen in a single dimension only which means that a potential error from having a triangle mesh is not present.

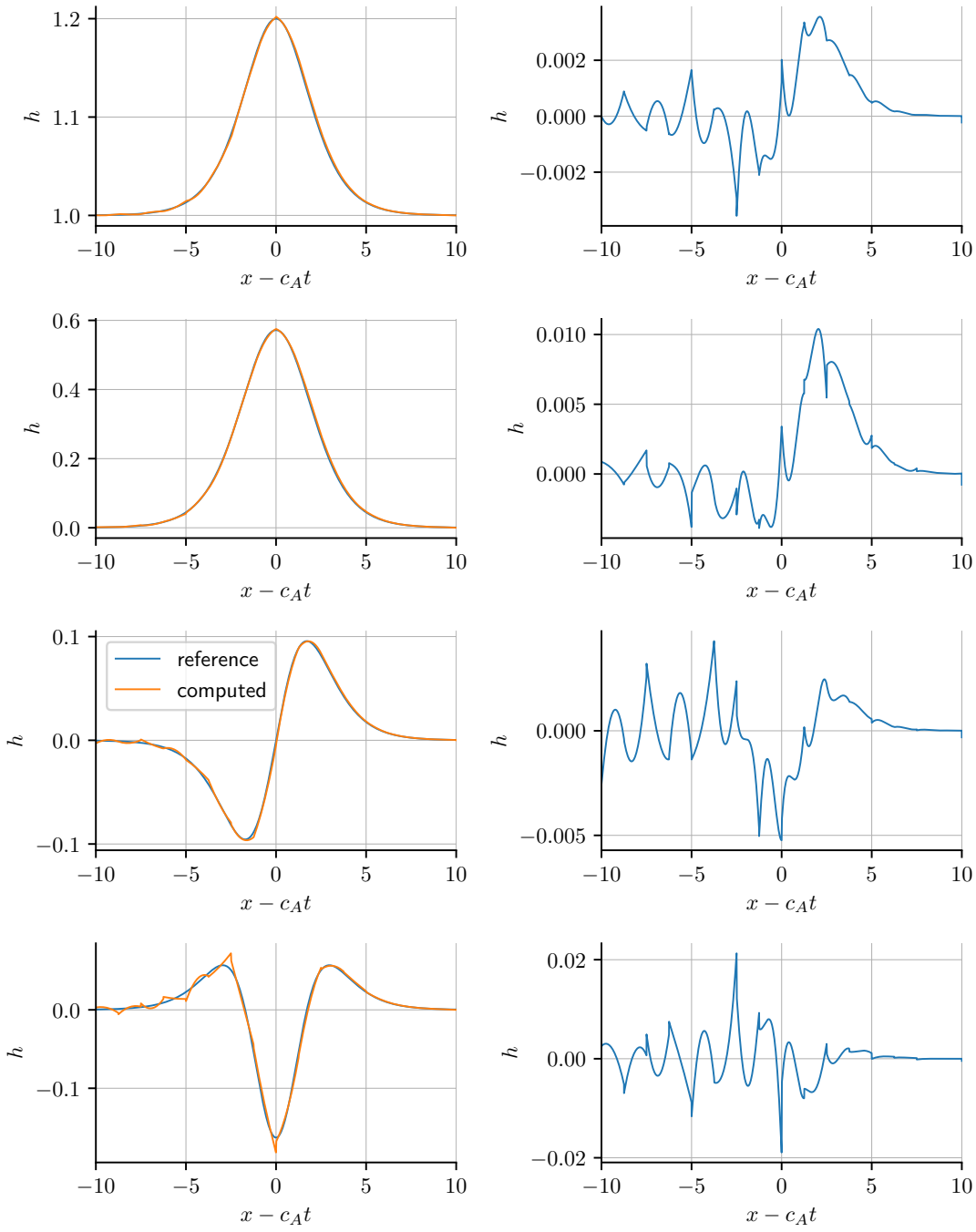


Figure 5.6: Reproduction of the solitary wave from Figure 2.1 in sam(oa)², and comparison with a simulation after $\approx 41s$ have passed.

5.3.2 Convergence Test in sam(oa)²

Next, we consider the convergence against the quasi-exact solitary wave solution from Section 2.3. We still run the solitary wave computed in Section 2.3 with $A = 0.01$ and $H = 1$, $\alpha = 5$, and $\gamma = 2$, but we simply stop after

$$\frac{50m}{c_A} \approx 15.88s \tag{5.3.1}$$

of runtime.

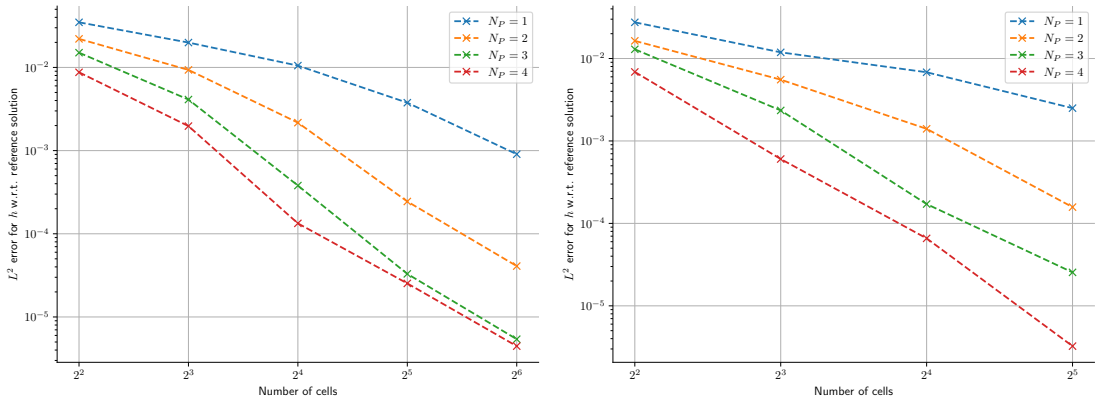
The quasi-exact solution is again computed using the Python scipy toolkit [45] with $\Delta\xi = 10^{-5}$, and until about $|\xi| \approx 7.58$ in both the positive and the negative direction. At this point, the difference $|h - H|$ was in the order of 10^{-13}, i.e. close to the prescribed conditions at infinity. For this accuracy of $\Delta\xi = 10^{-5}$, the generated Python and FORTRAN files had a size of 151.6 MiB and 209.4 MiB, respectively. We simply hard-coded the arrays into the code. The scenario in sam(oa)² is called `SOLITARY_WAVE`.

Instead of using a periodic domain, we use the solitary wave as target boundary condition; that is, `scenario` boundary conditions. Thus, we use a domain of size $200m$, and we start the solitary wave at $75m$, and let it run right-wards until we arrive at $125m$.

Note that it was beneficial to take a solitary wave of amplitude $A = 0.01$ instead of $A = 0.2$, since higher amplitudes may be interpolated incorrectly. This is because higher amplitudes cause the waves to become smaller in total horizontal size, and thus the drop from the peak with $h = (A + H)$ to the limit conditions $h = H$ happens quicker. Thus, we may get due to the projection to initial values for h which are lower than the initial water height H. This subsequently causes small oscillations in the long run, as seen in Figure 5.6.

In Table 5.2, we show the result, and a plot of the L^2 error in Figure 5.7. As it can be seen, the convergence is as expected for low degrees, but slightly lower than expected for the higher degrees. Most likely was our reference solution computed too coarsely to give a better result here: since the errors appeared around 10^{-5} to 10^{-6}, we conjecture that here the error of the linear interpolation begins to influence the result at about this range. This is supported by the following fact: over time, the solitary wave with polynomial degree 4 and subdivision level 12 showed a small, smooth error function in h which had the same shape as w in Figure 2.1. In short, the solitary wave was tilted slightly. This error function periodically reduced in amplitude and then rose again. For reference, the error for $N_P = 4$ and subdivision 12 was in the range of 10^{-6}, but periodically reduced to the range 10^{-7}. The initial projection error was in the range of 10^{-8}. An additional reason could be that we get the influence of the problem we have already seen in Figure 5.3. This would mean that the non-hydrostatic pressure causes larger errors, due to it having a high variation. We show an additional table with Table 5.3, where we took the minimum error over the last 25 steps which is when the wave passed the $100m$ mark on its way to $125m$. As it can be seen, the convergence rate is generally higher in this case, and it almost reaches the theoretical order in some cases.

In total, this still shows us that experimentally, we have convergence up to the accuracy of the reference solution for the lower polynomial degrees, and suboptimal convergence



(a) Even subdivisions $4, 6, 8, 10, 12$

(b) Odd subdivisions $5, 7, 9, 11$

Figure 5.7: L^2 error with respect to the reference solitary wave computed with $\Delta\xi = 10^{-5}$, and linearly interpolated in between, for the polynomial degrees $N_P = 1$ through $N_P = 4$.

for the higher (i.e. $N_P \in \{3, 4\}$) polynomial degrees.

Interestingly enough, once we had a high-enough order (order 4 or higher), and a high enough resolution, the number of predictor iterations was almost always as high as twice the number of cells: e.g. if we had 8192 cells, then there would be exactly 16384 iterations.

5.4 Riemann Problems

Since we have computed the solution to the Riemann Problem in Section 2.5, we can also run sam(oa)^2 on the same initial data, and compare its output. From now on, we will refer to the solution which we computed in Section 2.5 as reference solution.

Naturally, this will involve the limiter. To get to the situation of the Riemann problem (2.4.50), we assume that $\nabla b \equiv 0$, the friction is set to $n_m = 0$, and wave breaking is disabled. We also need to ensure that $w \equiv 0$ over time which can be achieved by setting $\gamma = 0$, and $w(0, \cdot) \equiv 0$. Then, $\partial_t(hw) = 0$ follows from (2.0.10). Since $(hw)(0, \cdot) \equiv 0$, we get $w \equiv 0$ by $h > 0$.

For all Riemann problems which will be computed in the following, we take the initial conditions $c = 4$, $g = 9.81$. Both are taken in order to conform with the solutions from Figure 2.4. The subdivision is set to 14, and the simulation is run for $10ms$ unless mentioned otherwise. This short simulation time is chosen this way, in order to reduce the influence from the boundary. Sometimes we will also run the simulation for a longer time, up to $50ms$, then the effects from the boundary become noticeable.

Throughout this section, we assume a CFL condition of 0.2. Such a low CFL number is expected, since we compute the timestep from the limiter averages. Thus, we do

N_P	N	L^2 error in h	rate for h	target
1	4	$3.4859 \cdot 10^{-02}$	-	2
1	8	$1.9901 \cdot 10^{-02}$	0.8087	2
1	16	$1.0513 \cdot 10^{-02}$	0.9207	2
1	32	$3.7654 \cdot 10^{-03}$	1.4813	2
1	64	$9.0505 \cdot 10^{-04}$	2.0567	2
2	4	$2.2042 \cdot 10^{-02}$	-	3
2	8	$9.3317 \cdot 10^{-03}$	1.2400	3
2	16	$2.1679 \cdot 10^{-03}$	2.1058	3
2	32	$2.4450 \cdot 10^{-04}$	3.1484	3
2	64	$4.0904 \cdot 10^{-05}$	2.5795	3
3	4	$1.5042 \cdot 10^{-02}$	-	4
3	8	$4.1102 \cdot 10^{-03}$	1.8717	4
3	16	$3.8001 \cdot 10^{-04}$	3.4351	4
3	32	$3.2966 \cdot 10^{-05}$	3.5270	4
3	64	$5.4039 \cdot 10^{-06}$	2.6089	4
4	4	$8.7371 \cdot 10^{-03}$	-	5
4	8	$1.9712 \cdot 10^{-03}$	2.1481	5
4	16	$1.3348 \cdot 10^{-04}$	3.8844	5
4	32	$2.5311 \cdot 10^{-05}$	2.3988	5
4	64	$4.4606 \cdot 10^{-06}$	2.5045	5

Table 5.2: Convergence rate for even subdivisions when comparing the L^2 error of h at $t = 50/c_A$ seconds in the solitary wave simulation. The scenario is `SOLITARY_WAVE`, the theoretical estimate is taken from [15].

N_P	N	L^2 error in h	rate for h	target
1	4	$3.4074 \cdot 10^{-02}$	-	2
1	8	$1.7099 \cdot 10^{-02}$	0.9948	2
1	16	$8.6860 \cdot 10^{-03}$	0.9772	2
1	32	$2.9606 \cdot 10^{-03}$	1.5528	2
1	64	$7.1119 \cdot 10^{-04}$	2.0576	2
2	4	$1.9642 \cdot 10^{-02}$	-	3
2	8	$7.4062 \cdot 10^{-03}$	1.4071	3
2	16	$1.8438 \cdot 10^{-03}$	2.0061	3
2	32	$1.9548 \cdot 10^{-04}$	3.2375	3
2	64	$3.4243 \cdot 10^{-05}$	2.5132	3
3	4	$1.3053 \cdot 10^{-02}$	-	4
3	8	$2.8242 \cdot 10^{-03}$	2.2084	4
3	16	$2.9878 \cdot 10^{-04}$	3.2407	4
3	32	$2.9773 \cdot 10^{-05}$	3.3270	4
3	64	$2.3363 \cdot 10^{-06}$	3.6717	4
4	4	$5.2688 \cdot 10^{-03}$	-	5
4	8	$9.2690 \cdot 10^{-04}$	2.5070	5
4	16	$8.7349 \cdot 10^{-05}$	3.4076	5
4	32	$3.1612 \cdot 10^{-06}$	4.7883	5
4	64	$7.1681 \cdot 10^{-07}$	2.1408	5

Table 5.3: Convergence rate for even subdivisions when comparing the L^2 error of h when taking the minimum of the error over the last 25 of 50 total steps for each run indiviually in the solitary wave simulation. The scenario is `SOLITARY_WAVE`, the theoretical estimate is taken from [15].

not capture the oscillations which are bound to occur at the contact discontinuities and shocks.

If not stated otherwise, the DMP check is enabled.

5.4.1 One-Dimensional Riemann Problems

We begin with the reproduction of the Riemann problems stated in Table 2.1, and we are going reproduce of all of them in a qualitative way. Naturally, since they are problems in one dimension, we will mostly show one-dimensional plots again.

As reference solution, we take the family of paths Ψ_C, since visually, it will not make a difference. However, we conjecture that this is due to the fact that the numerical fluxes use Ψ_P, i.e. linear paths in primitive variables. Therefore, we will most likely not obtain a completely perfect result, but still a very good one.

Dam Break Problem

We begin by comparing to the dam break, for which we take the values from Table 2.1 (a). As boundary conditions, we use the initial conditions. See Figure 5.8 for a comparison of the Riemann problem to our reference solution, as it can be seen in 2.4 in row 1 where the As we see in the plots, the solution structure is identical to the one we computed, but slightly smoothed out. This is perfectly in line with the back-projection from the finite volume grid to the DG grid, since we preserve the averages. We can see small artifacts at the contact discontinuity and the shock. Other than that, it approximates the solution very well. The only notable artifact is for u around the area where h and p have the contact discontinuity. Here, we see that it slightly dips down at this point. We also ran the simulation with $\gamma = 2$, however the results were so similar to what is seen in Figure 5.8 that we decided to omit them. This is most likely, since we only simulated for $10ms$.

The DMP check shown at $5ms$ is shown in Figure 5.11. This shows clearly that the DMP check limits the solution in the center, with occasional misses away from the dam break. Other than that, the DMP check seems to limit cells again that has already limited, so it looks almost as it would oscillate. For comparison, we ran the implementation from [37] on the shallow water dam break with limiter, and we got similar results regarding to where the limiter activated. If we disable the DMP check, we get a slightly different result, as it is seen in Figure 5.9. This time, we are able to see more unsteady artifacts at the DG cells which are adjacent to a shock or a contact discontinuity. For the rarefaction, there is no apparent visible difference.

In Figure 5.10, we ran the simulation a bit longer, and for more different fluxes. It can be seen that the `llfb` flux has the most smoothening effect, directly followed by the `hll` flux, and then the `roe` and `dot` flux which nearly give a completely identical result. Only for the `roe` and `dot` flux, we see the dip in u around the contact discontinuity, while `llfb` and `hll` slightly overshoot at the shock.

In total, that the Riemann problem can be successfully reproduced by our implementation in sam(oa)² which is completely unrelated to the Python reference script we wrote,

it indicates effectively two things at once. Firstly, our implementation in sam(oa)² is correctly implemented for such problems, and secondly, the calculations to the Riemann problem are correct, up to the choice of paths.

Symmetric Expansion

In Figure 5.12, the symmetric expansion (Table 2.1 (b), scenario `PULLAWAY_STRONG`) is plotted after $10ms$. It is visible that the rarefactions are approximated, and the non-differentiable points are smoothed out. To verify that the rarefactions are correctly computed, we run the simulation a bit longer, until we reach $40ms$. The plot for this timestep can be seen in Figure 5.13. Naturally, we plot a larger area. However, the rarefactions approximate the computed solution from Figure 2.4 better. This leads us to the conclusion that the rarefactions are indeed correctly computed, and the visible error is solely the approximation error. The small error in the middle is due to a small projection error in the initial conditions, where the velocity is set to the wrong values at the discontinuity.

Symmetric Collision

The symmetric collision (Table 2.1 (c), scenario `COLLISION`) is plotted in Figure 5.14 after Figure 5.12 shows the symmetric expansion and Figure 5.14 the symmetric collision, compared with the reference solution. As it can be seen, it behaves exactly as the reference solution, with the discontinuities slightly smoothed out as with the dam break. However, the initial data we chose is in the end too small to give a conclusive answer to the question which family of paths for the shocks is the correct one. In addition, this is also influenced by the fact that our numerical fluxes mostly use the linear paths in primitive variables Ψ_P.

Pressure Exchange

The pressure exchange (Table 2.1 (d)) is shown in Figure 5.15. It is the scenario `PRESSURE_EXCHANGE`. Once again, the behavior is as usual in the sense that it behaves exactly like the dam break, but with h and p switched. Other than that, we see no particularly different behavior here, even the small dip seen in u is seemingly the same as from the dam break.

Wet-Dry Front

Finally, we look at the wet-dry front problem (Table 2.1 (e)) which is the scenario `WETDRYFRONT`. The results can be seen in Figure 5.16. As the plots show, this time the approximation is not as exact as in the previous plots, however it qualitatively follows the same behavior, i.e. we have the rarefaction on the right-hand side, and the contact discontinuity to $h = 0$ in the middle. However, the computed values for u and p are rather low compared to our reference.

We tried to compare the fluxes for this scenario, i.e. we let it run a bit longer. Then, we compare the values from the `llfb`, `roe`, `hll`, and `dot` fluxes for the finite volume part. For the DG part, we continue to use `llfb` only. The result can be seen in Figure 5.17. Surprisingly, the `dot` flux shows to be the most divergent of all the fluxes, and the `roe` flux performs best. The `llfb` and `hll` flux behave almost similarly. Again, all of the fluxes go to 0 before the reference solution does so, but the `llfb`, `roe`, and `hll` fluxes show a much better approximation of the values for u and p, than the `dot` flux does. However, there is still a small problem with the `roe` implementation near the actual wet-dry front, in the sense that it cannot yet handle the case $C = 0$ correctly.

As a result, we conclude that the `roe` flux behaves best over the two test cases we tested it in which contradicts the intuition from e.g. [13] which tells us the opposite. Therefore, we conjecture that the reason is the more sophisticated choice of paths, i.e. the choice of Ψ_R.

5.4.2 Two-Dimensional Riemann Problems

In order to test in a two-dimensional setting, we now look at the whole two-dimensional problems.

One-Dimensional Problems in Two Dimensions

The previous, one-dimensional tests were all run in two dimensions. We will in particular report the behavior for the symmetric expansion (scenario `PULLAWAY_STRONG`), as we had absorbing boundary conditions as scenario boundary conditions. Subsequently, we gained an additional non-symmetric collision at each border. This can be seen in Figure 5.18. The two additional asymmetric collisions move inwards, while the expansion moves outwards. From both sides, there is a smoothing effect visible, since water flows into the expansion, and out of the collision.

Square Dam Break

Finally, we are going to consider a two-dimensional version of the dam break, with the initial values from Table 2.1 (a). This is the scenario `SQUARE_DAM_BREAK` from Table 5.1. In addition, we set $\gamma = 2$ this time. Running this gives us the results as shown in Figure 5.19 for h (or rather, η). The simulation is run until $50ms$. As we can see there, the results show, once again, a very similar structure to the reference solution which we computed in Chapter 2, at least in the cardinal directions. That is, we have a smooth transition from red to white-red which is the centered rarefaction. The next hard drop is the contact discontinuity. Finally, the jump from light blue to deep blue is the shock, and it can be seen to propagate very quickly. In the corners, we can see at the beginning a smoothening effect, but also a drop in water height between the contact discontinuity and the rarefaction. This could help to determine the behavior, if we should consider a two-dimensional Riemann problem. But still, it may be more useful for this case to use $\gamma = 0$, unlike here. However, the influence of γ is negligible in the beginning, as w is

near 0 then. The effect grows stronger over time, as it can be seen in Figure 5.20. We show w there, overlaying it with the plot of η at this timestep shows that the influence of w is the strongest around the contact discontinuity. It is interesting to see that the contact discontinuity seems to hold for w as well in the sense that it varies smoothly on both outside and inside the contact discontinuity ring.

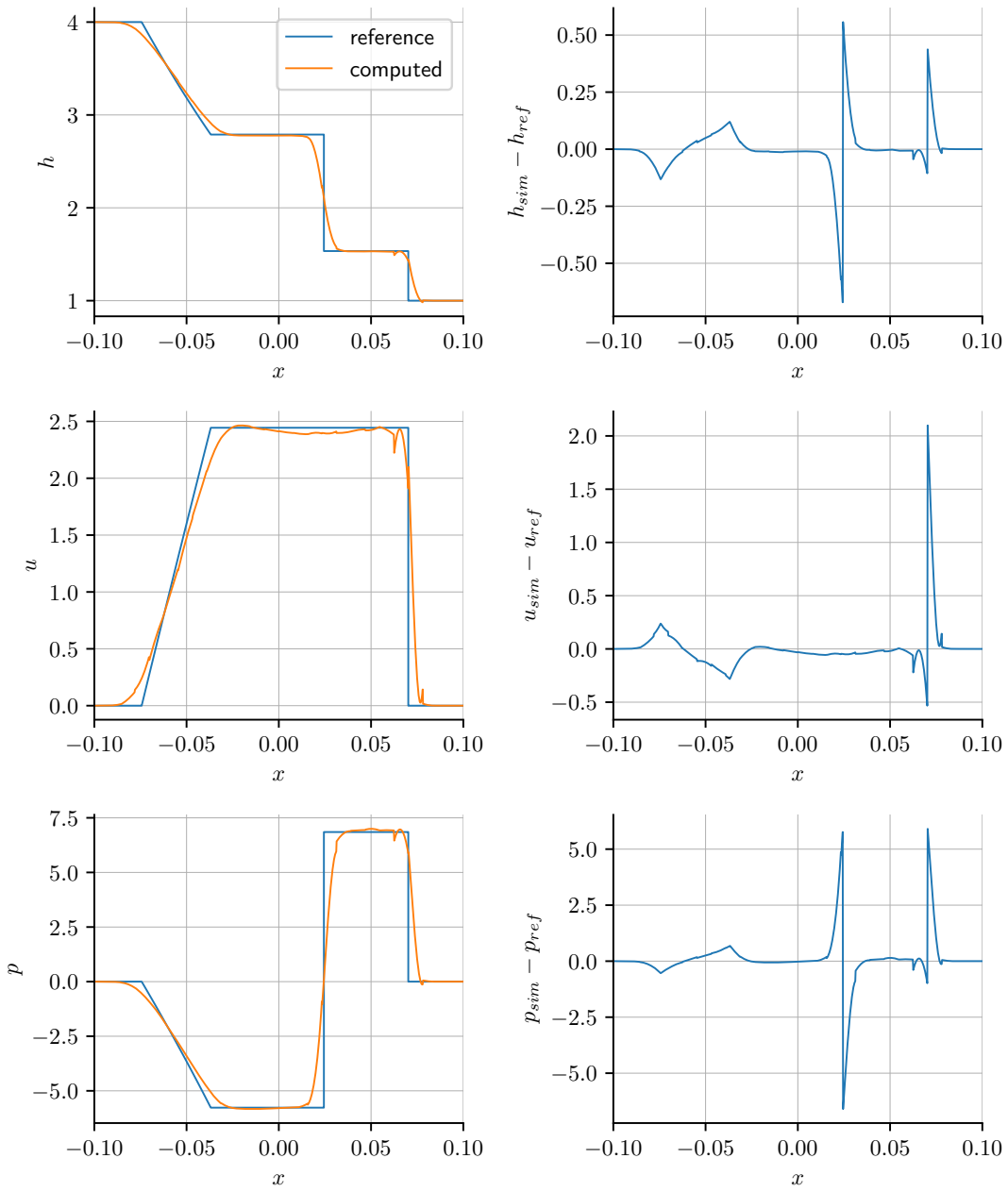


Figure 5.8: Comparing the sam(oa)²-computed dam break solution (Table 2.1 (a)) with the reference solution from Figure 2.4, while the DMP check is active. On the left-hand side, the two solutions are overlayed over one another, and on the right-hand side, the difference between reference and computed solution is shown. The constants are set to $g = 9.81, c = 4, \gamma = 0$. The final simulation time is $10ms$.

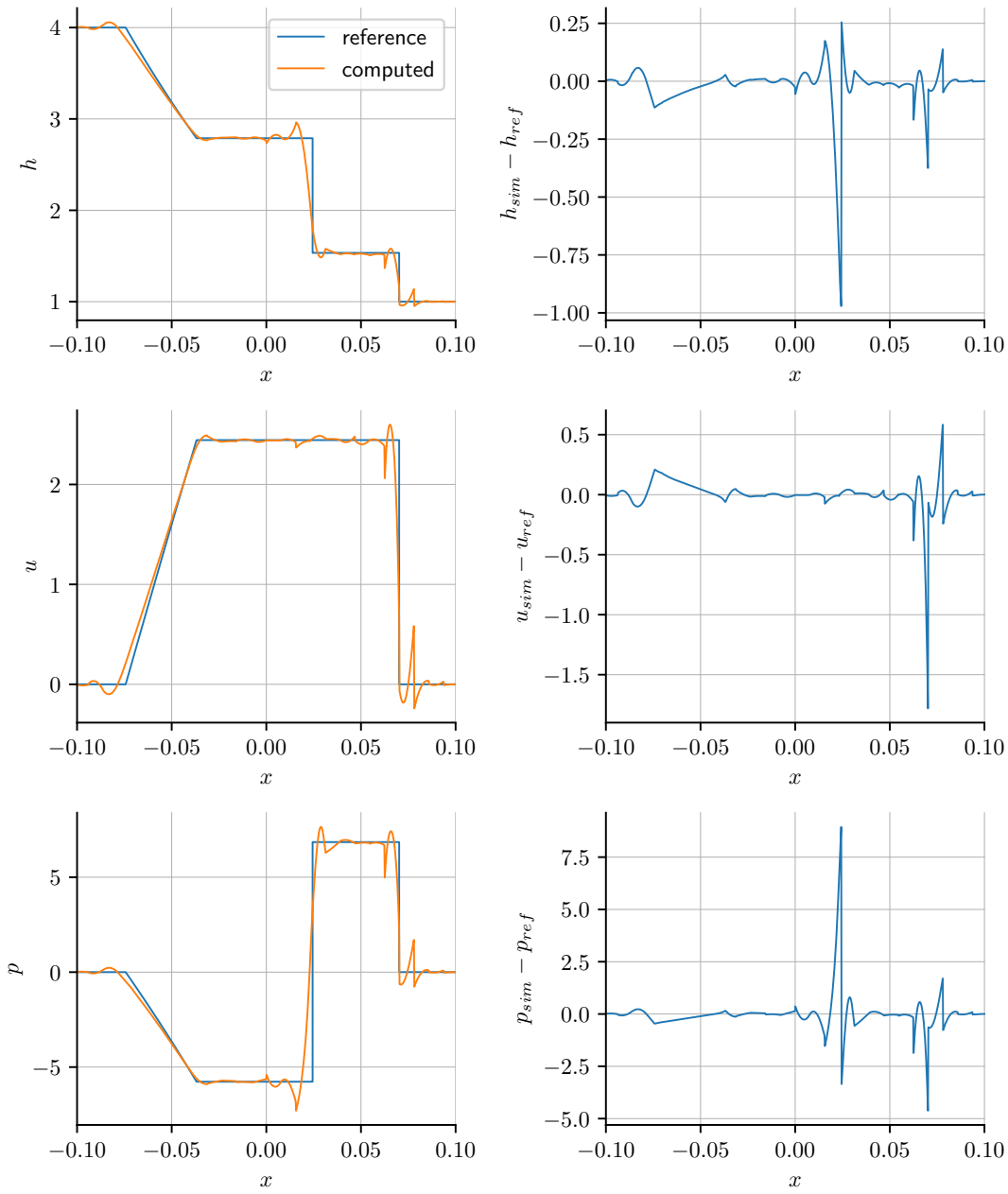


Figure 5.9: Comparing the sam(oa)²-computed dam break solution (Table 2.1 (a)) with the reference solution from Figure 2.4, while the DMP check is not active. On the left-hand side, the two solutions are overlayed over one another, and on the right-hand side, the difference between reference and computed solution is shown. The constants are set to $g = 9.81, c = 4, \gamma = 0$. The final simulation time is $10ms$.

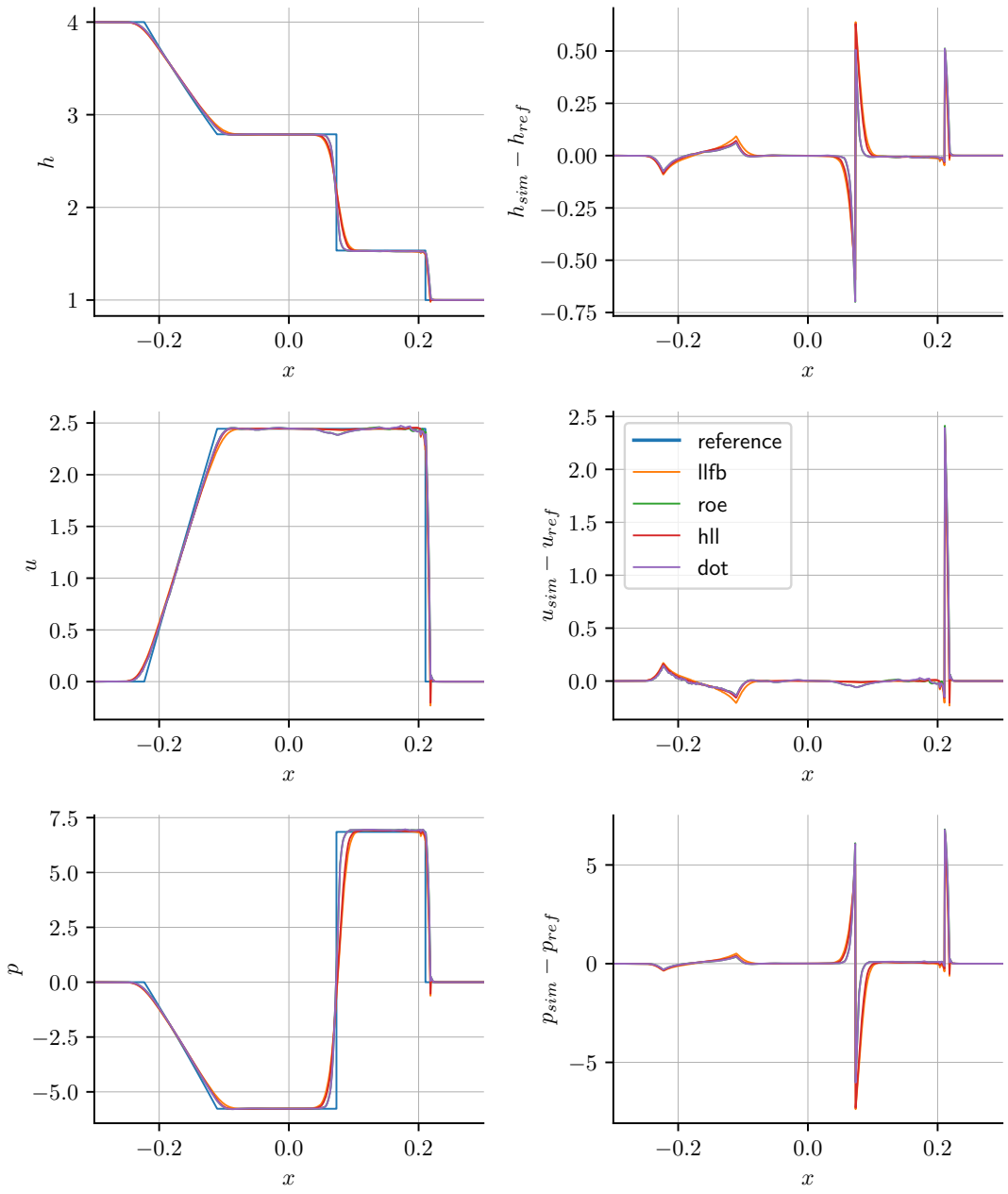


Figure 5.10: Comparing the sam(oa)²-computed dam break solution (Table 2.1 (a)) with the reference solution from Figure 2.4, while the DMP check is active. On the left-hand side, the two solutions are overlayed over one another, and on the right-hand side, the difference between reference and computed solution is shown. The result is plotted for different numerical fluxes. The `roe` flux is always hidden behind the `dot` flux and essentially yields the same result here. The constants are set to $g = 9.81, c = 4, \gamma = 0$. The final simulation time is $40ms$.

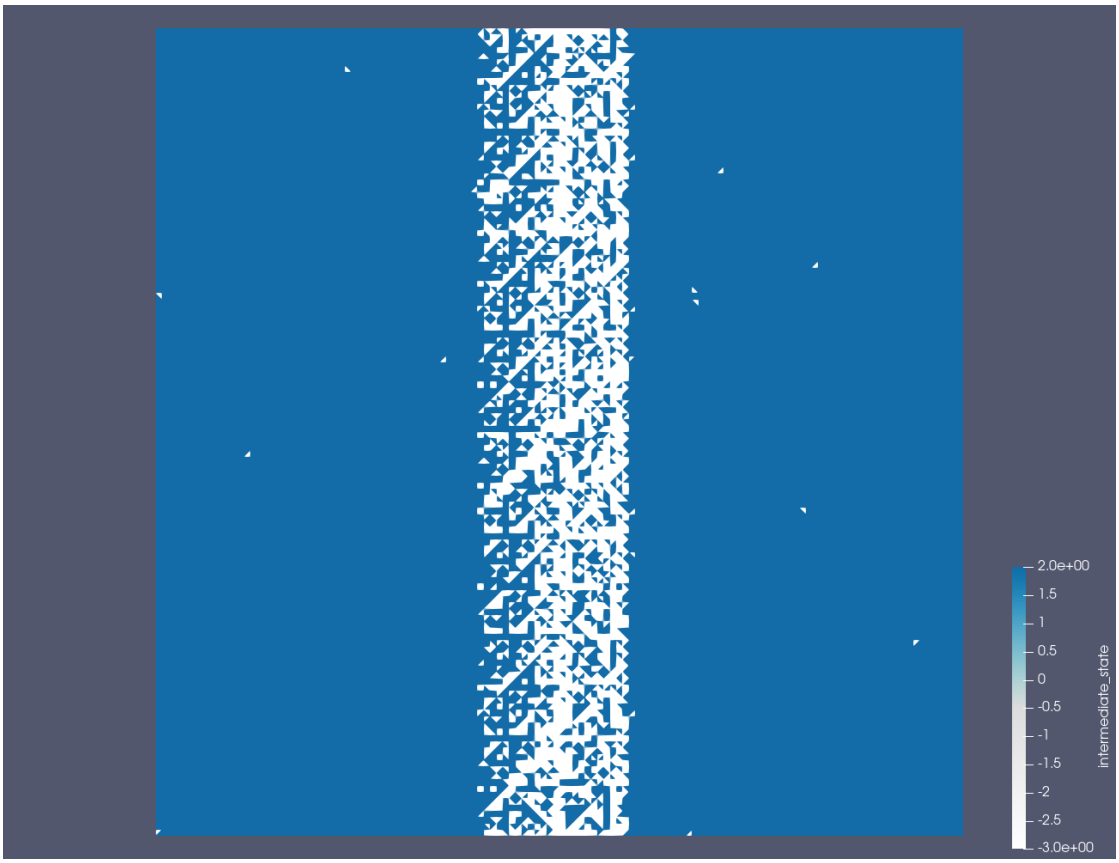


Figure 5.11: The locations where the DMP checker triggered at $5ms$ for the dam break, i.e. Table 2.1 (a). The dam break is located in the center.

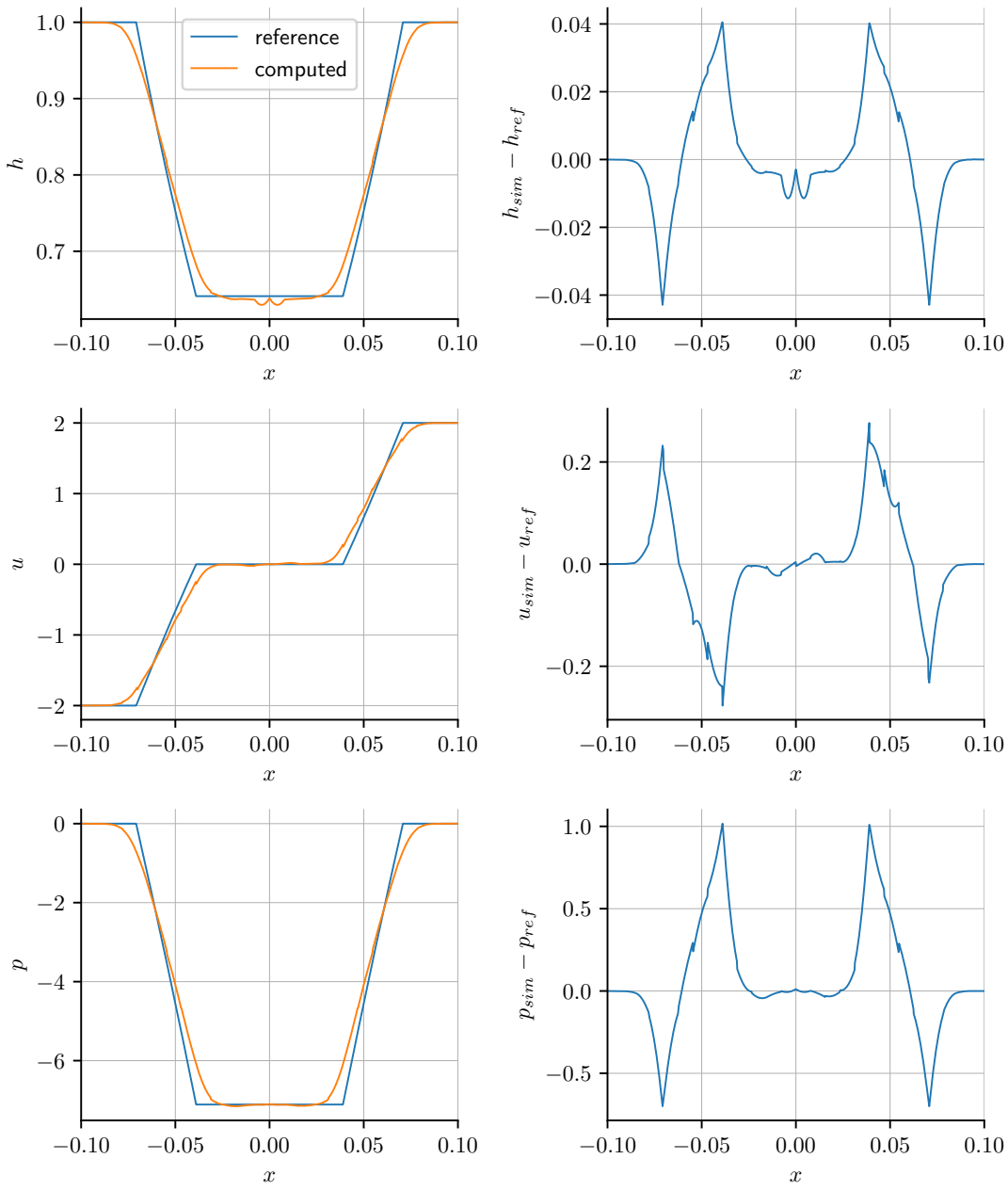


Figure 5.12: Comparing the sam(oa)²-computed symmetric expansion solution (Table 2.1 (b)) with the reference solution from Figure 2.4, while the DMP check is not active. On the left, reference and computed solutions are shown and on the right, their difference is plotted. The constants are set to $g = 9.81, c = 4, \gamma = 0$. The final simulation time is $10ms$.

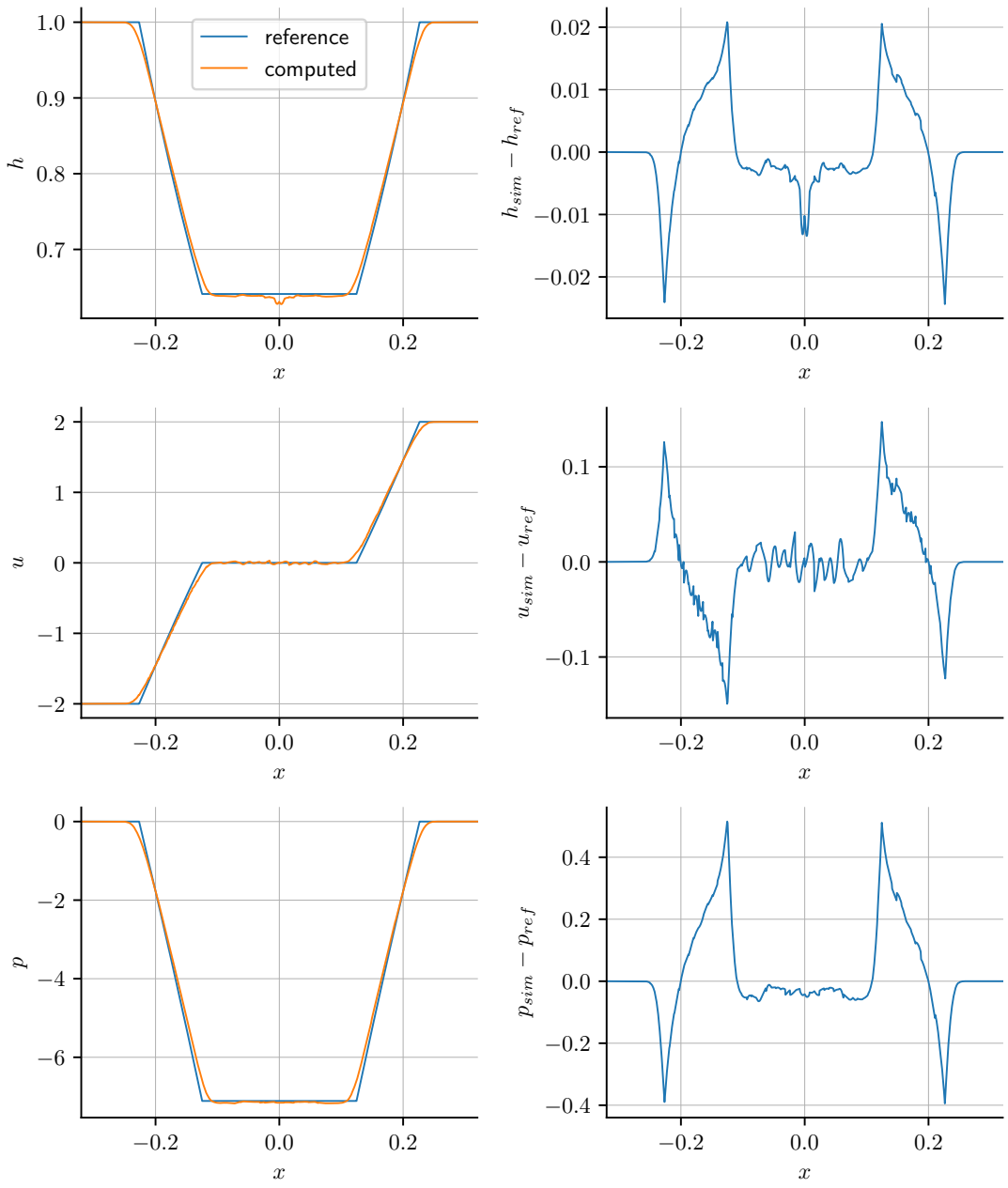


Figure 5.13: Comparing the sam(oa)²-computed symmetric expansion solution with the reference solution from Figure 2.4, while the DMP check is active. On the left, reference and computed solutions are shown and on the right, their difference is plotted. The constants are set to $g = 9.81, c = 4, \gamma = 0$. The final simulation time is $40ms$.

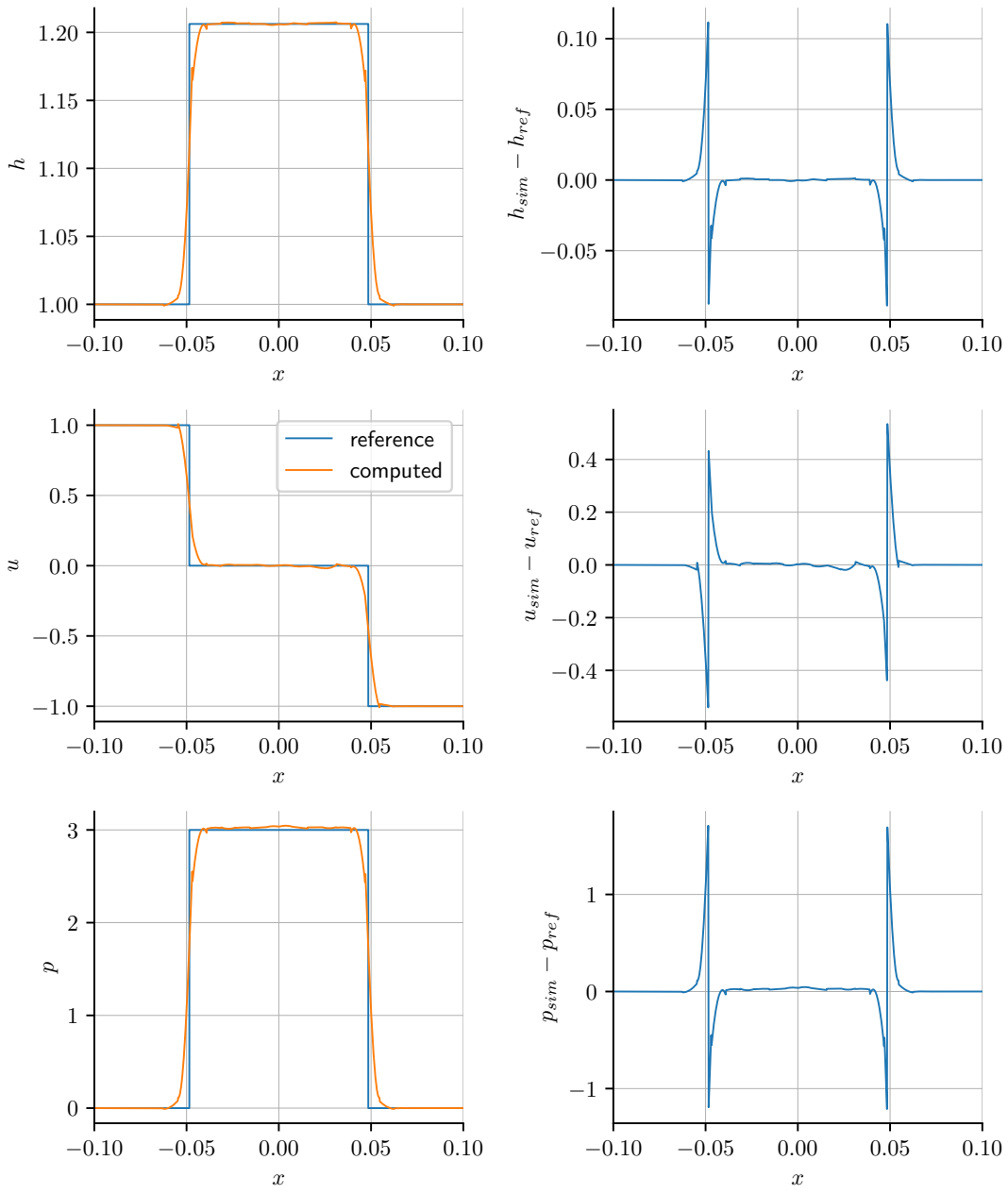


Figure 5.14: Comparing the sam(oa)²-computed symmetric collision solution (Table 2.1 (c)) with the reference solution from Figure 2.4, while the DMP check is not active. On the left, reference and computed solutions are shown and on the right, their difference is plotted. The constants are set to $g = 9.81, c = 4, \gamma = 0$. The final simulation time is $10ms$.

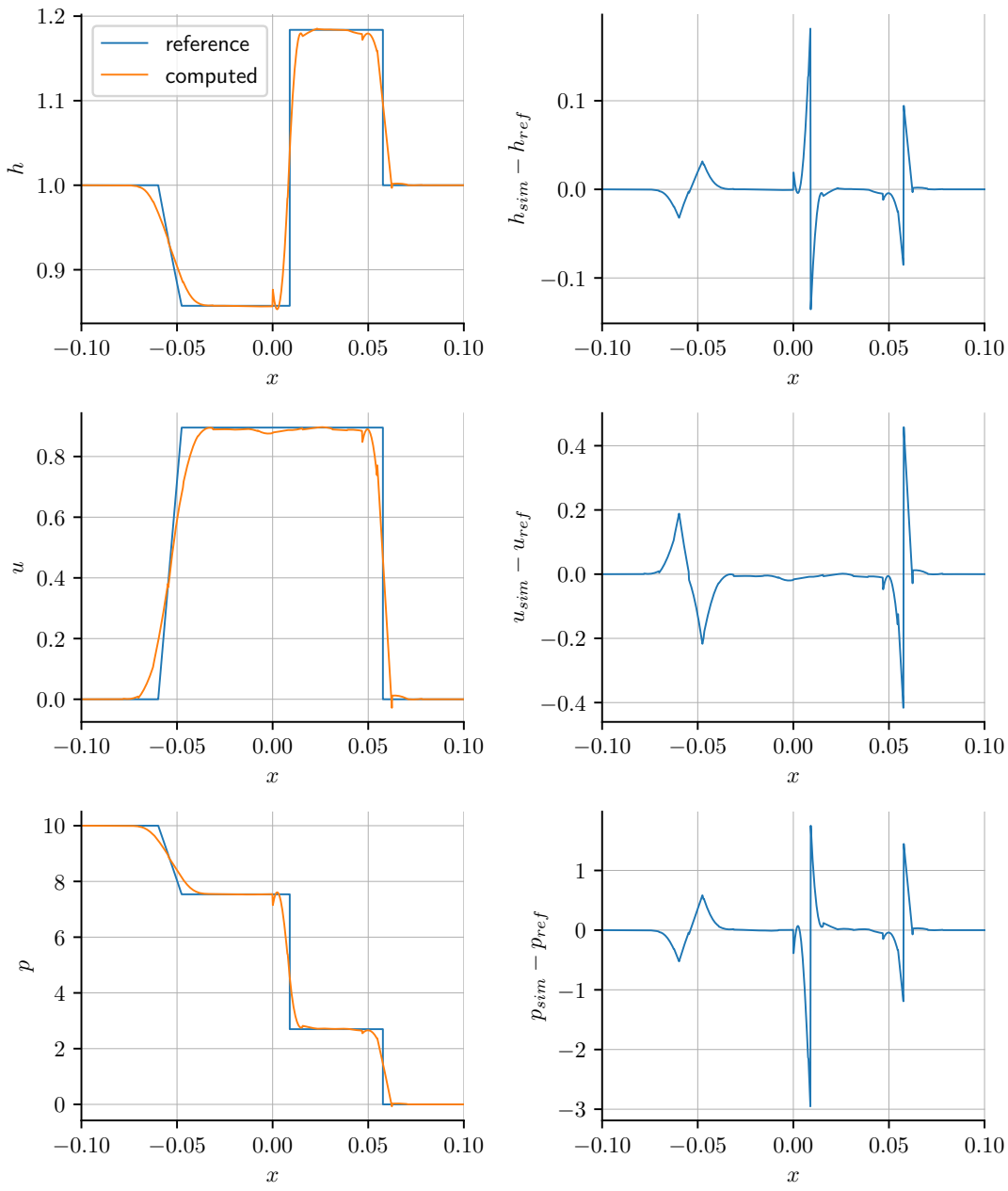


Figure 5.15: Comparing the sam(oa)²-computed pressure exchange solution (Table 2.1 (d)) with the reference solution from Figure 2.4, while the DMP check is active. On the left, reference and computed solutions are shown and on the right, their difference is plotted. The constants are set to $g = 9.81, c = 4, \gamma = 0$. The final simulation time is $10ms$.

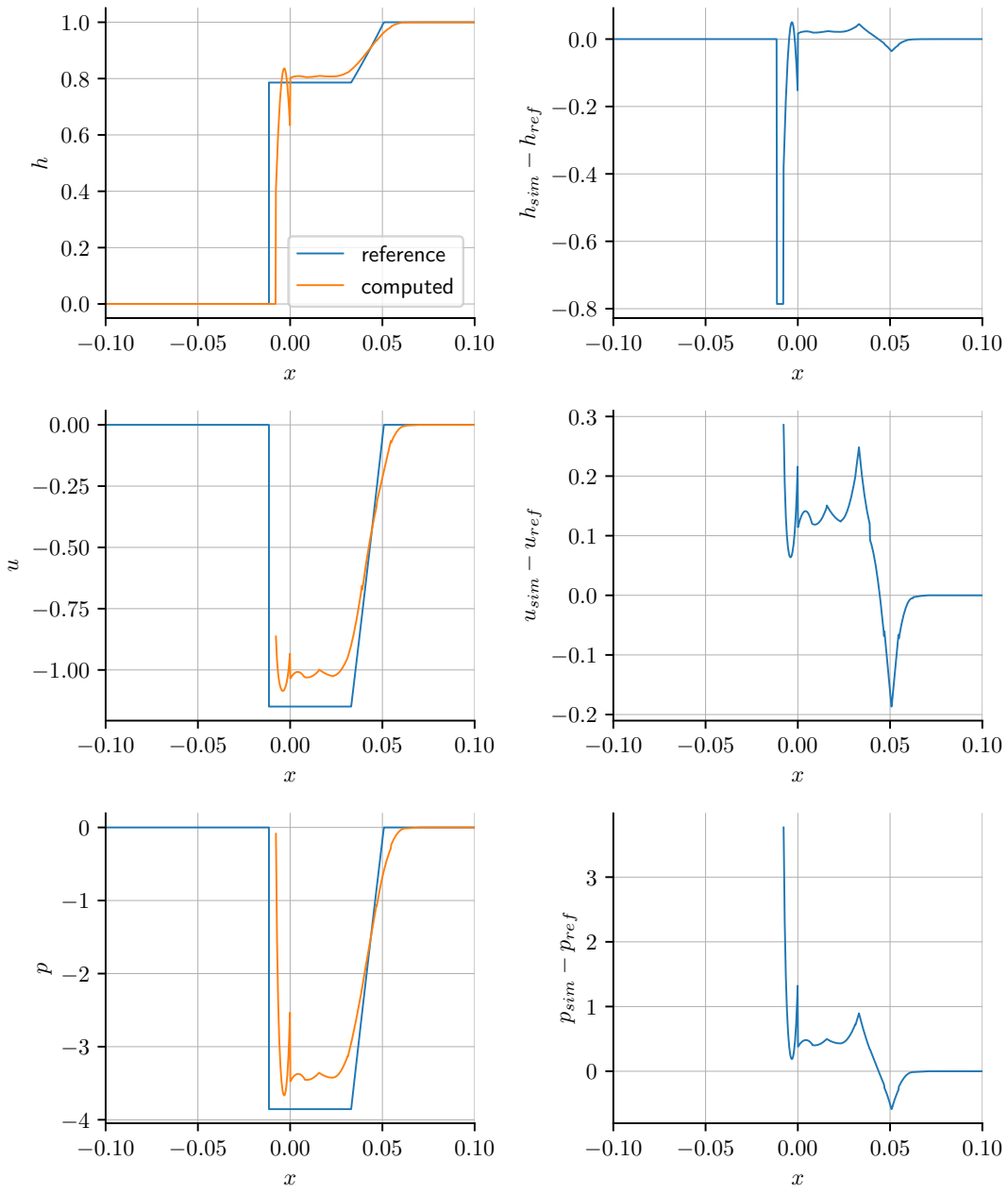


Figure 5.16: Comparing the sam(oa)²-computed wet-dry front solution (Table 2.1 (e)) with the reference solution from Figure 2.4, while the DMP check is active. On the left, reference and computed solutions are shown and on the right, their difference is plotted. The constants are set to $g = 9.81, c = 4, \gamma = 0$. The final simulation time is $10ms$.

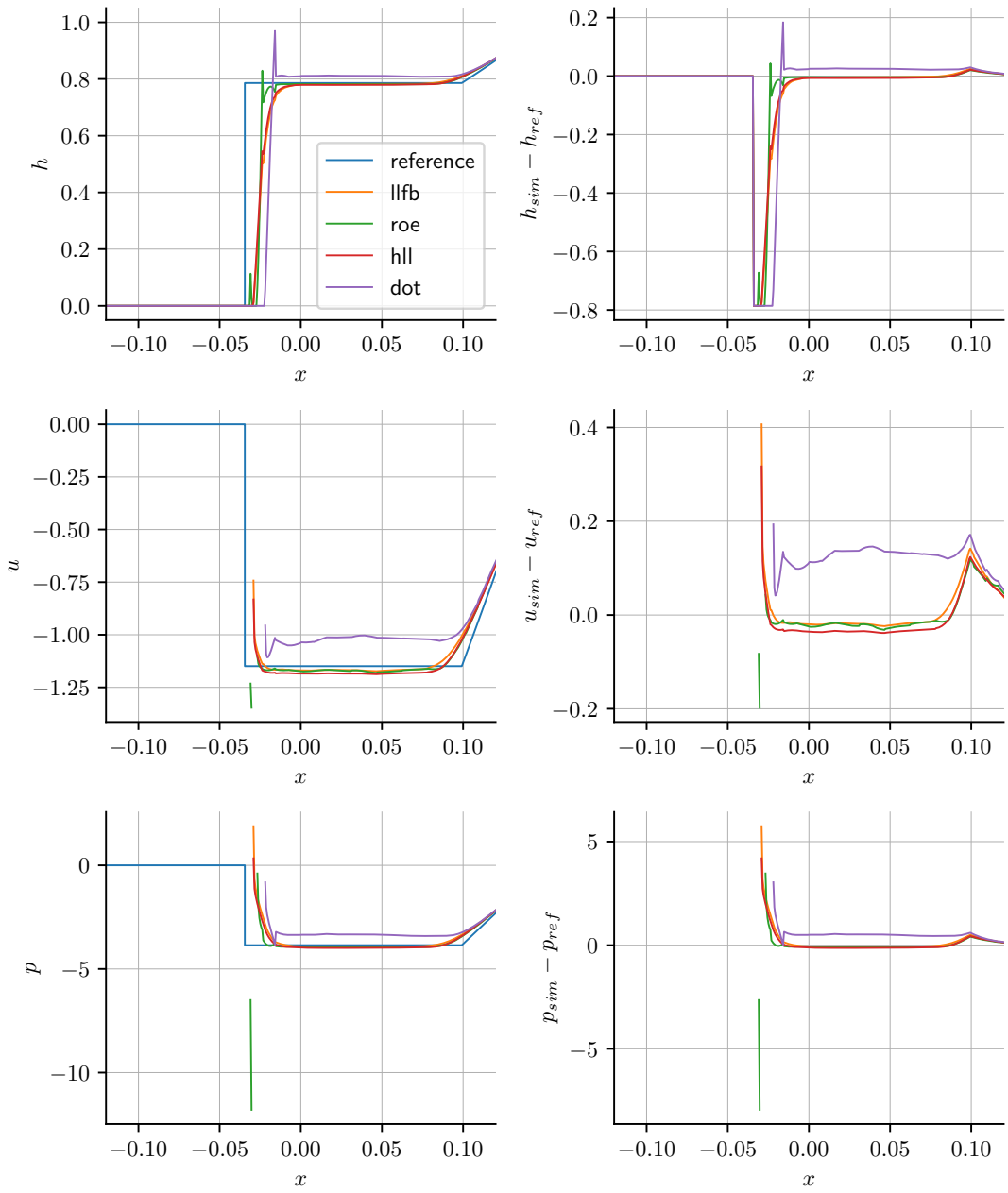


Figure 5.17: Comparing the sam(oa)²-computed wet-dry front solution Table 2.1 (e)) with the reference solution from Figure 2.4, while the DMP check is active. On the left, reference and computed solutions are shown and on the right, their difference is plotted. We test for different fluxes, and run for $40ms$. The constants are set to $g = 9.81, c = 4, \gamma = 0$. Near the wet-dry front, artifacts can be seen for the `roe` flux.

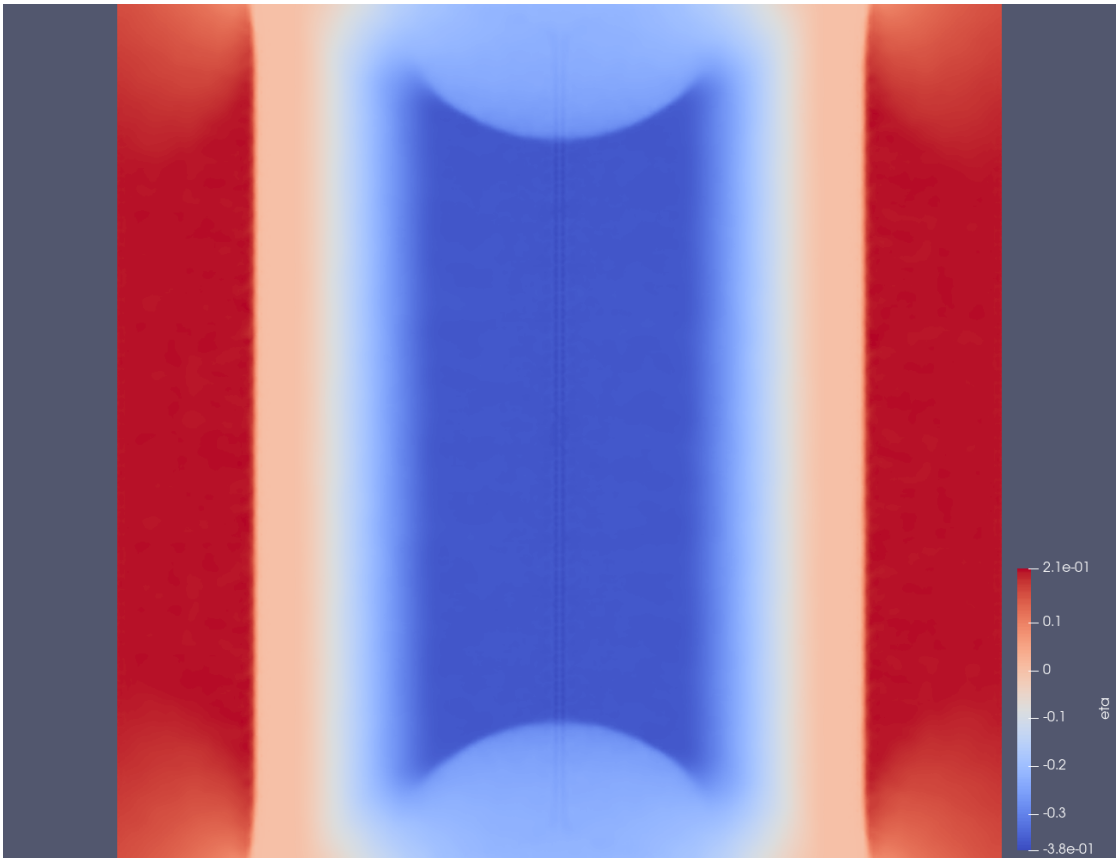
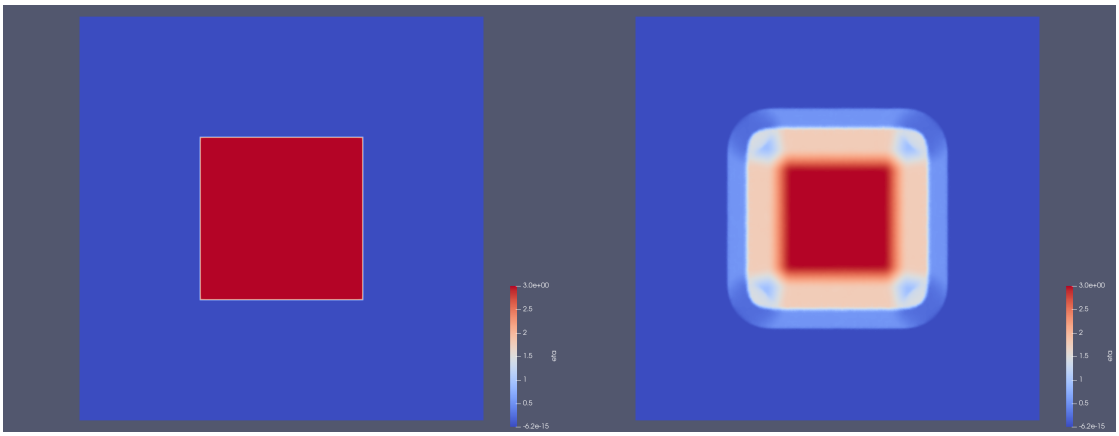
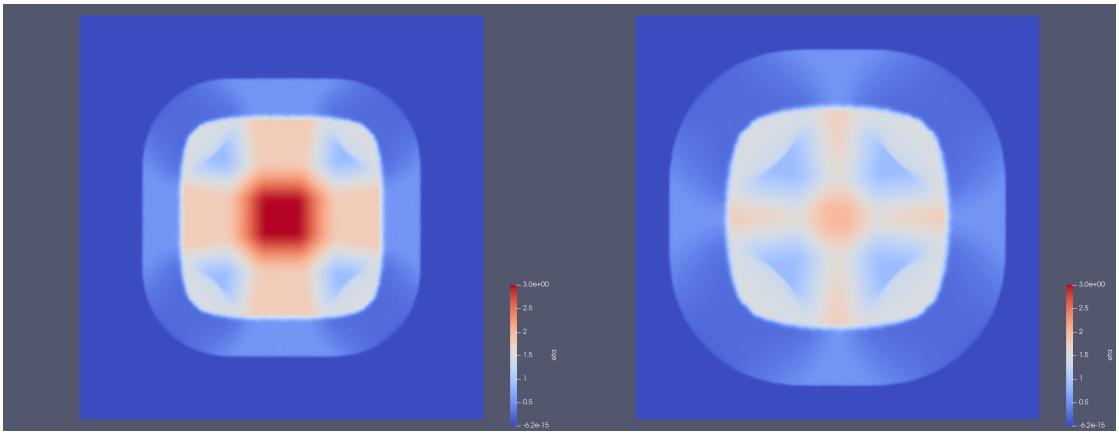


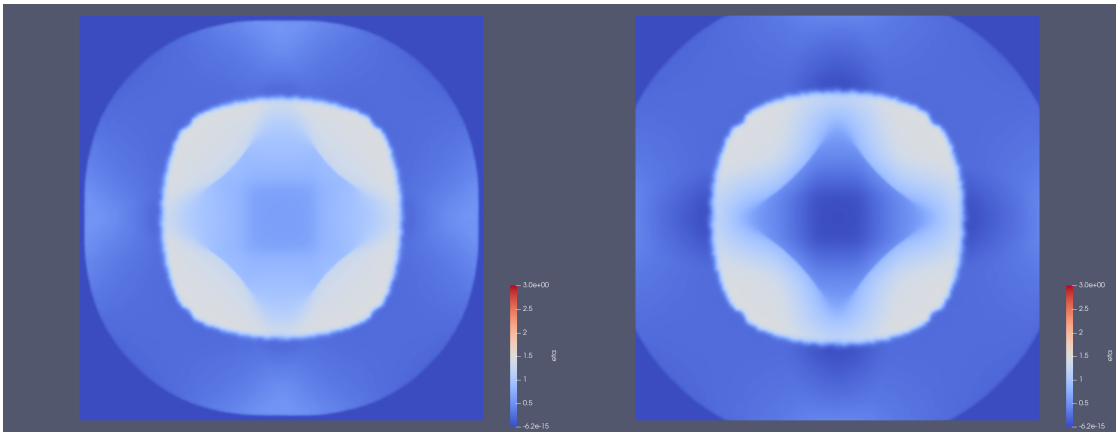
Figure 5.18: A two-dimensional plot of the symmetric expansion (Table 2.1 (b)) at $40ms$. The initial conditions were set in the left and right half-plane, respectively; $g = 9.81, c = 4$. Boundary conditions were $\mathbf{U} \equiv 0$. Red is higher, blue is lower.



(a) $t = 0ms$ (b) $t = 10ms$

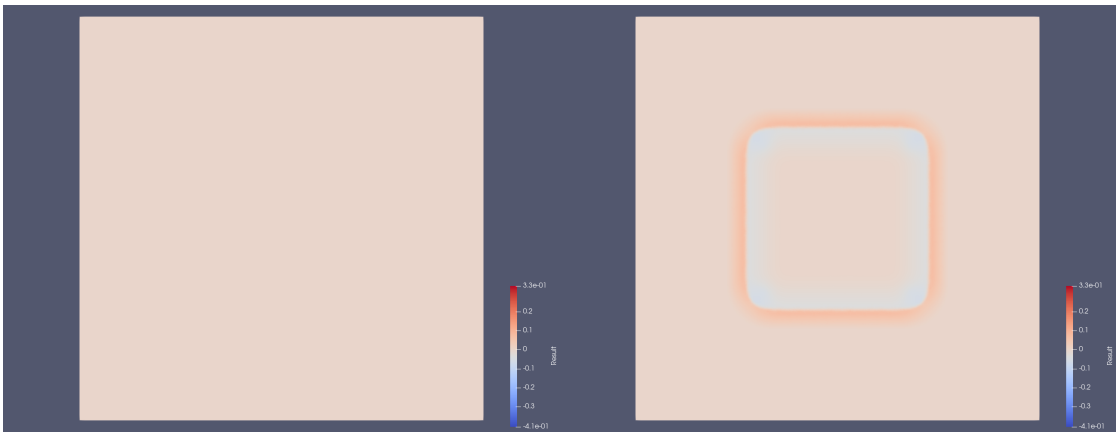


(c) $t = 20ms$ (d) $t = 30ms$

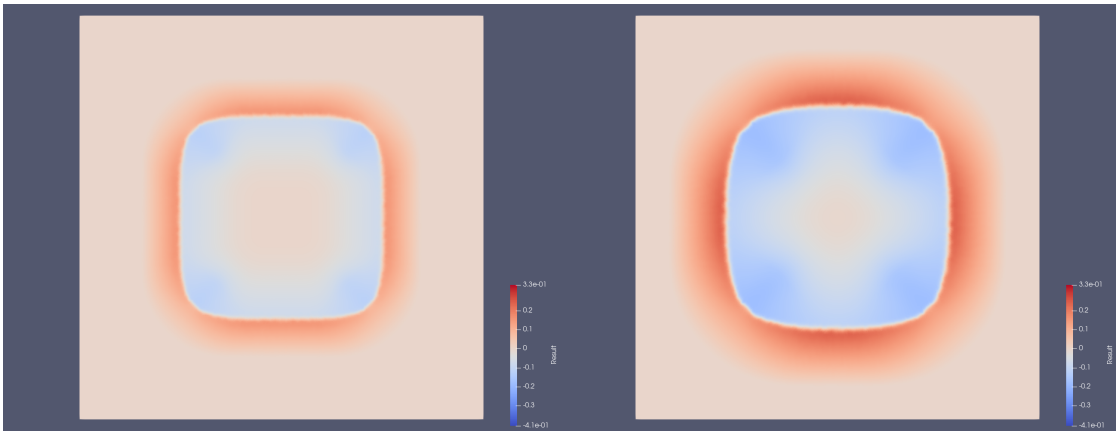


(e) $t = 40ms$ (f) $t = 50ms$

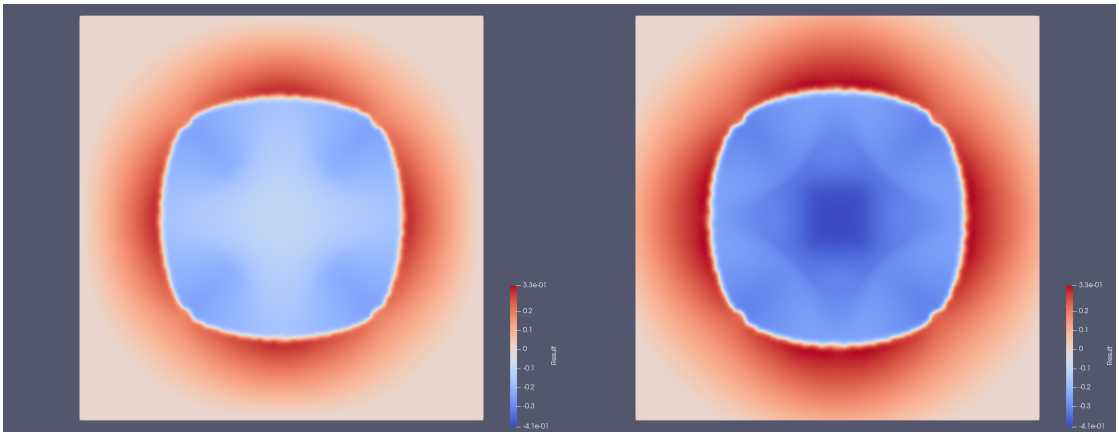
Figure 5.19: Water level η during the square dam break at different times. Initial conditions are similar to Table 2.1 (a). The constants are set to $g = 9.81, c = 4, \gamma = 2$.



(a) $t = 0ms$ (b) $t = 10ms$



(c) $t = 20ms$ (d) $t = 30ms$



(e) $t = 40ms$ (f) $t = 50ms$

Figure 5.20: Vertical velocity w during the square dam break at different times. Initial conditions are similar to Table 2.1 (a). In particular, $w \equiv 0$ in the beginning. The constants are set to $g = 9.81, c = 4, \gamma = 2$.

6 Summary

To summarize, we considered a hyperbolic equation system (2.0.10) which introduces a non-hydrostatic pressure component. We have considered and computed two types of solutions for this system. That is, we have improved the existing solitary wave solution, and we have computed the Riemann Problem for (2.0.10) without source and bathymetry, but for a larger families of paths and also for dry states. The structure of the Riemann Problem has shown to be a generalization of the shallow water equations, with the difference that we have a central contact discontinuity which exchanges water height with non-hydrostatic pressure. The effective influence of the contact discontinuity is dependent on the size of the approximation parameter c.

Next, we have described and implemented a finite volume method, and the ADER-DG method with a posteriori limiter in sam(oa)^2, and we empirically demonstrated that our scheme converges, and the rate matches the theoretical order for low polynomial degrees. Furthermore, we have shown its well-balancedness for a still resting lake in four test cases. In addition, we could reproduce both the solitary wave, as well as the Riemann problem. Especially for the latter, the fact that we were able to reproduce the previously-calculated solution for the Riemann problem is a strong indicator that both our implementation in sam(oa)^2, and our calculations of the reference solutions from Chapter 2 are both correct.

Further work can be made to improve the behavior of the system near wet-dry fronts, as well as extending the Riemann problem solution to varying bathymetry and vacuum situations. In addition, while we made first steps towards a more generic ADER-DG implementation inside sam(oa)^2, more work can be done in this direction as well.

References

[1] M. Bader. *Space-Filling Curves: An Introduction with Applications in Scientific Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. ISBN: 978-3-642-31046-1. DOI: 10.1007/978-3-642-31046-1. URL: https://doi.org/10.1007/978-3-642-31046-1.

[2] A. Breuer, A. Heinecke, and M. Bader. "Petascale Local Time Stepping for the ADER-DG Finite Element Method". In: *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2016, pp. 854–863. DOI: 10.1109/IPDPS.2016.109.

[3] M.-O. Bristeau, A. Mangeney, J. Sainte-Marie, and N. Seguin. "An energy-consistent depth-averaged Euler system: Derivation and properties". In: *Discrete & Continuous Dynamical Systems - B* 20.4 (2015), pp. 961–988.

[4] S. Busto, S. Chiocchetti, M. Dumbser, E. Gaburro, and I. Peshkov. "High Order ADER Schemes for Continuum Mechanics". In: *Frontiers in Physics* 8 (2020), p. 32. ISSN: 2296-424X. DOI: 10.3389/fphy.2020.00032. URL: https://www.frontiersin.org/article/10.3389/fphy.2020.00032.

[5] M. J. Castro, J. M. Gallardo, and A. Marquina. "Approximate Osher-Solomon Schemes for Hyperbolic Systems". In: *Trends in Differential Equations and Applications*. Ed. by F. Ortegón Gallego, M. V. Redondo Neble, and J. R. Rodríguez Galván. Cham: Springer International Publishing, 2016, pp. 1–16. ISBN: 978-3-319-32013-7. DOI: 10.1007/978-3-319-32013-7_1. URL: https://doi.org/10.1007/978-3-319-32013-7_1.

[6] M. J. Castro Díaz and E. Fernández-Nieto. "A Class of Computationally Fast First Order Finite Volume Solvers: PVM Methods". In: *SIAM Journal on Scientific Computing* 34.4 (2012), A2173–A2196. DOI: 10.1137/100795280. eprint: https://doi.org/10.1137/100795280. URL: https://doi.org/10.1137/100795280.

[7] D. E. Charrier and T. Weinzierl. "Stop talking to me - a communication-avoiding ADER-DG realisation". In: *CoRR* abs/1801.08682 (2018). arXiv: 1801.08682. URL: http://arxiv.org/abs/1801.08682.

[8] R. M. Corless, G. H. Gonnet, D. E. G. Hare, D. J. Jeffrey, and D. E. Knuth. "On the LambertW function". In: *Advances in Computational Mathematics* 5.1 (1996), pp. 329–359. ISSN: 1572-9044. DOI: 10.1007/BF02124750.

[9] A. Dedner, F. Kemm, D. Kröner, C.-D. Munz, T. Schnitzer, and M. Wesenberg. "Hyperbolic Divergence Cleaning for the MHD Equations". In: *Journal of Computational Physics* 175.2 (2002), pp. 645–673. ISSN: 0021-9991. DOI: 10.1006/jcph.2001.6961. URL: https://www.sciencedirect.com/science/article/pii/S002199910196961X.

[10] D. A. Di Pietro and A. Ern. *Mathematical Aspects of Discontinuous Galerkin Methods*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. ISBN: 978-3-642-22980-0. DOI: 10.1007/978-3-642-22980-0. URL: https://doi.org/10.1007/978-3-642-22980-0.

[11] M. Dumbser, D. S. Balsara, E. F. Toro, and C.-D. Munz. "A unified framework for the construction of one-step finite volume and discontinuous Galerkin schemes on unstructured meshes". In: *Journal of Computational Physics* 227.18 (2008), pp. 8209–8253. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2008.05.025. URL: https://www.sciencedirect.com/science/article/pii/S0021999108002829.

[12] M. Dumbser and R. Loubère. "A simple robust and accurate a posteriori subcell finite volume limiter for the discontinuous Galerkin method on unstructured meshes". In: *Journal of Computational Physics* 319 (2016), pp. 163–199. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2016.05.002. URL: https://www.sciencedirect.com/science/article/pii/S0021999116301292.

[13] M. Dumbser and E. F. Toro. "A Simple Extension of the Osher Riemann Solver to Non-conservative Hyperbolic Systems". In: *Journal of Scientific Computing* 48.1 (2011), pp. 70–88. ISSN: 1573-7691. DOI: 10.1007/s10915-010-9400-3.

[14] M. Dumbser, O. Zanotti, R. Loubère, and S. Diot. "A posteriori subcell limiting of the discontinuous Galerkin finite element method for hyperbolic conservation laws". In: *Journal of Computational Physics* 278 (2014), pp. 47–75. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2014.08.009. URL: https://www.sciencedirect.com/science/article/pii/S0021999114005555.

[15] C. Escalante, M. Dumbser, and M. Castro. "An efficient hyperbolic relaxation system for dispersive non-hydrostatic water waves and its solution with high order discontinuous Galerkin schemes". In: *Journal of Computational Physics* 394 (2019), pp. 385–416. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2019.05.035. URL: https://www.sciencedirect.com/science/article/pii/S0021999119303730.

[16] C. Escalante and T. M. de Luna. "A General Non-hydrostatic Hyperbolic Formulation for Boussinesq Dispersive Shallow Flows and Its Numerical Approximation". In: *Journal of Scientific Computing* 83.3 (2020), p. 62. ISSN: 1573-7691. DOI: 10.1007/s10915-020-01244-7.

[17] C. R. Ferreira and M. Bader. "A Generic Interface for Godunov-Type Finite Volume Methods on Adaptive Triangular Meshes". In: *Computational Science – ICCS 2019*. Ed. by J. M. F. Rodrigues, P. J. S. Cardoso, J. Monteiro, R. Lam, V. V. Krzhizhanovskaya, M. H. Lees, J. J. Dongarra, and P. M. Sloot. Cham: Springer International Publishing, 2019, pp. 402–416. ISBN: 978-3-030-22741-8.

[18] P. L. Floch. “Shock Waves for Nonlinear Hyperbolic Systems in Nonconservative Form”. In: 1989.

[19] S. Foundation. *SCons: A software construction tool*. URL: https://scons.org/ (visited on 03/13/2022).

[20] D. L. George. “Augmented Riemann solvers for the shallow water equations over variable topography with steady states and inundation”. In: *Journal of Computational Physics* 227.6 (2008), pp. 3089–3113. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2007.10.027. URL: https://www.sciencedirect.com/science/article/pii/S0021999107004767.

[21] E. Godlewski and P.-A. Raviart. *Numerical Approximation of Hyperbolic Systems of Conservation Laws*. New York, NY: Springer New York, 1996. ISBN: 978-1-4612-0713-9. DOI: 10.1007/978-1-4612-0713-9. URL: https://doi.org/10.1007/978-1-4612-0713-9.

[22] E. HAN and G. WARNECKE. “EXACT RIEMANN SOLUTIONS TO SHALLOW WATER EQUATIONS”. In: *Quarterly of Applied Mathematics* 72.3 (2014), pp. 407–453. ISSN: 0033569X, 15524485. URL: http://www.jstor.org/stable/43639119.

[23] J. Heinonen. *Lectures on Lipschitz Analysis*. 2004. URL: http://www.math.jyu.fi/research/reports/rep100.pdf (visited on 03/02/2022).

[24] D. I. Ketcheson, R. J. LeVeque, and M. J. del Razo. *Riemann Problems and Jupyter Solutions*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2020. DOI: 10.1137/1.9781611976212. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611976212. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611976212.

[25] G. Khakimzyanov, D. Dutykh, Z. Fedotova, and O. Gusev. *Dispersive Shallow Water Waves: Theory, Modeling, and Numerical Methods*. Cham: Springer International Publishing, 2020. ISBN: 978-3-030-46267-3. DOI: 10.1007/978-3-030-46267-3. URL: https://doi.org/10.1007/978-3-030-46267-3.

[26] P. D. Lax. “Hyperbolic systems of conservation laws II”. In: *Communications on Pure and Applied Mathematics* 10.4 (1957), pp. 537–566. DOI: 10.1002/cpa.3160100406. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpa.3160100406. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpa.3160100406.

[27] P. D. Lax. *Hyperbolic Systems of Conservation Laws and the Mathematical Theory of Shock Waves*. Society for Industrial and Applied Mathematics, 1973. DOI: 10.1137/1.9781611970562. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611970562. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611970562.

[28] P. G. LeFloch and M. D. Thanh. “The Riemann problem for the shallow water equations with discontinuous topography”. In: *Communications in Mathematical Sciences* 5.4 (2007), pp. 865–885. URL: https://doi.org/.

[29] P. G. LeFloch and A. E. Tzavaras. "Representation of Weak Limits and Definition of Nonconservative Products". In: *SIAM Journal on Mathematical Analysis* 30.6 (1999), pp. 1309–1342. DOI: 10.1137/S0036141098341794. eprint: https://doi.org/10.1137/S0036141098341794. URL: https://doi.org/10.1137/S0036141098341794.

[30] R. J. LeVeque. *Numerical Methods for Conservation Laws*. Basel: Birkhäuser Basel, 1992. ISBN: 978-3-0348-8629-1. DOI: 10.1007/978-3-0348-8629-1_1. URL: https://doi.org/10.1007/978-3-0348-8629-1.

[31] C. F. Matta, L. Massa, A. V. Gubskaya, and E. Knoll. "Can One Take the Logarithm or the Sine of a Dimensioned Quantity or a Unit? Dimensional Analysis Involving Transcendental Functions". In: *Journal of Chemical Education* 88.1 (2011). doi: 10.1021/ed1000476, pp. 67–70. ISSN: 0021-9584. DOI: 10.1021/ed1000476.

[32] O. Meister, K. Rahnema, and M. Bader. "Parallel Memory-Efficient Adaptive Mesh Refinement on Structured Triangular Meshes with Billions of Grid Cells". In: *ACM Trans. Math. Softw.* 43.3 (Sept. 2016). ISSN: 0098-3500. DOI: 10.1145/2947668. URL: https://doi.org/10.1145/2947668.

[33] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, Š. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, and A. Scopatz. "SymPy: symbolic computing in Python". In: *PeerJ Computer Science* 3 (Jan. 2017), e103. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.103. URL: https://doi.org/10.7717/peerj-cs.103.

[34] Muñoz-Ruiz, María Luz and Parés, Carlos. "Godunov method for nonconservative hyperbolic systems". In: *ESAIM: M2AN* 41.1 (2007), pp. 169–185. DOI: 10.1051/m2an:2007011. URL: https://doi.org/10.1051/m2an:2007011.

[35] C. Parés. "Numerical methods for nonconservative hyperbolic systems: a theoretical framework." In: *SIAM Journal on Numerical Analysis* 44.1 (2006), pp. 300–321. DOI: 10.1137/050628052. eprint: https://doi.org/10.1137/050628052. URL: https://doi.org/10.1137/050628052.

[36] Parés, Carlos and Castro, Manuel. "On the well-balance property of Roe's method for nonconservative hyperbolic systems. applications to shallow-water systems". In: *ESAIM: M2AN* 38.5 (2004), pp. 821–852. DOI: 10.1051/m2an:2004041. URL: https://doi.org/10.1051/m2an:2004041.

[37] L. Rannabauer, M. Dumbser, and M. Bader. "ADER-DG with a-posteriori finite-volume limiting to simulate tsunamis in a parallel adaptive mesh refinement framework". In: *Computers & Fluids* 173 (2018), pp. 299–306. ISSN: 0045-7930. DOI: 10.1016/j.compfluid.2018.01.031. URL: https://www.sciencedirect.com/science/article/pii/S0045793018300392.

[38] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version x.y.z)*. https://www.sagemath.org. 2022.

[39] N. Schlömer, N. Papior, D. Arnold, J. Blechta, and R. Zetter. *nschloe/quadpy: None*. Version v0.16.10. Sept. 2021. DOI: 10.5281/zenodo.5541216. URL: https://doi.org/10.5281/zenodo.5541216.

[40] T. sam(oa)² Team. *sam(oa)²*. URL: https://gitlab.lrz.de/samoa/samoa (visited on 03/02/2022).

[41] T. sam(oa)² Team. *sam(oa)² User Manual*. URL: https://samoa.readthedocs.io/en/latest (visited on 03/02/2022).

[42] E. Toro. "Computational Methods for Hyperbolic Equations". In: *Jets From Young Stars III: Numerical MHD and Instabilities*. Ed. by S. Massaglia, G. Bodo, A. Mignone, and P. Rossi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 3–69. ISBN: 978-3-540-76967-5. DOI: 10.1007/978-3-540-76967-5_1. URL: https://doi.org/10.1007/978-3-540-76967-5_1.

[43] E. F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997. ISBN: 978-3-662-03490-3. DOI: 10.1007/978-3-662-03490-3. URL: https://doi.org/10.1007/978-3-662-03490-3.

[44] H. F. Trotter. "On the product of semi-groups of operators". In: *Proceedings of the American Mathematical Society* 10.4 (1959), pp. 545–551. DOI: 10.1090/s0002-9939-1959-0108732-6. URL: https://doi.org/10.1090/s0002-9939-1959-0108732-6.

[45] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[46] D. H. Wagner. "The Riemann Problem in Two Space Dimensions for a Single Conservation Law". In: *SIAM Journal on Mathematical Analysis* 14.3 (1983), pp. 534–559. DOI: 10.1137/0514045. eprint: https://doi.org/10.1137/0514045. URL: https://doi.org/10.1137/0514045.

[47] O. Zanotti, F. Fambri, M. Dumbser, and A. Hidalgo. "Space–time adaptive ADER discontinuous Galerkin finite element schemes with a posteriori sub-cell finite volume limiting". In: *Computers & Fluids* 118 (Sept. 2015), pp. 204–224. ISSN: 0045-7930. DOI: 10.1016/j.compfluid.2015.06.020. URL: http://dx.doi.org/10.1016/j.compfluid.2015.06.020.

List of Figures

List of Tables

List of Algorithms